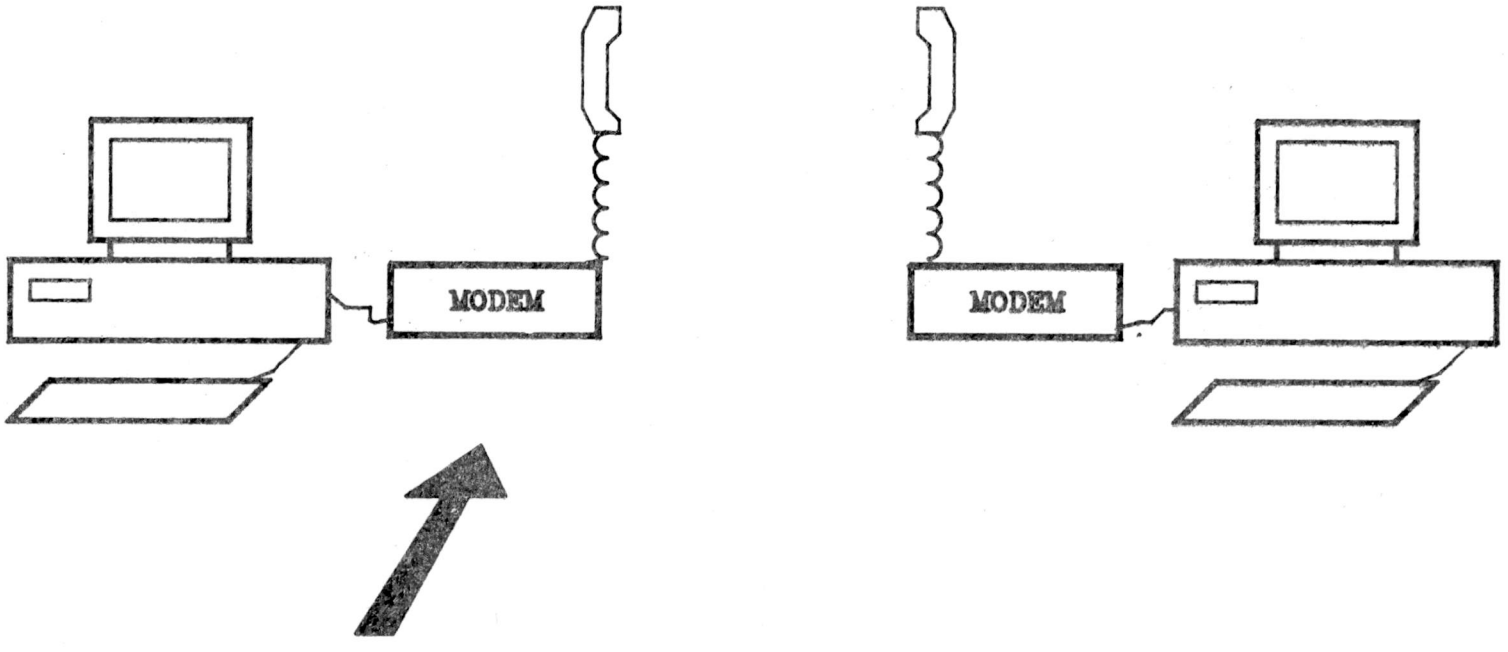
ONTWIKKELING
VAN EEN GEAVANCEERDE MODEM

(van de WIEG tot in de KINDERSCHOENEN)



P.H.G.F. Van de Venne

& M.O. Gomperts

Juli 1987

SUMMARY

Part of the development of an advanced modem is discussed in this report. There is a focus on the software which has been designed during this assignment. The software has a modular design and is written in Omega Pascal.

The hardware is also discussed, since a thorough knowledge thereof is essential for the software production. The hardware has a modular design and is divided into 4 print boards:

- microprocessor board
- modem board
- line interface board
- supply.

The software performs the following functions:

- user interfacing
- data transport
- protection
- Hayes command processor
- elementary telephone functions
- control of FSK and DPSK modem chips.

A general frame has been set up which implements these functions. Much software was designed within this frame, but some work remains to be done.

Samenvatting

In dit verslag wordt een deel van de ontwikkeling van een geavanceerde modem besproken. Het spitst zich toe op de software die tijdens de taak is ontwikkeld. Deze is modulair opgebouwd en geschreven in Omega Pascal.

Ook wordt de hardware besproken, daar een grondige kennis daarvan onontbeerlijk is om enige software te kunnen produceren. De hardware is modulair opgebouwd en opgesplitst in 4 delen, te weten;

- microprocessor kaart
- modem kaart
- lijn interface kaart
- voeding

De software bestaat uit de volgende onderdelen:

- gebruikersinterface
- data transport
- beveiliging
- Hayes commando processor
- elementaire telefoon functies
- besturing van de modem IC's (FSK, DPSK)

Er is een raamwerk opgezet waarbinnen het geheel is ingepast. Dit raamwerk is zoveel mogelijk uitgewerkt en opgevuld. De ontwikkeling van de software is echter niet helemaal afgerond. Derhalve is een deel van het verslag gewijd aan hetgeen nog dient te worden uitgewerkt.

Voorwoord

Als onderdeel van het doctoraal examen (oude stijl) voor de studie voor elektrotechnisch ingenieur (TU Delft) moet een 80 middagen taak worden verricht. In dit kader is gewerkt aan de ontwikkeling van een modem. De taakomschrijving luidde als volgt:

- Test en verbeter de ontworpen hardware van de modem;
- Ontwerp software voor de besturing van de modem, op zodanige wijze dat deze voldoet aan eisen en normen van zowel de CCITT [3] en de PTT [4] als de vakgroep.

Deze werkzaamheden zijn verricht bij de vakgroep Telecommunicatie en Verkeersbegeleidingsystemen (TVS). De mentor van deze opdracht was Ir. J.A.M. Nijhof.

Bij de uitvoering van de taak zijn we veel praktische en organisatorische problemen tegengekomen. Het herkennen en oplossen van deze problemen was voor ons een leerzaam ervaring. Voor praktische ondersteuning gaat onze dank hierbij in het bijzonder uit naar de heren R.F. Luxen, J. van Laren en H.C. Stikkel van de vakgroep. Maar ook alle anderen die ons met raad en daad hebben bijgestaan verdienen een woord van dank.

Tot slot wensen wij onze opvolgers een prettige voortzetting van deze taak en dat dit verslag daarvoor een goede basis vormt.

Inhoudsopgave

Appendices

I	(software)	Omega pascal + FLEX implementatie
II	(hardware)	PIA
III		ACIA
IV		timer
V	telefoonlijn	signalen

Bijlagen

(programmatuur)

(1)	modem (+ declare)	overkoepelend programma
(2)	interf	user interface module
(3)	seslog	session module
(4)	Hayes	Hayes command module
(5)	ACU	telephone interface module
(6)	FSK	FSK modem chip module
(7)	DPSK	DPSK modem chip module
(8)	CLOCK	hardware timer test program
(9)	TSUP + INTRPT	assembler routines

(algemene informatie)

(10)	Hayes code informatie	

Inleiding

In de komende jaren zal het dataverkeer[1] over het telefoonnet steeds verder toenemen. Daar wordt op ingespeeld door over te stappen op een Integrated Services Digital Network (ISDN) [1]. Hierbij zal onder meer direkt digitaal verkeer tussen abonnees mogelijk worden zonder gebruik te maken van analoge componenten. Echter, voordat ISDN mondiaal is ingevoerd zullen nog wel enige decennia verstrijken. (Zo verwacht de Nederlandse PTT pas in 1992 een penetratie van 5% te hebben bereikt). Voor het zover is, zal het nog zeker de moeite waard zijn om aan verbetering van de huidige analoge modems te werken.

Een modem (MOdulator DEModulator) zorgt strikt genomen voor de aanpassing van data aan de transmissieweg en omgekeerd. Hiermee is het een onderdeel van de fysieke laag in het Open Systems Interconnection (OSI) model [2]. De transmissieweg is bijna altijd het telefoonnet zodat deze functie al snel uitgebreid werd met automatische kieseenheden en detektie circuits. Zo ging de modem steeds meer lijken op een volwaardig intelligent randapparaat. Bij randapparaten is de algemene trend het streven naar een grotere lokale intelligentie en flexibiliteit. Voor de modem kunnen naar analogie de volgende eisen worden geformuleerd:

- verschillende modulatie technieken
- verschillende abonneelijnen
- eigen bel/ontvang fasciliteiten
- inzetbaar bij bestaande communicatie pakketten
- locale intelligentie
- mogelijkheid tot service en onderhoud
- beveiliging tegen misbruik

Het laatste punt verdient nog enige toelichting. Bij een toenemend aantal diensten en het verder afnemen van de prijzen van de apparatuur komen er meer modem gebruikers bij. Het risico op misbruik van informatie wordt dan ook groter. Nu is het systeem waar de modem aan gekoppeld is meestal al beveiligd. Toch biedt beveiliging in de modem voordelen, namelijk:

- aangezien de modem het eerste is wat de misbruiker van buitenaf tegen komt, is het verstandig om de beveiliging daar te laten beginnen

---

[1] Met data verkeer is hier geen gedigitaliseerde spraak bedoeld

Hoofdstuk 1

Functionele Omschrijving

§1.1 Inleiding

De modem is een hulpmiddel die communicatie tussen verschillende machines via het telefoonnet tot stand kan brengen. Om hierin te slagen moet de modem aan bepaalde eisen en specificaties voldoen. Deze zijn opgesteld door de CCITT [3]. De modem vervult zijn taak binnen een bepaalde infrastructuur. Ook hieruit vloeien eisen en aanbevelingen voort, die zijn opgesteld door de PTT [4]. De modem wordt ingezet door een gebruiker. Deze heeft ook allerhande wensen ten aanzien van zijn mogelijkheden met de modem.
Zo ontstaat er een heel pakket van eisen, aanbevelingen, specificaties en wensen. Dit pakket geeft het raamwerk aan waarbinnen een modem ontwikkeld dient te worden. Dit geldt zowel voor de bouwstenen, die de fabriekanten aanleveren, als de wijze waarop deze in een systeem worden verwerkt. Ook de opzet van de software wordt er deels door bepaald.

In dit hoofdstuk zal het systeem aan de hand van zijn functies beschreven worden. Voor zover aan deze functies bepaalde eisen ten grondslag liggen worden deze vermeld, of wordt verwezen naar werken waar deze te vinden zijn. Om de beschrijving zo toegankelijk mogelijk te houden, is voor een hoge mate van abstractie gekozen. In de eerste paragraaf wordt zelfs slechts tegen het apparaat aan gekeken.

§1.2 De modem als black box

Van buiten af gezien heeft de modem een aantal aansluitingen, om zijn taak te kunnen vervullen en te kunnen begeleiden. Er kunnen verschillende abonnee lijnen en verschillende terminals op aangesloten worden. Er kan meetapparatuur op aangesloten worden en er kunnen led's aangesloten zijn. Een en ander is geïlustreerd in fig 1.1. Om iets over de hoeveelheid van de aansluitingen te kunnen zeggen is het van belang wat de wensen en eisen zijn die hierop van toepassing zijn. Deze zijn geformuleerd door de gebruiker.

gerealiseerd worden. Werknemers dienen dan eerst met de modem van de werkgever kontakt op te nemen om dan van daaruit de databank te bellen. Alle kosten worden dan door de modem van de instantie afgevangen.
Een bijkomend voordeel van een tweede telefoonlijn is de mogelijkheid om met een (goedkope) modem die een vaste modulatie techniek heeft, een modem met een andere modulatie techniek te kunnen bereiken.
Als een van de bovenstaande configuraties mogelijk moet zijn is een tweede telefoonlijn onontbeerlijk (één telefoonlijn als inkomende lijn en één als uitgaande lijn).
Een derde telefoonlijn maakt het mogelijk verschillende kontakten tegelijk te leggen. De modem wordt daardoor echter onevenredig complex in vergelijking met de plaatsing van een tweede modem, die dan weer meer mogelijkheden biedt.
Wanneer de modem over telefoonlijnen en terminal aansluitingen beschikt kan deze in principe zijn taak verrichten. Om daar op directe wijze enig toezicht op te kunnen houden is het prettig met led's de belangrijke toestanden naar buiten uit te voeren. Om de modem volledig in het vervullen van zijn taak te kunnen volgen is het noodzakelijk een aansluiting te hebben voor een service terminal. De modem kan dan op deze terminal berichten zenden die aangeven waar hij mee bezig is en op welke signalen hij wacht of welke hij verzendt.

Voor service, onderhoud en ontwikkel doeleinden zou het prettig zijn als enkele signaallijnen naar buiten toe worden uitgevoerd. Een dergelijke maatregel kan de levensduur en de 'vriendelijkheid' van het apparaat aanzienlijk verhogen. Zo is testapparatuur eenvoudig op de uitgang aan te sluiten, hoort het zoeken naar printbanen tot de verleden tijd en is de kans op kortsluiting met probe's nihil.

§1.3 De modem functies

De modem als apparaat heeft verschillende taken te vervullen. Daarvoor is de modem voorzien van software en hardware. In de volgende paragrafen zullen deze aspekten nader worden ingevuld.

De systeembeheerder moet de volgende aktiviteiten kunnen uitvoeren:
- toevoegen van gebruikers
- weghalen " "
- wijzigen " " (prioriteit, budget)

Hiernaast moet het ook mogelijk zijn voor de gebruiker om hun eigen password en telefoonnummer (autodial back) te kunnen wijzigen.

§1.3.2 Hardware

Hoe de hardware ingezet wordt om de verschillende functies te vervullen, zal nu beschreven worden. Ook komen de eventuele eisen die van toepassing zijn ter sprake. De hardware functies van de modem zijn in de volgende categorieën onderverdeeld:

- modem communicatie functies
- lijn koppeling
- user koppeling
- interne communicatie
- besturing

In de onderstaande figuur (fig 1.2) wordt dit geïlustreerd.

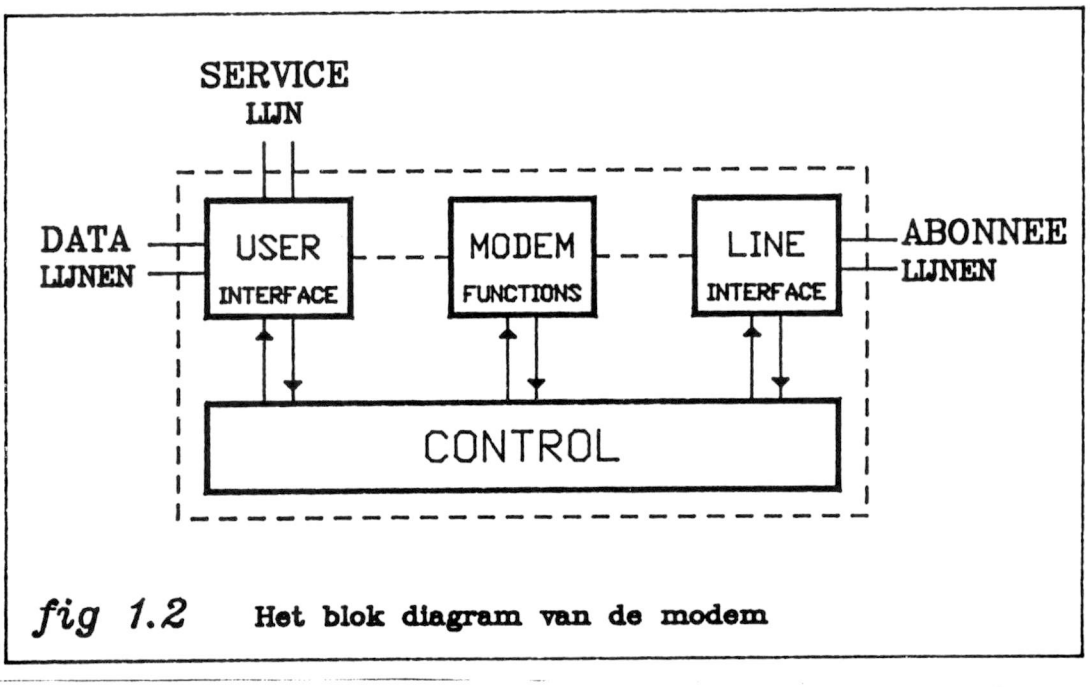


fig 1.2 **Het blok diagram van de modem**

§1.3.2.1 Modem communicatie functies

Er zijn verschillende manieren om data op analoge wijze over het telefoonnet te zenden. Dat zijn onder andere:

- Differential Phase Shift Keyed (DPSK)
- Frequency Shift Keyed (FSK)

detecteren met een optocoupler die in serie in de lijn is opgenomen.

Testen en wachten op kiestoon
Voordat een nummer uitgezonden wordt, moet getest worden of de centrale gereed is voor ontvangst. Dit maakt de centrale kenbaar door een kiestoon te zenden. Behalve de kiestoon kunnen er echter ook andere tonen door de centrale naar de modem verzonden worden, zoals bezettoon of congestietoon. Het is echter zelfs mogelijk dat er helemaal niets door de centrale verzonden. Om de tonen te kunnen detekteren kan er op de uitgang van de interface een toon detektor worden aangesloten. Al naar gelang van hetgeen de detektor detekteert wordt het nummer verstuurd of de lijn verbroken en eventueel opnieuw geprobeerd.

Nummer uitzenden
Het nummer kan op twee wijzen uitgezonden worden. Dit kan met behulp van pulsen of tonen. Als met pulsen wordt uitgezonden wordt met een pulsrelais de lijnstroom onderbroken volgens daarvoor opgestelde regels (zie Appendix V). Bij toonkiezen wordt gebruik gemaakt van een DTMF (double tone multi frequency) generator. Deze zendt twee tonen tegelijk uit die gezamelijk een nummer voorstellen. Het functioneren van deze generator is ook aan regels gebonden (zie Appendix V,[4]). De besturing van het pulsrelais en de DTMF generator wordt verzorgt door de microprocessor.

Testen van de toestand van de gebelde
Na het zenden van het nummer moet de toestand van de gebelde getest worden. Er kunnen zich ook nu weer verschillende toestanden voordoen, die ieder met een bepaald signaal worden aangegeven. Zo kan de gebelde gewekt worden. Dan moet er gewacht worden in de hoop dat de aanvraag tot een gesprek geaccepteerd wordt.
De gebelde kan ook bezet zijn of er kan congestie optreden. Al deze toestanden van de gebelde worden aangegeven met toon patronen. Deze tonen zijn ook te detekteren met een toon detektor. De microprocessor kan dan bepalen van welk ritme sprake is.

Gesprek aanvraag honoreren
Los van het bovenstaande kan het ook voorkomen dat een derde de modem tracht te bereiken. Om dit mogelijk te maken moet deze in staat zijn de belstroom te detekteren. Dit is net als de lijnstroom een energetisch signaal. Ook voor de belstroomdetectie kunnen dus de optocouplers worden ingezet. Als een

verschillende onderdelen naar de aansluitingen van de modem. Ten tweede zijn er de besturingslijnen van de besturingseenheid. Deze vormen het grootste gedeelte van de interne communicatie.

§1.3.2.5 De besturingseenheid

Voor de besturing zal gezien de complexe eisen en de eis van doorzichtigheid van het ontwerp voor een microprocessor configuratie gekozen worden. De programma code wordt in ROM opgeslagen, data in RAM. Aangezien veel gegevens niet vluchtig mogen zijn (password beveiliging, etc.) is ook een backup voeding voor de RAM noodzakelijk. Gezien de strikte eisen die worden gesteld aan de timing van signalen zal ook een timer noodzakelijk zijn. De besturing wordt compleet met enkele ondersteunings circuits zoals selektoren en dergelijke.

Hoofdstuk 2

Implementatie

§2.1 Inleiding

De in hoofdstuk 1 beschreven functionele specificatie van een modem is omgezet in een praktisch ontwerp. In dit hoofdstuk wordt dit ontwerp besproken. Eerst de hardware, vervolgens de software.

§2.2 Hardware implementatie

De hardware is door derden ontwikkeld. Deze moest getest worden, en daarna moest er software voor geschreven worden om het geheel naar behoren te laten functioneren. Bij het vervullen van deze opdrachten is een grondige kennis van de hardware noodzakelijk. In deze paragraaf wordt een overzicht van de hardware en de documentatie daarover gegeven.

Het uiteindelijke concept, is ondergebracht op drie printkaarten, te weten:

- microprocessor kaart
- modem kaart
- lijninterface kaart.

Later moet hier nog een vierde kaart voor de voeding aan toe worden gevoegd. In de testopstelling werd gebruik gemaakt van een externe voeding.

§2.2.1 De microprocessor kaart

De microprocessor kaart is een standaard kaart die ontwikkeld is door de vakgroep TVS. De gebruikte microprocessor is een 6809. Daaromheen treffen we ACIA's, PIA's en een timer aan. Er bestaat de indruk dat deze kaart niet geheel voldoet om de modem te sturen, daar de timer op een te hoge frequentie draait en daardoor geen al te lange tijden kan meten. Daar deze kaart een standaard model van de vakgroep is, is voldoende informatie [5] en kennis aanwezig. Het wordt dan ook overbodig geacht hier diep op deze kaart in te gaan.

Wel is hier enige kennisoverdracht op z'n plaats. Allereerst is uitgezocht welke tijden van belang zijn, die de timer onder toeziend oog van de microprocessor moet meten, om communicatie met het telefooncircuit

mogelijk te maken. De tijden zijn in appendix V opgenomen en afkomstig uit de eisenlijsten [4] van de PTT.

Bovendien zijn met moeite de timer, de PIA's en de ACIA's in gebruik genomen. De op deze kaart gebruikte timer is de 6540. voor informatie omtrent deze bouwsteen wordt verwezen naar de datasheet. Wordt de timer voor het eerst in gebruik genomen, dan wordt aangeraden appendix IV te raadplegen. De PIA is een Pheriferal Interface Adapter. Het in dit concept verwerkte exemplaar is de MC-6821. De PIA's functioneren als buffers voor de datalijnen en maken het mogelijk in verschillende richtingen te lezen en te schrijven. Voor meer informatie omtrent de PIA wordt verwezen naar de datasheet. Moet een PIA in gebruik worden genomen dan wordt aangeraden appendix II te raadplegen. De ACIA is een Asynchrone Communication Interface Adapter. Het in dit concept verwerkte type is de G-6551. Deze verzorgt de conversie van seriële data naar parallelle data en vice versa. Voor meer informatie omtrent de ACIA wordt verwezen naar de datasheet. Moet een ACIA in gebruik worden genomen, dan wordt aangeraden appendix III te raadplegen.

§2.2.2 De Modemkaart

De modem kaart ziet eruit als figuur 2.2 toont. Het bijbehorende schema is in figuur 2.1 opgenomen. De gebruikte IC's zijn de volgende:

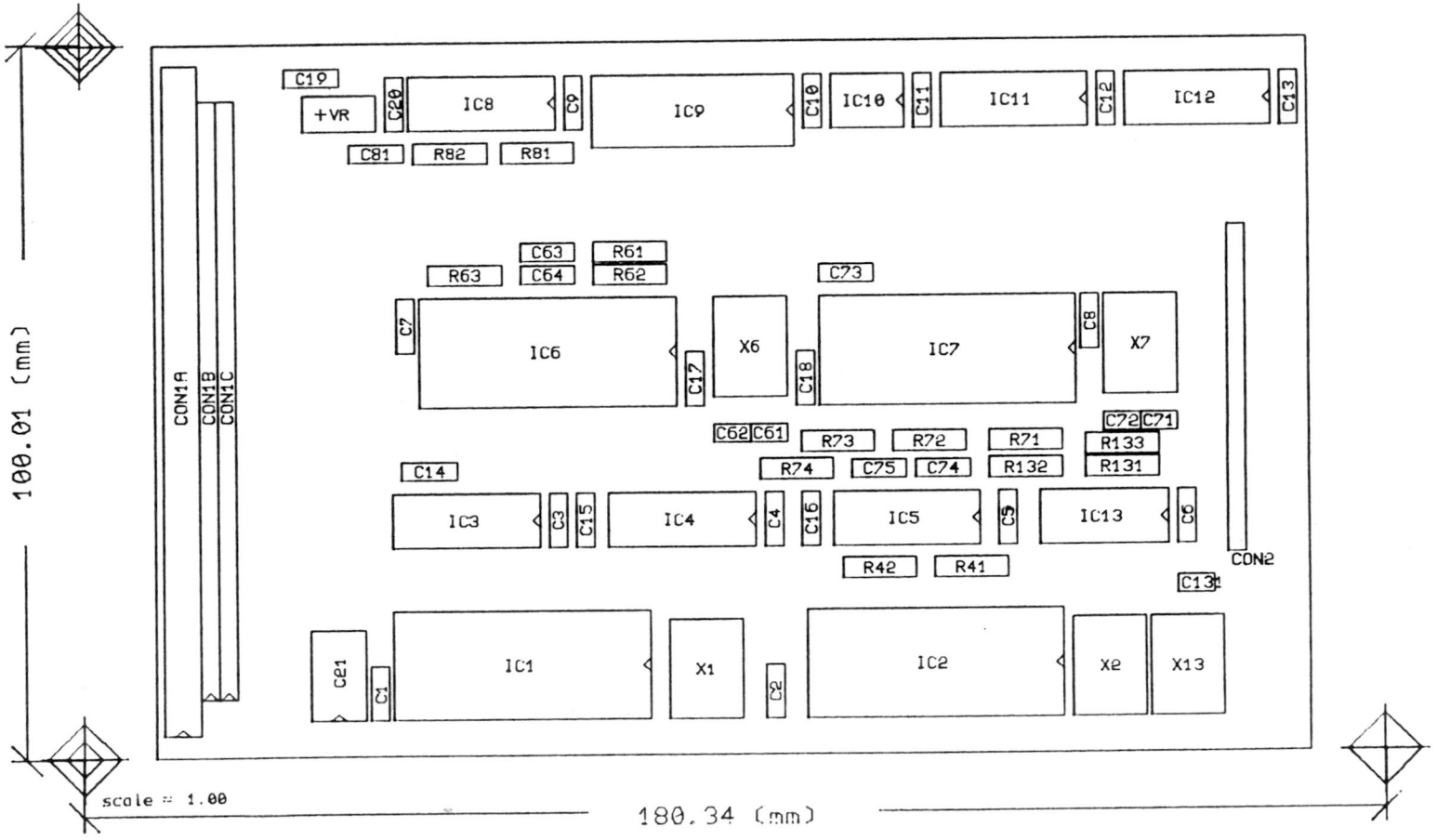
- IC 1,2 : G-6551 ACIA
- IC 3,4,5 : HEF-4053 3-voudig 2-kanaals analog (de)multiplexer
- IC 6 : AM-7911 FSK-modem IC
- IC 7 : EFG-7515 DPSK-modem IC

- IC 8 : TP-5089 DTMF generator
- IC 9 : M-957 DTMF receiver
- IC 10 : M-980 dial tone detector
- IC 11,12 : NE-590 addressable peripheral driver

Met PIA 2 van de microprocessor kaart zijn de beide modem IC's in te stellen. Via de ACIA's is de communicatie te realiseren. Hoe dit precies te verwezenlijken is, staat beschreven in de datasheets van de IC's.

Met PIA 1 van de microprocessor kaart kunnen de resterende bouwstenen bestuurd en afgetast worden. Uitleg omtrent de handelingen om dit op de juiste wijze te laten plaatsvinden vindt men terug in de datasheets.

fig 2.2



CED-871017 25-Feb-87 871017

CEDZK BPL=1 SEG=8

39

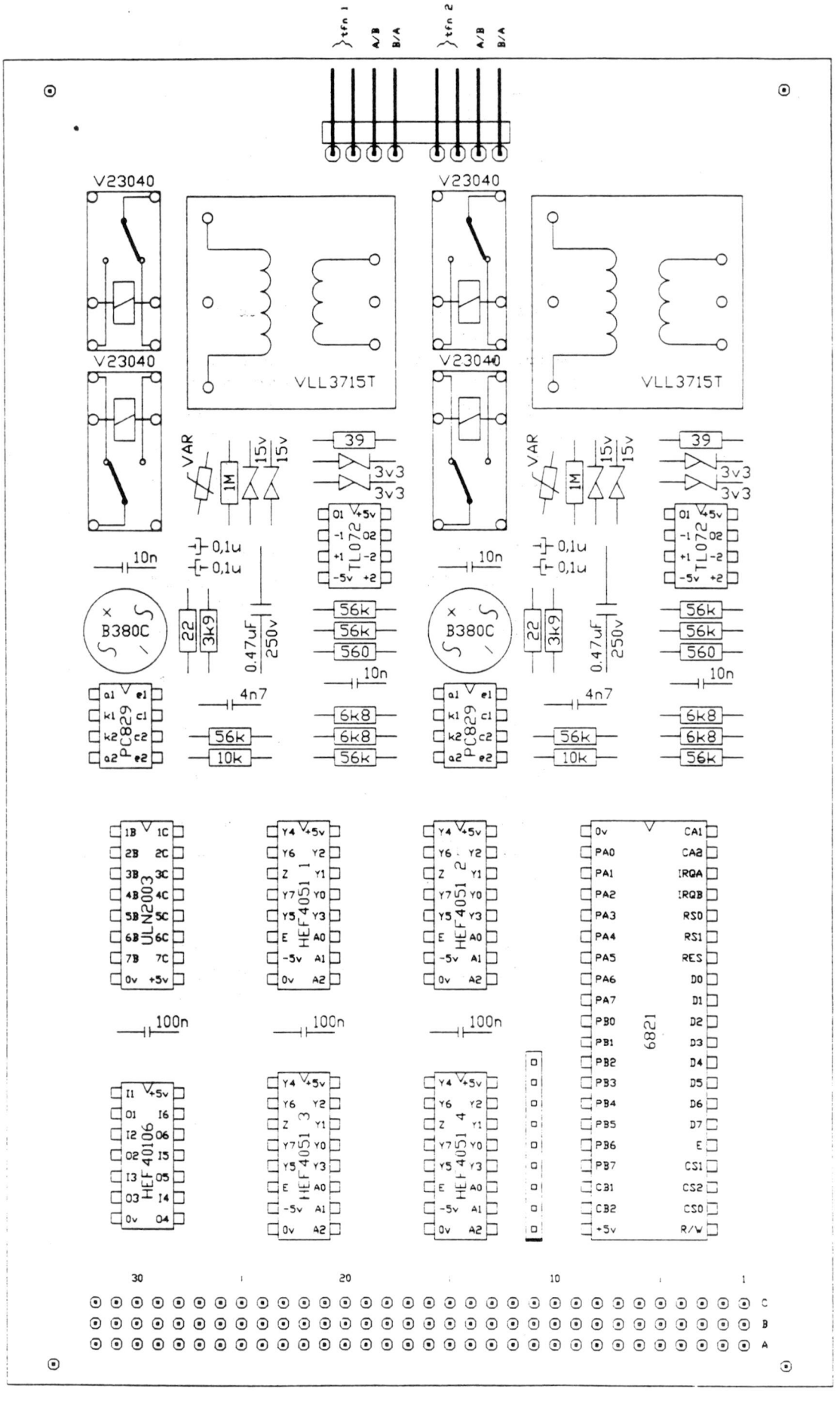

tfn 1
A/B
B/A
tfn 2
A/B
B/A
V23040
V23040
V23040
V23040
VLL3715T
VLL3715T
VAR
1M
15v
15v
39
3v3
3v3
TL072
10n
0,1u
0,1u
B380C
22
3k9
0.47uF
250v
56k
56k
560
10n
PC829
4n7
56k
10k
6k8
6k8
56k
ULN2003
HEF4051 1
HEF4051 2
HEF4051 3
HEF4051 4
HEF40106
100n
6821
0v
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
CB1
CB2
+5v
CA1
CA2
IRQA
IRQB
RS0
RS1
RES
D0
D1
D2
D3
D4
D5
D6
D7
E
CS1
CS2
CS0
R/W
30
20
10
1
C
B
A

fig 2-4

§2.3 Software implementatie

De software zal eerst van de gebruikers kant worden besproken. Vervolgens wordt op de interne software opbouw ingegaan.

§2.3.1 De software gezien vanuit de gebruiker

Om de toegang tot het systeem te kunnen bespreken, moet onderscheid worden gemaakt tussen een:

- local user (normale gebruiker)
- remote user (gebruiker op afstand, die vanuit een andere modem toegang wenst).

In rust toestand staat de modem te wachten totdat een local- of een remote user zich aanmeldt (access). De local user wordt herkend door het indrukken van een toets op de terminal. De remote user wordt herkend door het ontstaan van een belsignaal op één van de twee telefoonlijnen en dan alleen indien de modem op autoanswer is ingesteld.
Nadat een gebruiker is herkend wordt de user interface (bijlage 2) ingesteld en moet hij zich identificeren. Dit identificeren gebeurt in session (bijlage 3).
Als er sprake is van een local user schakelt de modem naar commando mode, bij een remote user naar data mode.
Het bovenstaande is in figuur 2.5 geschetst.

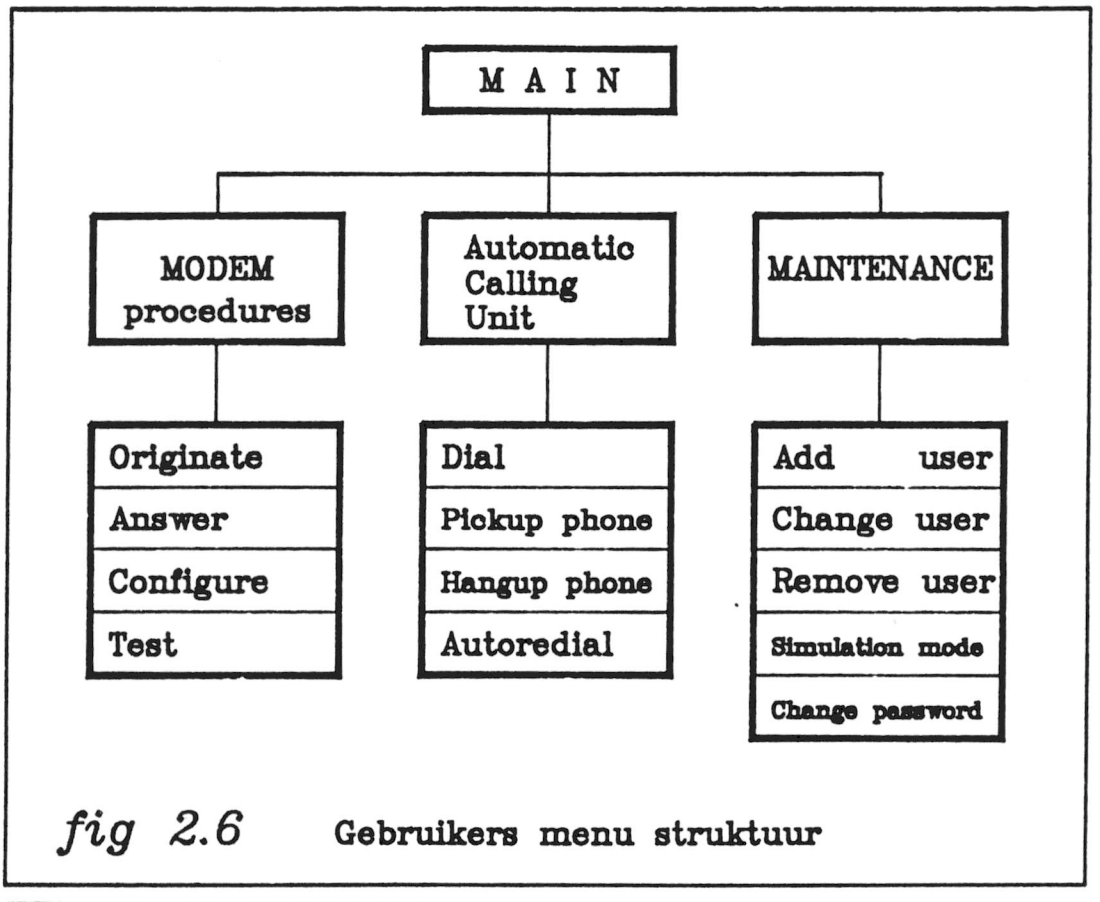


fig 2.6 **Gebruikers menu struktuur**

Zoals uit de figuur blijkt is de software van de gebruiker onderverdeeld in drie stukken:

- Automatic Calling Unit
- Modem procedures
- Maintenance

Deze stukken zullen nu achtereenvolgens worden besproken.

§ 2.3.1.1. Automatic Calling Unit

Voor het tot stand brengen van een verbinding is het dial commando beschikbaar. Er kan een telefoonnummer worden ingetoetst, eventueel met een 'P' of 'T' erdoor gemengd voor puls- of toon kiezen. (Als geen 'P'/'T' wordt gebruikt wordt puls als default gekozen).
Indien de verbinding niet kon worden opgebouwd kan met behulp van het autoredial commando het laatst gekozen nummer nog een keer worden geprobeerd.
De Hangup- en Pickup phone commando's zijn bedoeld om handmatig de modem van en op de lijn te kunnen schakelen.

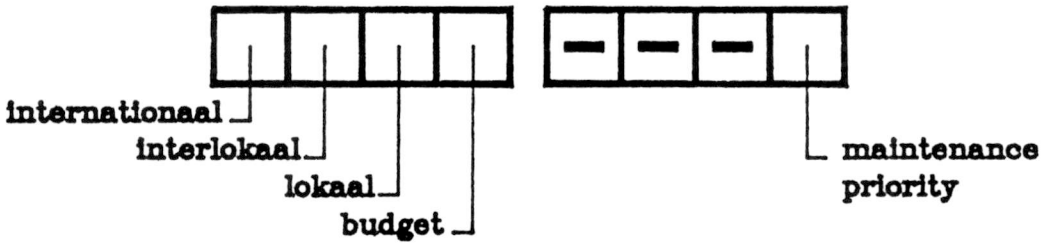


fig 2.7 **Gebruiker prioriteits verdeling**

Voor het bekijken van de modem toestand (tijdens de ontwikkel fase en tijdens onderhoud en reparatie) is het 'Simulation mode' commando bedoeld. Als Simulation mode is geaktiveerd worden regelmatig berichten naar de service terminal gestuurd over de status van de modem. (De service terminal moet wel zijn aangesloten).

§2.3.2 De software opbouw

In figuur 2.8 is de algemene software struktuur met de verschillende modules weergegeven. Het programma dat alles omvat heet "modem" (bijlage 1).
In het nu volgende zal alle geschreven software worden besproken, te beginnen met het hoofdprogramma, 'modem.txt'.

De Maintenance procedure schrijft status informatie naar de service terminal, als simulatie mode is geaktiveerd. Om run tijd te sparen bij het doorgeven van paramters is de informatie gecodeerd in een request byte.

Procedure wait wacht een aantal milliseconden. Dit wachten gebeurt door te incrementeren tot de gewenste tijd (time) is bereikt. Van te voren is de procedure geijkt met de constante TimeScaleFactor. Door ontwikkel problemen met de hardware timer en de getestte nauwkeurigheid van de huidige wait procedure is deze nog niet vervangen door een bij de hardware timer behorende procedure.

Om een timeout te kunnen detekteren is de volgende struktuur bedacht:

```
sec:=30;
REPEAT
  <statement>
  CountDown
UNTIL (Eind Conditie) OR (TOF);
```

Hierbij is 'sec' het aantal seconden voor de timeout vlag (TOF) wordt gezet als de eind conditie niet wordt bereikt. Variabele sec wordt door CountDown gedecrementeerd. Ook hier is geen gebruik gemaakt van de hardware timer en geldt wat bij procedure wait is gezegd. De tijdconstante is TimeoutFactor. Indien een 'IF' statement met drie opdrachten wordt toegevoegd in de REPEAT lus. ontstaat een gemeten toeneming van de timeout tijd van maximaal 10%. Mèt de harware timer zou de struktuur als volgt kunnen zijn:

```
TimeOut(30);
REPEAT  ;  UNTIL (Eind Conditie) OR (TOF);
```

Als laatste service procedure is delete genoemd. Deze procedure bestaat onder gelijke vorm al voor veel andere versies van pascal (Turbo,Propas, etc.). Met deze procedure kan in een string variable een aantal (cnt) karakters op een bepaalde plaats (pos) worden weggehaald.

Voorbeeld bij gebruik:

```
naam:='PQRST';
delete(naam,3,2);
==>>naam=PQT
```

Procedure Take coördineert het binnenhalen van commando's. De gebruiker kan ofwel een keuze uit een menu maken òf een Hayes commando geven. Na het uitvoeren van een Hayes commando wordt procedure Take niet verlaten. In het geval van een keuze uit een menu wordt eerst gekeken of het gewenste getal is toegestaan (≤max). Vervolgens wordt bij een correcte keuze variabele choice ingesteld en de procedure verlaten.

Via de led procedure kunnen 16 leds worden aan- of uit gestuurd. Een voorbeeld: led(7,on). De led procedure wordt nog niet gebruikt maar kan in de toekomst voor onder andere "carrier detect" en "connect" indicatie worden gebruikt.

Het laatste onderdeel van de interf.txt module is de procedure DataMode. Het is een procedure voor het datatransport van local (altijd ACIA1) naar remote (gemoduleerd) en vice versa. De parameters worden gevormd door de ACIA's die voor zenden en ontvangen zijn ingesteld. (Alleen bij half duplex mode, bij FSK modulatie, is ACIA3 benodigd. In alle andere gevallen wordt ACIA4 gebruikt voor zender èn ontvanger). Voor zowel zender als ontvanger wordt een gelijke procedure gevolgd. Eerst wordt gecontroleerd of er een karakter is binnengekomen. Is dit het geval dan wordt het karakter aan de partner ACIA (is dus meestal dezelfde) doorgegeven. Als laatste wordt gecontoleerd of de data carrier is weggevallen. Het wegvallen van de data carrier heeft nog geen consequentie, het wordt alleen aangegeven op de service terminal. Ook de led indicatie is nog niet verwezenlijkt.

§2.3.2.3 seslog.txt

Module "seslog.txt" (bijlage 3) bevat alle gebruikers-bestand functies. Het moduul zelf bestaat uit slechts één procedure: "Session". Session bestaat uit de volgende service fasciliteiten:

- initial
- login
- add
- remove
- change
- password.

Om deze fasciliteiten te kunnen bieden zijn een aantal locale procedures geschreven. Bij de nu volgende bespreking zullen deze vanzelf ter sprake komen.

"Initial" wordt in het hoofdprogramma (modem.txt) bij de initialisaties aangeroepen. Indien de systeembeheerder ("root") niet meer bestaat wordt het

Als er gebruikers bestaan kan bij SelectUser een gebruiker worden geselecteerd. Een vraagteken geeft een lijst van de bestaande gebruikers (althans hun identificatie).

Procedure Edit functioneert als een interface tussen de gebruiker en het session moduul. Afhankelijk van de gewenst service (login, change, add, password) moet bepaalde informatie wel of niet worden gegeven of gevraagd. Indien later voor bijvoorbeeld de tarifering verdere informatie uitwisselingen moeten plaatsvinden zal procedure Edit moeten worden uitgebreid.

§2.3.2.4 Hayes.txt

Module Hayes.txt (bijlage 4) bestaat uit procedure HayesCmd. HayesCmd is een Hayes commando processor (bijlage 10, [6]) die door veel software paketten wordt gebruikt. Tot nu toe is de struktuur opgezet en zijn alleen de commando's "A,B,C,D,F,H" geimplementeerd.
Als eerste worden de herkenningsletters van een Hayes commando ("AT") en alle spaties geëlimineerd. Vervolgens worden alle kleine letters naar hoofdletters overgezet. Dan wordt het gekeken of het commando bestaat. Is dit het geval, dan wordt het commando uitgevoerd.

§2.3.2.5 ACU.txt

De telefoon functies zijn ondergebracht in module "ACU.txt" (bijlage 5). Het moduul bestaat uit de volgende procedures:

- Pickup — modem op een lijn schakelen
- Hangup — " van " " afschakelen
- LineState — lijn status na het opschakelen
- Condition — lijn status voor het opschakelen
- CheckNumber — controle op geldige kiesinfo
- DialNumber — zenden van kiesinfo (puls/toon)

Met Pickup kan de modem op een lijn (1 of 2) worden geschakeld. Dit wordt gedaan door bij de gewenste lijn:

- puls relais + kortsluit relais te bekrachtigen
- "connect" led aan te doen
- SetUpTime te wachten, totdat de lijn in rust is
- lijn detektie vlag te resetten
- kortsluit relais af te schakelen

van de optocoupler. Is geen puls ontvangen dan is de lijn vrij.

DialNumber zorgt voor het uitzenden van kiesinformatie. Puls en toon kiezen kan door elkaar worden gebruikt. Deze verschillende kiesmethoden zijn in de procedures DialP en DialT ondergebracht. Door het gebruik van komma's kunnen pauses worden ingelast van 2 seconden per komma.

§2.3.2.6 FSK.txt

Module "FSK.txt" (bijlage 6) bevat de procedure "FSKModem" die voor de juiste aansturing van het FSK modem IC moet zorgen. Evenals bij module seslog (zie §2.3.2.4) bestaat deze procedure uit een aantal service faciliteiten (genoemd in MODEMSERVICE van de globale declaraties). Deze zijn in principe: .

- originate
- answer
- relay
- configure
- show
- test
- init
- connect
- disconnect
- transmit.

Van al deze gedefiniëerde service faciliteiten zijn tot nu toe alleen show, configure, en originate uitgewerkt.

Service "show" stuurt informatie over de gekozen FSK chip instelling naar de gebruiker. Deze instelling is in het gebruikers record USERTYPE (zie global declarations) terug te vinden. Wat betreft de modem parameters zijn de volgende variabelen aanwezig:

- transmission (FSK/DPSK) vb. transmission:=FSK
- protocol1 (CCITT/Bell) vb. protocol1:='C'
- duplex (Full/Half) vb. duplex:='F'
- baud (baudrate) vb. baud:=300
- special (soft turn off, etc.)(zie fig. 2.10)
- encoded: code die naar het FSK IC wordt gestuurd.

§2.3.2.8 Interrupt handling

Voor de implementatie van de hardware timer is een interrupt handler noodzakelijk. In principe zijn twee (kleine) assembler routines nodig en een (in pascal geschreven) initialisatie en kern procedure.

Omdat de timer niet direct noodzakelijk was voor de goede werking van het hoofdprogramma is een onafhankelijk programma ("clock.txt") geschreven om de timer besturing te ontwikkelen en te controleren. Dit programma bestaat uit de volgende delen:

- clock.txt hoofdprogramma (pascal)
- TSUP.txt timer interrupt handler (assembler)
- INTRPT.txt interrupt vector instelling (assembler)

clock.txt

Het hoofdprogramma (clock.txt, bijlage 8) bestaat uit de volgende procedures:

- InitTimer
- Timer
- wait
- TimeOut

Bij procedure "InitTimer" wordt de gebruiker gevraagd om de juiste tijd in te toetsen. Vervolgens worden de registers van de hardware timer ingesteld en de timer gestart. (Hoe de timer moet worden geprogrammeerd is in Appendix IV beschreven).

Procedure "Timer" is de "echte" interrupt routine. Er wordt iedere halve seconde door de hardware timer een interrupt gegeven. Bij iedere interrupt wordt de halve-seconden-teller ("HS") geïncrementeerd. Afhankelijk van de tijd veranderen ook de andere tijdregisters.

De procedures "wait" en "TimeOut" zijn procedures die voor het in §2.3.2.1 beschreven modem.txt hoofdprogramma kunnen worden gebruikt. De manier waarop dit kan gebeuren is in die paragraaf beschreven.
Procedure wait wacht een aantal milliseconden. Om dit te bereiken wordt de huidige timer-register-stand bekeken en een eindwaarde bepaald. Dan wordt gewacht totdat deze eindwaarde is bereikt.
Bij procedure TimeOut wordt op soortgelijke manier als bij wait een eindstand (in seconden) bepaald. Ook de timeout vlag ("TOF") wordt gereset. Vervolgens wordt de procedure verlaten; TOF wordt gezet door procedure Timer.

Hoofdstuk 3

Taakverloop

§3.1 De voorgeschiedenis

Met drie vrienden hadden wij het idee opgevat om een klein netwerkje op te zetten. Ieder stond namelijk op het punt een computer te kopen. Voor de communicatie onderling wilden we modems inzetten. Met de gedachte de modems zelf te bouwen, zijn wij bij de vakgroep TVS langs gegaan. Daar kregen wij te horen dat men zelf met de ontwikkeling van een geavanceerd modem bezig was. Het projekt boeide ons zozeer dat wij in de vorm van deze taak aan de ontwikkeling van de modem hebben meegewerkt.

§3.2 Aanvang van de taak

Op het moment dat wij aan onze taak begonnen was de hardware van de modem samengebracht in een groot concept. Vanwege de omvang was dit ondergebracht op twee prints. Het enige wat wij wisten was wat de modem moest doen. Grote vragen waren echter, hoe de modem zijn taken zou vervullen en hoe de modem de verschillende taken aan zijn IC's zou delegeren. Wij zijn dan ook begonnen met het verzamelen van alle mogelijke informatie en documentatie over de modem. Met name enkele artikelen uit EDN [10] bleken relevante informatie te bevatten. Na bestudering van de verzamelde informatie werd het mogelijk een beeld te vormen over de juiste werking van de modem.

§3.2.1 De modem en zijn testopstelling

Om met de vergaarde kennis over de modem de hardware te kunnen testen en de software te ontwikkelen, is deze aan een computer aangesloten. Dit om de besturing, die uiteindelijk door een microprocessor wordt verzorgt, tijdens de ontwikkeling te kunnen simuleren. De simulatie werd gerealiseerd met een 6800 microprocessor systeem. Deze werd met een kabel op de data- en adreslijnen van de modem aangesloten. Deze aansluiting introduceerde echter storingen waardoor data en adressen vermirkt werden. De modem reageerde op testprogramma's dus niet zo als wij verwachten. De

§3.3 Het tweede begin

Naar aanleiding van onze conclusies en de ideëen die bij onze mentoren leefden, is de opzet van de modem gewijzigd. Deze is modulair opgezet en opgedeeld in een microprocessor kaart, een modem kaart en een interface kaart. Bovendien is ons een FLEX systeem (met een 6809 microprocessor) ter beschikking gesteld, waarop het mogelijk was in Omega Pascal te programmeren.

Het principe van de werking van de modem bleef hetzelfde. Wij moesten echter opnieuw thuis raken in het concept. Zo bleken in het nieuwe concept enkele andere IC's verwerkt te zijn, te weten, de ACIA 6551 en de timer 6840. Daarnaast moesten we leren omgaan met pascal, met name wat betrefd het aansturen van de hardware vanuit deze hogere programmeertaal.

Opnieuw is met testen begonnen. Ook nu weer aan de hand van kleine programma's. Bij het opstellen van deze programma's is meteen rekening gehouden met de mogelijkheid deze in het grote geheel te verwerken. De tests die zijn uitgevoerd worden nu in de volgende paragraaf besproken.

§3.3.1 Hardware tests

Allereerst probeerden wij de led uitgangen aan te sturen. Dit gaf geen problemen, mede door de al opgedane ervaring met de vorige modem. Het was echter prettig met deze relatief eenvoudige test te beginnen om de combinatie van modem met ontwikkelsysteem te testen. Met name het gebruik van pascal bij het aansturen van hardware (zoals bijvoorbeeld PIA's) was van belang.

Daarna hebben wij ons op de interface kaart gericht. Allereerst probeerden wij een telefoonlijn aan te schakelen. Daarvoor dienden we de relais te kunnen bereiken. Dit lukte aanvankelijk niet. Nadat wij er van overtuigd waren dat de software foutloos was hebben we de hardware onder de loep genomen. Het bleek dat de relais aangestuurd werden door ze met aarde te verbinden. Het vaste kontakt moest dus verbonden zijn met de 5 volt lijn. Per abuis waren de relais echter aan de aardlijn aangesloten. Aansturing was daardoor niet mogelijk.

Nadat dit gewijzigd was kon de modem kontakt leggen met de lijnen. Vanaf dat moment werd het interessant de informatie die daarop aanwezig kon zijn, te kunnen 'horen'. Daarvoor moest de toondetektor ingezet worden.

(iedere halve seconde) met een interrupt even uit het hoofdprogramma te stappen en de counters van de timer naar wens te incrementeren. Toen de timer op interrupt basis getest werd bleek deze onregelmatig te lopen. Het bleek dat FLEX ook interrupts genereerde waarop onze timer ook reageerde. De FLEX interrupts zouden van de FLEX timer af komen. Daar tijdelijke verwijdering het systeem geen schade zou toebrengen dachten wij het probleem daarmee op te lossen. Het hielp helaas niet veel, daar andere FLEX onderdelen ook interrupts afgaven. Er is toen een stukje software geschreven om de interrupts scheidt. Dit is enkel nodig in de testopstelling en kan in het uiteindelijke concept achterwege worden gelaten.

Behalve signaalbewaking kan met behulp van de timer ook een nummer in het juiste ritme uitgezonden worden. De tijden die daarvoor van belang zijn, zijn ook ondergebracht in Apendix V. Er is een programma geschreven waarmee het mogelijk is de DTMF-generator aan te sturen, en zo het nummer met behulp van tonen uit te zenden. Er is echter ook een programma geschreven waarmee het pulsrelais aangestuurd kan worden, om het nummer op de conventionele wijze uit te zenden. De gebruiker kan zelf kiezen of hij het nummer in toon of puls wil zenden.

§ 3.3.2 Software ontwikkeling

Nu dan waren alle essentiële telefoonfuncties uitvoerbaar. Wij hebben ons toen een tijd gericht op de software die de modem voor de gebruiker op een acceptabele wijze bruikbaar maakt. Op deze software werd dieper ingegaan in hoofdstuk 2. Globaal kan gesteld worden dat deze software het volgende op 'vriendelijke' wijze mogelijk maakt:

- de communicatie van de modem naar de gebruiker
- de communicatie van de gebruiker naar de modem
- de led aansturing
- de loginprocedure's
- de aansturing van de menu's
- de prioriteitsregeling

Net zoals voor de hardware is ook voor de software voor een modulair opbouw gekozen. Behalve de gebruikelijke voordelen, die voor modulair werken gelden, verwachtten wij een aanzienlijke tijdwinst bij het ontwikkelen.
Het Pascal op de FLEX had de mogelijkheden om modulair te programmeren. Op de afdeling was hiermee echter geen ervaring. Ook was de documentatie summier. Met name door de ervaringen die wij met Modula-2 hadden

aan medestudenten traden er echter weer storingen op. Ook nu konden wij de oorzaak niet in de software ontdekken. Zelfs in de hardware scheen alles in orde. Wij hebben toen de modem op afstand door een ander ontvangen. De modem die wij allereerst gebruikten kon zenden en ontvangen tegelijk. Andere modems die ons ter beschikkig stonden konden echter alleen zenden of ontvangen. Dit is toen allebei apart uitgeprobeerd, en functioneerde. Onze conclusie was dus dat de modem waar wij aanvankelijk mee werkte stuk was gegaan.

Daar onze tijd, die voor een taak als deze ter beschikking staat, al ruimschoots overschreden was, er bij ons al was aangedrongen verslag uit te brengen en we constateerden dat er waarschijnlijk weer een externe fout was opgetreden, besloten wij de opdracht te beëindigen.

Hoofdstuk 4

Conclusies en Aanbevelingen

4.1 Conclusies

(1) De taak waarvoor 80 manuren (ofwel 2 maanden full-time) staat, geeft een grote indruk van wat er zoal komt kijken bij het verrichten van projekten in de R&D sector. Een algemeen probleem was dat de benodigde tijd de geplande tijd overtrof. Zo zijn we dan ook geen twee maar vier maanden bezig geweest. Tijdens de uitvoering van de taak zijn de volgende aspecten aan de orde gekomen:
- literatuur onderzoek (Zowel algemene literatuur over modems als datasheets zijn bestudeerd)
- taak organisatie (Tijdplanningen, volgorde, verdeling, overleg, etc.)
- testen van hardware (Metingen en het schrijven van testprogramma's)
- ontwerpen van software (Modulair programmeren in Pascal)
- rapportering (Bijhouden van een logboek en het schrijven van dit verslag)
- voordracht (Deze moet nog worden gehouden)

(2) Een microprocessor bestuurde modem biedt voordelen:
- flexibiliteit
- software instelbare parameters
- tijdbesparend

(3) Het modulair opzetten van zowel de hardware als de software is functioneel (Tijdwinst bij de ontwikkeling en onderhoud door doorzichtigheid)

(4) Het in een hogere programmeertaal schrijven van programma's heeft de volgende voordelen:
- snel complexe programma's kunnen schrijven
- goede programma structuur
- goed leesbaar
- minder systeem afhankelijk
- minder kans op dramatische fouten

literatuurlijst

1. ISDN: the Commercial Benefits
   (R. Kee, D. Lewin : Ovum ltd. 1986)

2. Computer Networks
   (A.S. Tanenbaum : Prentice Hall 1981 : 15..21.109..110)

3. CCITT aanbevelingen V.21 (1980)
   V.22 (F. VIII.I : 76..90)
   V.23 (F. VIII.I : 91..97)
   V.24 (F. VIII.I : 98..111)
   V.25 (F. VIII.I : 112..118)

4. PTT eisenlijsten
   Htf 02-20 apr 1985 no.7 : autom. beantwoord.apparaten
   Htf 02-28 mrt 1976 : autom. kiesapparaten
   Htf 02-29 mei 1985 no.7 : oproepdecoktie
   Htf 02-51 dec 1976 no.2 : kiestoondetektors
   Htf 02-68 mrt 1980 no.4 : toonfrequentiekiezen

5. Datasheet 6809 SBCPU
   (vakgroep TVS, TU Delft 1987)

6. Handleiding bij modem Discovery 1200/2400

7. Omegasoft Pascal version 2 language handbook
   (Omegasoft nov 1982)

8. Elektronische Automatische Telefonie (1-100)
   (J.L. de Kroes e.a. : collegedictaat dec 1986 : 214..287)

9. Kostenteller Detektor
   (J. Verstraten : Radio Bulletin juni 1987 : 45..47)

10. EDN modem artikelen
    - Low cost modems
      (Jim Lange : EDN mei 1984 : 187..200)
    - Modem ICs
      (William Twaddell : EDN mrt 1985 : 159..172)
    - Use controller and chips to design an intelligent modem
      (R.Chirayil e.a. : EDN mei 1985 : 213..220)

APPENDIX I

Omega Pascal op FLEX

In het hierna volgende wordt beschreven:
1) wat de kracht is van de Omega Pascal compiler
2) hoe onder FLEX files kunnen worden gemanipuleerd
3) compilatie: van pascal source naar object code
4) modulair programmeren
5) assembly interfacing

1) Omega Pascal

De Omega Pascal compiler genereert code voor een 6809 microprocessor. De compiler heeft een aantal voordelen die hem interessant maken om te gebruiken.
Zo is hij geschikt voor modulaire programmering en assembly interfacing. Verder wordt er efficient met code omgesprongen (zo leidde een programma van 50 kbytes text tot 4 kbytes code).
Een belangrijk voordeel is de mogelijkheid om een variabele op een vastgesteld adres te plaatsen. Zo kan bijvoorbeeld een PIA,ACIA of timer zonder ingewikkelde instrukties worden bestuurd. Een voorbeeld:

```
(declaratie)   VAR  PIA : byte at $FBE8;
(gebruik   )   PIA:=#$7F;
```

2) FLEX

Onder FLEX kunnen onder andere de volgende file en disk manipulaties worden verricht:

```
- formateren: NEWDISK 1                    (drive 1)
- editten   : STYLO   test                 (.txt is default)
- copieren  : COPY    test.txt nieuw.txt
- deleten   : PDEL    TEST.TXT             (hoofdletters !)
- printen   : PRINT   test.txt
- bekijken  : LIST    test.txt
```

Gebruik de [ESC] toets voor het onderbreken van een listing. Door nog een keer de [ESC] toets in te drukken gaat de listing verder, [RETURN] breekt de listing af en springt terug naar het operating system.

3) Pascal compilatie

Bij het compileren van een programma worden de volgende stappen doorlopen:

```
- PC <test > >> O L
                  |  |=> test.OL  (Compiler Listing)
                  |===> test.CO  (Compiler Object )
```

Door het meegeven van de 'L' optie aan de Pascal Compiler wordt een uitgebreide foutanalyse uitgevoerd.

```
MODULE external (output);
  {$I1.DECLARE.TXT}
    FUNCTION Square (I:integer): integer; ENTRY;
    BEGIN
      Square:=I*I;
    END; {Square}
MODEND.
```

De declaratie file ("DECLARE.TXT") zou er als volgt uit zien:

```
VAR kwadraat: integer;
```

Voor het verkrijgen van een executeerbaar programma uit dit voorbeeld moeten de volgende stappen worden doorlopen:

- PC <external > >> O L (compileer de module)
- RA <external.co > external.ro O L (assembleer de module)
- zorg dat bij de .CF file de externe module wordt meegeladen: LOAD=EXTERNAL

De module is van nu af aan direct bruikbaar en de bovenstaande stappen behoeven bij veranderingen aan andere programma eenheden niet te worden doorlopen. Het proces gaat verder gewoon als beschreven bij punt 3. (Zie voor verdere voorbeelden de bijlagen 1 tm 7).

5) Assembly interfacing

Een assembly routine die door pascal wordt aangeroepen moet in de routine als "XDEF" (vgl. "ENTRY") worden gedeclareerd. Variabelen en procedures uit een pascal programma kunnen door "XREF" (vgl. "EXTERNAL") worden binnen gehaald. Als voorbeeld kan de interrupt routine van de hardware timer worden gebruikt (zie bijlagen 8 en 9). Hierbij verdienen de instructies:

```
LDY #$BEEF
LDA #-1
```

nog een toelichting. De eerste instructie is nodig om het begin van de globale variabelen aan te wijzen. Om aan $BEEF te komen is gekeken waar de stackpointer (S) is gedefiniëerd ($CC2B) en is van de inhoud van dit adres ($BFFF) $110 afgetrokken (#$110 is het aantal systeem variabelen bij FLEX).
De tweede instructie is noodzakelijk omdat het hier gaat om het aanroepen van een niet geneste (!) procedure.

Het verkrijgen van een executeerbaar programma gaat zoals bij het modulair programmeren (zie 4). Verschillend is dat het programma niet gecompileerd moet worden (het is immers geen pascal), maar alleen geassembleerd:

```
RA <TSUP.TXT >TSUP.RO O
```

APPENDIX III

De ACIA 6551

Wij willen met dit schrijven de beginnende gebruiker op weg helpen met de ACIA.
Enkele waardevolle tips die veel tijd kunnen besparen.
Een software reset wordt gegenereerd door in het Status Register te schrijven. Het is niet van belang wat er geschreven wordt.
Het wordt aanbevolen de ACIA te resetten voordat hij wordt geinitialiseerd.
De initialisatie begint dan met het Controle Register daar deze vaak vast wordt ingesteld.
Het Commando Register moet nu naar wens worden ingesteld. De instelling van het Commando Register kan overigens tijdens het proces wijzigen.
Over het algemeen wordt ingesteld op 1 stopbit, 8 databits en pariteit af. Hoe dit alles gerealiseerd kan worden vindt je in de tabellen van de datasheet.

De programmeble timer biedt ook de mogelijkheid om interrupts af te geven. welke van de 3 timers de interrupt af geeft is aangegeven in het Status Register. Deze kan gelezen en getest worden. Wordt dit gevolgd door het lezen van een timer inhoud dan wordt de interrupt automatisch gereset.
Als bij de ontwikkeling van een systeem waarin deze timer is verwerkt een FLEX systeem als microprocessor simulator wordt gebruikt, wees dan op je hoede. Het FLEX systeem geeft namelijk zelf ook enige interrupts af. Je bent in een dergelijke situatie verplicht een interrupthandler te bouwen. Als voorbeeld zou het programma van ons kunnen worden gebruikt. Wij hebben een klok in pascal gebouwd en een hulpprogramma voor behandeling van de interrupts in assembler. Zie daarvoor ons taakverslag.

Eisenlijst 02-28

5.4	Wachttijd op kiestoon is 10 tot 15 sec.	
8.2.2.1	Kiestoon detectie duurt 1 tot 2 sec.	
8.9	Pulskiezen: pulsbreedte	60 msec.
	pauze	40 msec.
	interdigit pauze	800 msec.

Eisenlijst 02-68

4.5	Toonkiezen: toon	70 msec.
	pauze	70 msec.
5.4.2	Ontvanger van deze tonen na 20 msec detecteren (dit is in de hardware vastgelegd)	

Afhankelijk van de aard van het signaal wordt voor de detectie en generatie een bepaald systeemdeel ingezet.

```
    #$04: writeln('occupied tone detected');
    #$05: writeln('congestion tone detected');
    #$06: writeln('ERROR: line detection malfunction');
    #$07: writeln('wait for Data Carrier Detect');
    #$08: writeln('transmit MARK');
    #$09: writeln('data transmission');
    #$0A: writeln('wait for answer tone');
    #$0B: writeln('Hayes command');
    #$0C: writeln('TIMEOUT: no answer tone detected');
    #$10: ;
    #$20: writeln('detection on line 1');
    #$21: writeln('line 1 in use');
    #$22: writeln('noise on line 1');
    #$23: writeln('ringing on line 1');
    #$24: writeln('detection on line 2');
    #$25: writeln('line 2 in use');
    #$26: writeln('noise on line 2');
    #$27: writeln('ringing on line 2');
    #$28: writeln('terminating connection');
    #$30: writeln('ACCESS: polling local & remote');
    #$31: writeln('LOCAL user');
    #$32: writeln('REMOTE user');
    #$33: writeln('ENTERING LOGIN');
    #$40: writeln('connecting receiver');
    #$41: writeln('connecting transmitter');
    #$42: writeln('waiting for answer tone');
    END;
END; {Maintenance}


PROCEDURE wait (time:NATURAL); ENTRY;       {wait <time> milliseconds}
  VAR II:0..TimeScaleFactor;
      JJ:NATURAL;
BEGIN
  FOR JJ:=0 TO time DO
  FOR II:=0 TO TimeScaleFactor DO {wait};
END; {wait}


PROCEDURE CountDown; ENTRY;
BEGIN
  IF fsec>0 THEN fsec:=fsec-1 ELSE
  BEGIN
    fsec:=TimeoutFactor;
    IF msec>0 THEN msec:=msec-1 ELSE
    BEGIN
      msec:=1000;
      IF sec>0 THEN sec:=sec-1;
    END;
  END;
  IF (fsec=0) AND (msec=0) AND (sec=0)
    THEN TOF:=true ELSE TOF:=false;
END; {CountDown}
```

```
BEGIN
{************************  INITIALISATION  ********************************

{service}
  fsec:=0;  msec:=0;  sec:=0;

{user interface:  9600 baud, 8 bits, 1 stop, N}
  ACIA1st:=flag;  {reset}
  ACIA1ct:=#$1E;
  ACIA1cm:=#$0B;

{session}
  Session(initial);

{PIA's}
  PIA1AControl:=#$00;
  PIA1A:=#$FF;
  PIA1AControl:=#$04;
  PIA1BControl:=#$00;
  PIA1B:=#$F0;
  PIA1BControl:=#$36; {CB2=led output, CB1= ' off}

  PIA2AControl:=#$00; {FSK Modem control}
  PIA2A:=#$FF;
  PIA2AControl:=#$04;
  PIA2A:=#$24;        {CCITT V21 orig}
  PIA2BControl:=#$00;
  PIA2B:=#$FF;
  PIA2BControl:=#$36; {CB2=output, CB1= ' off}

  LCpiaAControl:=#$00;
  LCpiaA:=#$FF;
  LCpiaAControl:=#$16;
  LCpiaA:=#$FF;
  LCpiaBControl:=#$00;
  LCpiaB:=#$FF;
  LCpiaBControl:=#$04;
  LCpiaB:=#$FF;

{ACIA's}
{Modem Back channel}
  ACIA3st:=flag; {reset}
  ACIA3ct:=#$15; {150 baud: 1 stop bit; no parity}
  ACIA3cm:=#$02; {transmitter off}

{Modem Main channel}
  ACIA4st:=flag; {reset}
  ACIA4ct:=#$16; {300 baud: 1 stop bit; no parity}
  ACIA4cm:=#$02; {transmitter off}

  simulation:=true;
  echo       :=true;
  holdup     :=true;
  autoanswer:=true;
```

```
{SERVICES}
  IF NOT (status=timeout) THEN
  BEGIN
    TellUser('',2);
    TellUser('AVS modem  V1.0',1);
    TellUser('MAY ALL YOUR BITS BE SAVED...',2);


{DATA mode}
    IF source=remote THEN DataMode(ACIA4,ACIA4st,ACIA4,ACIA4st);


{COMMAND mode}
    REPEAT
      TellUser('',1);
      TellUser('MAIN MENU',1);
      TellUser('(0)   Exit',1);
      TellUser('(1)   Automatic Calling Unit',1);
      TellUser('(2)   Modem Procedures',1);
      TellUser('(3)   Maintenance',1);
      Take(choice,3);
      CASE choice OF
      0: ;
      1: BEGIN
           REPEAT
             TellUser('',1);
             TellUser('Automatic Calling Unit',1);
             TellUser('(0)   Exit',1);
             TellUser('(1)   Dial',1);
             TellUser('(2)   Pickup phone',1);
             TellUser('(3)   Hangup phone',1);
             TellUser('(4)   Autoredial',1);
             Take(choice,4);
             CASE choice OF
             0: ;
             1: BEGIN {Dial}
                  TellUser('dial number: ',0);
                  AskUser;
                  IF status=ok THEN
                  BEGIN
                    data:=concat('AT',data);
                    HayesCmd;
                  END;
                END;
             2: BEGIN {Pickup phone}
{...line 1/2}
                  Pickup(1);
                END;
             3: BEGIN {Hangup phone}
```

```
                FSKModem(configure);
              end ELSE
              IF symbol IN ['d','D'] THEN
              begin
                transmission:=DPSK;
                DPSKModem(configure)
              end;
              TellUser('Keep changes? (Y/N): ',0);
              AskUser;
              IF symbol IN ['y','Y'] THEN
              begin
                user[userno].transmission:=transmission;
                user[userno].protocol1:=protocol1;
                user[userno].baud:=baud;
                user[userno].special:=special;
                user[userno].encoded:=encoded;
                CASE transmission OF
                FSK : PIA2A:=encoded;
                DPSK: ;
                END;
                CASE baud OF
                 300: ACIA4ct:=#$16;
                 600: ACIA4ct:=#$17;
                1200: ACIA4ct:=#$18;
                END;
              end ELSE TellUser('changes not saved',1);
            end;
          END;
        4: ;
        END;
      UNTIL choice=0;
      choice:=2;
    END;
3: BEGIN
    REPEAT
      TellUser('',1);
      TellUser('MAINTENANCE MENU',1);
      TellUser('(0)   Exit',1);
      TellUser('(1)*  Add    a user',1);
      TellUser('(2)*  Change a user',1);
      TellUser('(3)*  Remove a user',1);
      TellUser('(4)*  Simulation mode',1);
      TellUser('(5)   Change password',1);
      Take(choice,5);
      CASE choice OF
      0: ;
      1: Session(add);
      2: Session(change);
      3: Session(remove);
      4: IF 1 IN user[userno].priority THEN
         begin
           IF simulation THEN TellUser('simulation mode off ? ',0)
                         ELSE TellUser('simulation mode on ? ',0);
           AskUser;  TellUser('',1);
```

BIJLAGE .1b

```
{************************  GLOBAL DECLARATIONS ********************************
{
        INCLUDE file

                                        Written by : P.H.G.F. Van de Venne
                                             date : May 1987


}

  CONST
  {main}
    tired              = false;       {repeat program forever}
    esc                = #27;         {escape character}
    rtn                = #13;         {return,enter character}
    LF                 = #10;         {line feed character}

  {service}
    TimeScaleFactor =  28;            {no of incrementations/millisecond  }
    TimeoutFactor   =   2;            {no of      countdowns/millisecond  }

  {user interface}
    UserTimeout     =  30;            {after no response for 30s -> timeout}

  {session}
    MaxUsers           =   2;         {maximum no. of users allowed}
    UserIdSize         =  15;         {user identification size}
    UserPwSize         =  10;         {user password       size}
    UserTpSize         =  15;         {user telephone      size}

  {elementary telephone}
    SetupTime          =   500;       {standard setup time}
    ToneIdTime         =  1100;       {time necessary to identify the signal}
    IdMaxTime          = 10000;       {tone id timeout time}


 TYPE
  {main}
    NATURAL     = 0..30000;
    MSGTYPE     = string[50];              {message string   }
    DATATYPE    = string[20];              {user input string}
    ONOFFTYPE   = (on,off);
    SOURCETYPE = (local,remote,undefined);

  {service}
    DEVICE     = (FSK,DPSK);

  {user interface}
    STATUSTYPE= (busy,ok,escape,timeout,fault);
```

```
{session}
  users       : 0..MaxUsers;                          {number of users      }
  userno      : 0..MaxUsers;                          {user number          }
  user        : ARRAY [0..MaxUsers] OF USERTYPE;      {contains all user info}

{elementary telephone}
  PIA1A         : byte at $FBE4;
  PIA1AControl  : byte at $FBE5;
  PIA1B         : byte at $FBE6; {LEDs , DTMF receiver}
  PIA1BControl  : byte at $FBE7; {control register     }
  {Line Control PIA}
  LCpiaA        : byte at $FBFC;
  LCpiaAControl: byte at $FBFD;
  LCpiaB        : byte at $FBFE;
  LCpiaBControl: byte at $FBFF;
  counter       : NATURAL;          {no of interrupts}
  dial          : TONEORPULSE;
  Number        : datatype;
  LastDial      : datatype;

{modem}
  transmission : DEVICE;
  protocol1    : char;
  baud         : NATURAL;
  special      : byte;
  encoded      : byte;
  autoanswer   : boolean;
  DataEscape   : char;             {3x this character -> DATA to COMMAND}

  PIA2A         : byte at $FBE8;   {modem PIA}
  PIA2AControl  : byte at $FBE9;
  PIA2B         : byte at $FBEA;
  PIA2BControl  : byte at $FBEB;

  ACIA3         : byte at $FBF4;   {back channel modem}
  ACIA3st       : byte at $FBF5;
  ACIA3cm       : byte at $FBF6;
  ACIA3ct       : byte at $FBF7;

  ACIA4         : byte at $FBF8;   {main channel modem}
  ACIA4st       : byte at $FBF9;
  ACIA4cm       : byte at $FBFA;
  ACIA4ct       : byte at $FBFB;
```

```
        flag:=stat AND #$10;
        CountDown;
      UNTIL (flag>#$00) OR TOF;
      destination:=LF;
    END;
  END; {Tell}
BEGIN
  CASE source OF
  local : Tell(ACIA1,ACIA1st);
  remote: Tell(ACIA4,ACIA4st);
  END;
END; {TellUser}


PROCEDURE AskUser: ENTRY;
  VAR ready : BOOLEAN;
      flag  : byte;
      temp  : byte;

  PROCEDURE Ask (VAR origin:byte; VAR stat:byte);
  BEGIN
    flag:=stat AND #$08;
    ready:=false;  data:='';  control:=false;
    REPEAT
      sec:=UserTimeout;
      REPEAT
        flag:=stat AND #$08;
        CountDown;
      UNTIL (flag>#$00) OR TOF;
      IF TOF THEN
      BEGIN
        Maintenance(#$01);
        TellUser('! TIMEOUT: no data from user',0);
        symbol:=esc;
        status:=timeout;
        ready:=true;
      END ELSE
      BEGIN
        temp:=origin;
        temp:=temp AND #$7F;
        IF temp=rtn THEN
        BEGIN
          TellUser(temp,0);
          ready:=TRUE;
        END ELSE
        BEGIN
          symbol:=temp;
          IF not (symbol IN [' '..'>','@'..'^']) THEN
          BEGIN
            control:=true;
            IF symbol=esc THEN status:=escape;
            IF (symbol=#$08) AND echo
              THEN delete(data,length(data),1)
```

```
PROCEDURE led (nmb:NATURAL; state:ONOFFTYPE); ENTRY;
  VAR data       : byte;
      UpperGroup: BOOLEAN;
BEGIN
  IF state=on THEN data:=#$80 ELSE data:=#$00;
  IF nmb>7 THEN BEGIN nmb:=nmb-8; UpperGroup:=TRUE;  END;
  IF nmb>4 THEN BEGIN nmb:=nmb-4; data:=data OR #$40 END;
  IF nmb>1 THEN BEGIN nmb:=nmb-2; data:=data OR #$20 END;
  IF nmb=1 THEN data:=data OR #$10;
  PIA1B:=data;
  IF UpperGroup THEN
  BEGIN
    PIA1BControl:=#$36;     {CB2=0, select LEDs}
    PIA1BControl:=#$3E;     {CB2=1}
  END ELSE
  BEGIN
    PIA1AControl:=#$36;     {CB2=0, select LEDs}
    PIA1AControl:=#$3E;     {CB2=1}
  END;
END; {led}


PROCEDURE DataMode (VAR transmitter,Tstatus,receiver,Rstatus:byte); ENTRY;
  VAR  cnt: NATURAL;
BEGIN
  cnt:=0;
  REPEAT
  {transmit}
    flag:=ACIA1st AND #$08;  {local user has sent char}
    IF flag>#$00 THEN
    BEGIN
      symbol:=ACIA1;
      symbol:=symbol AND #$7F;
      IF symbol=DataEscape THEN cnt:=cnt+1 ELSE cnt:=0;
      sec:=1;
      REPEAT
        flag:=Tstatus AND #$10;
        CountDown;
      UNTIL (flag>#$00) OR TOF;
      IF flag>#$00 THEN transmitter:=symbol;
    END;
  {receive}
    flag:=Rstatus AND #$08;  {remote user has sent char}
    IF flag>#$00 THEN
    BEGIN
      symbol:=receiver;
      symbol:=symbol AND #$7F;
      IF symbol=DataEscape THEN cnt:=cnt+1 ELSE cnt:=0;
      sec:=1;
      REPEAT
        flag:=ACIA1st AND #$10;
        CountDown;
      UNTIL (flag>#$00) OR TOF;
```

BIJLAGE 3

```
MODULE SESLOG (output);
{*****************************************************************************
{

                                    Written by : P.H.G.F. Van de Venne
                                         date : June 1987


}
{******************  GLOBAL DECLARATIONS  ***********************************

  {$I1.DECLARE.TXT}


{************************ EXTERNAL PROCEDURES  ******************************

  PROCEDURE wait (time:NATURAL);                                  EXTERNAL;
  PROCEDURE TellUser (msg:MSGTYPE; OpenLines:NATURAL);            EXTERNAL;
  PROCEDURE AskUser;                                              EXTERNAL;
  PROCEDURE Take (VAR choice:NATURAL; max:NATURAL);               EXTERNAL;


{************************  SESSION LAYER  *********************************
{Session}

  PROCEDURE Session (service:SESSIONSERVICE); ENTRY;
    VAR   tmp     : USERTYPE;                              {temporary user info
}
          choice : NATURAL;

    PROCEDURE Edit;
      VAR ready: boolean;
    BEGIN
      IF NOT (service=password) THEN
      BEGIN
        ready:=false;
        REPEAT
          TellUser('',1);
          TellUser('userid  : ',0);
          AskUser;
          IF NOT (data='') OR (status=timeout) THEN
          BEGIN
            ready:=true;
            tmp.id:=data;
            TellUser('password: ',0);
          END;
        UNTIL ready;
      END ELSE TellUser('Old password: ',0);
```

```
PROCEDURE SelectUser(VAR status:STATUSTYPE; VAR nmb:MAXTYPE);
  VAR found: BOOLEAN;
      II   : MAXTYPE:
BEGIN
  status:=busy;
  IF users=0 THEN TellUser('!  0 users',1) ELSE
  REPEAT
    data:='';
    TellUser('select a user  (userid/?/esc): ',0);
    AskUser;
    IF symbol='?' THEN
    BEGIN
      TellUser('presently known users:',1);
      FOR II:=1 TO users DO TellUser(user[II].id,1);
    END ELSE
    IF symbol=esc THEN status:=escape ELSE
    BEGIN
      tmp.id:=data;
      found:=LocateUser(nmb);
      IF found AND (nmb>0) THEN status:=busy {root can never be changed
                           ELSE TellUser('NOT found',1);
    END;
  UNTIL (status=ok) OR (status=escape);
END; {SelectUser}


PROCEDURE LoginUser;
  CONST MaxLogin=3;
  VAR attempt : NATURAL;
      valid   : BOOLEAN;
BEGIN
  TellUser('',1);
  TellUser('MODEM LOGIN',0);
  attempt:=1;
  REPEAT
    Edit;
    IF NOT (status=timeout) THEN
    BEGIN
      valid:=LocateUser(userno);
      attempt:=attempt+1;
      IF (attempt>MaxLogin) AND NOT valid THEN
      BEGIN
        TellUser('',1);
        TellUser('TIME PENALTY !',0);
        wait(5000);  attempt:=0;
      END;
    END ELSE TellUser('',2);
  UNTIL valid OR (status=timeout);
END; {LoginUser}
```

```
            WITH user[0] DO
            BEGIN
              id:='root';
              password:='root';
              priority:=[0..7];
              transmission:=FSK;
              protocol1:='C';
              baud:=300;
              duplex:='F';
              special:=#$00;
              encoded:=#$24;
            END;
            DataEscape:=esc;
          END;
        END;
      login:
        LoginUser;
      add:
        WITH user[userno] DO
        BEGIN
          IF 1 IN priority THEN AddUser ELSE TellUser('NOT ALLOWED',1);
        END;
      change:
        WITH user[userno] DO
        BEGIN
          IF 1 IN priority THEN ChangeUser ELSE TellUser('NOT ALLOWED',1);
        END;
      remove:
        WITH user[userno] DO
        BEGIN
          IF 1 IN priority THEN DeleteUser ELSE TellUser('NOT ALLOWED',1);
        END;
      password:
        Edit;
    END;
    END; {Session}

MODEND.
```

```
    UNTIL II>length(data);
  {convert to upper case}
    data:=upshift(data);
    LastCmd:=data;
    II:=1;
    IF NOT (data[II] IN ['A'..'H']) THEN data[II]:=#$00;
    CASE data[II] OF
    'A': BEGIN {pickup phone line 1}
           Pickup(1);
         END;
    'B': BEGIN {protocol1}
           II:=II+1;
           IF data[II]='0' THEN user[userno].protocol1:='C' ELSE
           IF data[II]='1' THEN user[userno].protocol1:='B' ELSE
           TellUser('! ERROR: non existing protocol1',1);
         END;
    'C': BEGIN {carrier on/off}
           II:=II+1;
           IF data[II]='0' THEN LCpiaA:=LCpiaA OR  #$3F ELSE
           IF data[II]='1' THEN LCpiaA:=LCpiaA AND #$C0 ELSE
           TellUser('! ERROR: use C0/C1',1);
         END;
    'D': BEGIN {dial a number}
           II:=II+1;
           Number:='';
           WHILE data[II] IN ['0'..'9',',','T','P'] DO
           BEGIN
             Number:=concat(Number,data[II]);
             II:=II+1;
           END;
           IF length(Number)<2 THEN TellUser('! ERROR: INCORRECT NUMBER',1)
           ELSE BEGIN
             flag:=LCpiaA AND #$80;
             IF flag>#$00 THEN Pickup(1);
{...line 2}
             LineState:
             IF status=ok THEN
             BEGIN
               LastDial:=Number;
               DialNumber(Number);
               IF (II>length(data)) OR NOT (data[II]=';') THEN
               BEGIN
                 CASE user[userno].transmission OF
                 FSK : FSKModem(originate);
                 DPSK: ;
                 END;
                 Maintenance(#$28); {terminating connection}
                 {disable transmitter}
                 ACIA4cm:=#$02;
                 IF TOF THEN
                 BEGIN
                   Maintenance(#$0C);
                   TellUser('! TIMEOUT: no answer tone',1);
                 END;
```

BIJLAGE 5

```
MODULE ACU (output);
{*****************************************************************************
{Automatic Calling Unit

                                        Written by : P.H.G.F. Van de Venne
                                                     M.O. Gomperts
                                             date : June 1987


}
{************************  GLOBAL DECLARATIONS  ******************************

{$I1.DECLARE.TXT}


{************************  EXTERNAL PROCEDURES  ******************************

  PROCEDURE wait (time:NATURAL);                    EXTERNAL;
  PROCEDURE CountDown:                              EXTERNAL;
  PROCEDURE Maintenance (request:byte);             EXTERNAL;
  PROCEDURE led (nmb:NATURAL; state:ONOFFTYPE);     EXTERNAL;



{**********************  AUTOMATIC CALLING UNIT  *****************************
{Pickup}
{LineState}
{Condition (line)}
{CheckNumber}
{DialPNumber (Pulse)}
{DialTNumber (Tone)}

  PROCEDURE Pickup(L:NATURAL); ENTRY;
  BEGIN
  {    - PR on,  KR on :  connect line
       - connect led on
       - wait for line to settle
       - reset line detection flag
       - KR off
  }
    CASE L OF
   1: BEGIN
         LCpiaA:=LCpiaA AND #$3B;
         led(15,on);
         wait(SetupTime);
         flag:=PIA1B;
         LCpiaA:=LCpiaA OR #$40;
       END;
```

```
  IF counter>5 THEN counter:=6;
  status:=fault;
  CASE counter OF  {identification}
    0: Maintenance(#$02);     {NO dial tone}
    1: BEGIN
         Maintenance(#$03);   {dial tone detected}
         status:=ok;
       END;
  2,3: BEGIN
         Maintenance(#$04);   {occupied tone detected}
         status:=busy;
       END;
  4,5: BEGIN
         Maintenance(#$05);   {congestion tone detected}
         status:=busy;
       END;
    6: Maintenance(#$06);     {ERROR: line detection malfunction}
  END;
END; {LineState}


PROCEDURE Condition (VAR line:NATURAL); ENTRY;
  VAR counter : NATURAL;
      tmp     : byte;
      flag    : byte;
BEGIN
  line:=0;
  flag:=LCpiaAControl AND #$80;
  IF flag>#$00 THEN
  BEGIN
    Maintenance(#$20); {detection on line 1}
    counter:=0;
    sec:=10;
    REPEAT
      flag:=LCpiaAControl AND #$80;
      IF flag>#$00 THEN
      BEGIN
        counter:=counter+1;
        tmp:=LCpiaA;  {reset line 1+2 flag}
      END;
      CountDown;
    UNTIL TOF;
    IF counter<5  THEN Maintenance(#$21) {line 1 in use}   ELSE
    IF counter<50 THEN Maintenance(#$22) {noise on line 1} ELSE
    BEGIN
      Maintenance(#$23); {ringing on line 1}
      line:=1;
    END;
  END;
  flag:=LCpiaAControl AND #$40;
  IF flag>#$00 THEN
  BEGIN
    Maintenance(#$24); {detection on line 2}
    counter:=0;
```

```
        {PR on}
        LCpiaA:=LCpiaA AND #$7F;
        wait(40);
      END;
      wait(500);
    END;
    {KR off}
    LCpiaA:=LCpiaA OR #$40;
  END; {DialP}

  PROCEDURE DialT(nmb:DATATYPE);
    VAR code: byte;
        II  : NATURAL;
  BEGIN
    {KR off}
    LCpiaA:=LCpiaA OR #$40;
    {connect DTMF-Gen}
    LCpiaA:=LCpiaA AND #$D7;
    {dial number}
    FOR II:=1 TO length(nmb) DO
    BEGIN
      CASE nmb[II] OF
      '0': code:=#$D7;
      '1': code:=#$EE;
      '2': code:=#$DE;
      '3': code:=#$BE;
      '4': code:=#$ED;
      '5': code:=#$DD;
      '6': code:=#$BD;
      '7': code:=#$EB;
      '8': code:=#$DB;
      '9': code:=#$BB;
      END;
      PIA1A:=code;  {transmit code}
      wait(100);
      PIA1A:=#$00;  {invalid code => tone disabled}
      wait(50);
    END;
  END; {DialT}
BEGIN
  dial:=pulse; {default}
  II:=1;
  WHILE II<=length(nmb) DO
  BEGIN
    IF nmb[II]='P' THEN dial:=pulse ELSE
    IF nmb[II]='T' THEN dial:=tone  ELSE
    IF nmb[II]=',' THEN wait(2000)  ELSE
    BEGIN
      st:='';
      WHILE (II<=length(nmb)) AND (nmb[II] IN ['0'..'9']) DO
      BEGIN
        st:=concat(st,nmb[II]);
        II:=II+1;
      END;
```

BIJLAGE .6

```
MODULE FSK (output);
{*****************************************************************************
{

                                    Written by : P.H.G.F. Van de Venne
                                                 M.O. Gomperts
                                         date : June 1987


}
{******************  GLOBAL DECLARATIONS  ***********************************

  {$I1.DECLARE.TXT}


{************************* EXTERNAL PROCEDURES  ******************************

  PROCEDURE Maintenance (request:byte);                          EXTERNAL;
  PROCEDURE CountDown;                                           EXTERNAL;

  PROCEDURE TellUser (msg:MSGTYPE; OpenLines:NATURAL);           EXTERNAL;
  PROCEDURE AskUser;                                             EXTERNAL;
  PROCEDURE DataMode (VAR transmitter,Tstatus,receiver,Rstatus:byte);
                                                                 EXTERNAL;


{***************************  FSK MODEM ***********************************

  PROCEDURE FSKModem (service:MODEMSERVICE); ENTRY;
    VAR  code : byte;
         flag : byte;
  BEGIN
    CASE service OF
    show:
      BEGIN
      {show present configuration}
        TellUser('PRESENT CONFIGURATION',2);
        TellUser('transmisson  : Frequency Shift Keyed',2);
        TellUser('protocoll    : ',0);
        IF protocoll='C' THEN TellUser('CCITT',2)
                         ELSE TellUser('Bell',2);
        TellUser('baud rate    : ',0);
        CASE baud OF
          300: TellUser('300   (full duplex)',2);
          600: TellUser('600   (half duplex)',2);
         1200: TellUser('1200  (half duplex)',2);
        END;
        TellUser('SPECIAL',1);
```

```
        END;
        encoded:=code;
      END;
    originate:
      BEGIN
        Maintenance(#$40); {connecting receiver}
        LCpiaA:=LCpiaA AND #$F8;
        Maintenance(#$42); {waiting for answer tone}
        sec:=30;
        REPEAT
          flag:=ACIA4st AND #$20;
          CountDown;
        UNTIL (flag=#$00) OR TOF;
        IF flag=#$00 THEN
        BEGIN
          Maintenance(#$41); {connecting transmitter}
          LCpiaA:=LCpiaA AND #$C7;
          Maintenance(#$08); {transmit MARK}
          ACIA4cm:=#$AA;
          Maintenance(#$09); {data transmission}
          ACIA4cm:=#$0B;
          DataMode(ACIA4,ACIA4st,ACIA4,ACIA4st);
        END;
writeln('disconnecting receiver+transmitter');
        LCpiaA:=LCpiaA OR #$3F;
      END;
    END;
  END; {FSKModem}


MODEND.
```

```
    configure:
      begin
        TellUser('',1);
        TellUser('protocol1    CCITT / Bell      : ',0);
        AskUser;
        IF symbol IN ['c','C'] THEN protocol1:='C' ELSE
        IF symbol IN ['b','B'] THEN protocol1:='B';
        TellUser('',2);
        TellUser('baud rate    300/600/1200      : ',0);
        AskUser;
        IF data='300'  THEN baud:=300 ELSE
        IF data='600'  THEN baud:=600 ELSE
        IF data='1200' THEN baud:=1200;
      end;
    originate:
      BEGIN
        TellUser('NOT INSTALLED',1);
      END;
    END;
  END; {DPSKModem}


MODEND.
```

```
PROCEDURE InitTimer;
  VAR  valid : BOOLEAN;
       II    : NATURAL;
       st    : STRING[2];
BEGIN
  writeln; writeln;
  TCR2:=#$01;    { cr1 may be written }
  TCR13:=#$01;   { all timers held in preset state }

  REPEAT
    write('time: ');
    readln(data);
  {hours}
    II:=1;  st:='';
    WHILE (II<3) AND (data[II] IN ['0'..'9']) DO
    BEGIN
      st:=concat(st,data[II]);
      II:=II+1;
    END;
    Time.H:=vali(st);
  {minutes}
    II:=II+1;  st:='';
    WHILE (II<6) AND (data[II] IN ['0'..'9']) DO
    BEGIN
      st:=concat(st,data[II]);
      II:=II+1;
    END;
    Time.M:=vali(st);
  {seconds}
    II:=II+1;  st:='';
    WHILE (II<9) AND (data[II] IN ['0'..'9']) DO
    BEGIN
      st:=concat(st,data[II]);
      II:=II+1;
    END;
    Time.S:=vali(st);
    IF (Time.H<24) AND (Time.M<60) AND (Time.S<60) THEN valid:=true ELSE
    BEGIN
      writeln('! ERROR: invalid time');
      valid:=false;
    END;
  UNTIL valid;

  TCR2:=#$00;    { cr2: outp off, int off, continuous,
                        normal count, ext. clock from t1 !!!!,
                        cr3 may be written }
  TCR13:=#$C3;   { prescaled by :8 , enable clock, int on, output on }
  TMSB3:=#$F4;   { init on 62,5 msec }
  TLSB3:=#$24;

  TCR2:=#$01;    { cr1 may be written }
  TCR13:=#$83;   { cr1: outp on, int off, continuous,
                        normal count, enable clock, timer held }
```

```
            Time.H:=Time.H+1;
            IF Time.H>23 THEN Time.H:=0;
          END;
        END;
{       write(esc,'7',esc,'[1;70H',Time.H:2,Time.M:3,Time.S:3);
        write(esc,'8');
}     END;
  END; {Timer}


BEGIN
{*************************MAIN*************************************************

  InitTimer;
  SETIRQ;

  writeln;  writeln('START in 10 sec');
  Timeout(10);
  repeat {nothing}; until TOF;
  writeln('START');

  write('waiting 10 sec  ');
  wait (1000);
  wait (1000);
  wait (1000);
  wait (1000);
  wait (1000);
  wait (1000);
  wait (1000);
  wait (1000);
  wait (1000);
  wait (1000);
  writeln;
  writeln(Time.H:2,Time.M:3,Time.S:3);
  writeln('DONE');
END. {clock}
```

BIJLAGE 10

HAYES COMMAND STRUCTURE

```
DIALING
D-      Digits 0..9
        Symbols # and *
        commands: P   Pulse dialing
                  T   Tone  dialing
                  ,   dialing Pause
                  :   return to COMMAND MODE after dialing
                  R   calling an originate only modem

A/      automatically reexecute last command


ANSWERING
A       manually answers an incoming call


MODEM OPERATIONS
AT      beginning of a HAYES command
-C  C0  transmit carrier signal off
    C1      "       "      "   on
-E  E0  command echo off
    E1  command echo on
-F  F0  half duplex
    F1  full duplex
-H  H0  modem    on-hook
    H1  modem   off-hook
    H2  special off-hook
-M  M0  modem speaker off
    M1  speaker until carrier
    M2  modem speaker on
-O      return to data mode
-Q  Q0  send modem responses
    Q1  do not send modem responses
-Sr?    read value in register r
-Sr=n   set register r to value n
-V  V0  single digit response
    V1  word response
-X  X0  basic    response set
    X1  extended    "       "
    X2  recognize no dial tone
    X3  recognize busy signal
    X4  recognize no dial tone and busy tone
-&G &G0 no guard tone
    &G1  550 Hz guard tone
    &G2 1800 Hz   "    "
```

FILE INDEX

{source files}

1	MODEM	.txt	MAIN PROGRAM
2	DECLARE	.txt	general global declarations
3	INTERF	.txt	user interfacing
4	SESLOG	.txt	session services
5	Hayes	.txt	command processor
6	ACU	.txt	elementary telephone services
7	FSK	.txt	FSK modem IC
8	DPSK	.txt	DPSK modem IC
9	CLOCK	.txt	test programme for hardware timer
10	TSUP	.txt	assembler: interrupt intermediary
11	INTRPT	.txt	assembler: interrupt vector definition

chain file: modem.CF

```
RA <MODEM.CO >MODEM.CA 0
RE <MODEM.PS >MODEM.PA 0
LL
STRP=$0100
LOAD=MODEM.PA MODEM.CA
LOAD=INTERF
LOAD=SESLOG
LOAD=ACU
LOAD=HAYES
LOAD=FSK DPSK
LIB= 0.RL
OBJA=MODEM.BIN
MAPC
EXIT
```