A stand-alone Dutch text-to-speech system

Institute for Perception Research
PO Box 513, 5600 MB Eindhoven

RJHD/rjhd 91/02
28.01.1991

Rapport no. 777

A stand-alone Dutch text-to-speech system Part 2: Construction

R.J.H. Deliege

A stand-alone Dutch text-to-speech system
Part 2: Construction.

R.J.H. Deliege

Contents

1 Introduction **2**

2 Hardware **2**
- 2.1 Introduction . . . 2
- 2.2 Parts list . . . 4
- 2.3 Parts count . . . 7
- 2.4 Order information non-standard components . . . 8

3 Software **9**
- 3.1 Introduction . . . 9
- 3.2 Program generation on the PMDS . . . 9
- 3.3 Program generation on the PC . . . 10

4 Diagnostic program **11**
- 4.1 Introduction . . . 11
- 4.2 Connections . . . 12
- 4.3 Power-up action . . . 12
- 4.4 Interactive mode . . . 12
  - 4.4.1 EPROM . . . 12
  - 4.4.2 RAM . . . 13
  - 4.4.3 Serial . . . 13
  - 4.4.4 Audio . . . 13
  - 4.4.5 Timer . . . 13
  - 4.4.6 Keyboard . . . 13
  - 4.4.7 Display . . . 14
  - 4.4.8 Power . . . 14
  - 4.4.9 Jumpers . . . 14

5 Auxillary programs **14**
- 5.1 Dhex . . . 14
- 5.2 Pcfdif . . . 16
- 5.3 68000lex and Lexcom . . . 16
- 5.4 Update . . . 16
- 5.5 Speak (PC) . . . 17
- 5.6 Speak (Vax) . . . 17
- 5.7 Rules and Fonpars . . . 17

1 Introduction

This report describes how to build the hard- and software for the stand-alone text-to-speech board. It covers those details that were not mentioned in part 1 (description). In addition this report explains all the auxillary software available for building or using the system.

2 Hardware

2.1 Introduction

The schematic diagrams have been redrawn for the generation of the PCB layout. Unfortunately the numbering of the components has been changed in this process. The information given in this report refers to the new diagrams. These diagrams are not included in this report because they require a larger paper size to be legible. The schematic diagrams and PCB layout are made using the software package "Cadstar" from Racal Redac. Further information is available at the Philips Nat. Lab. PCB design center. The design has Philips number 8222 255 63981. The board layout is given in figure 1. **Note:** R66 (R625 in report 1) was added after the design of the PCB. It is put directly in series with pin 10 of D18.

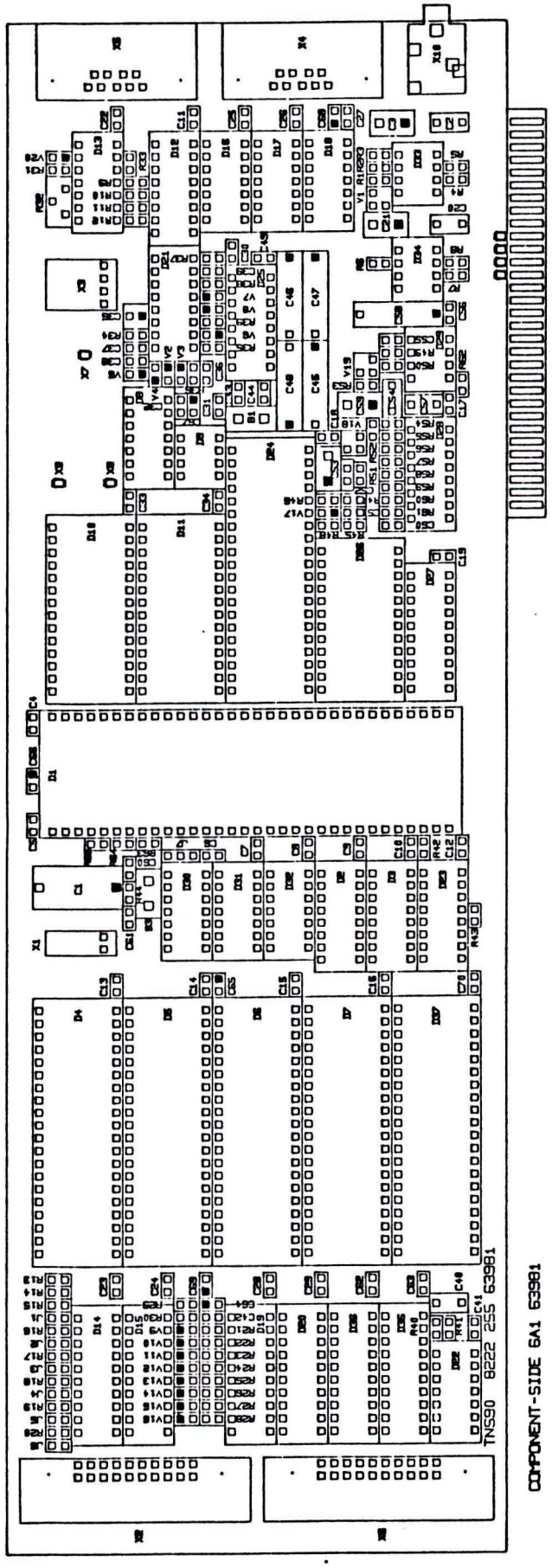

Figure 1: Board layout.

2.2 Parts list

Table 1: Crystals

B1	3.6864 MHz	B3	24 MHz

Table 2: Capacitors

C1	100 μ	C26	47 n	C51	1 n 5
C2	47 n	C27	47 n	C52	3 n 3
C3	47 μ	C28	47 n	C53	47 μ
C4	47 n	C29	47 n	C54	220 n
C5	47 n	C30	47 n	C55	10 n
C6	47 n	C31	47 n	C56	47 n
C7	47 n	C32	47 n	C57	47 n
C8	47 n	C33	47 n	C58	22 μ
C9	47 n	C34	47 n	C59	10 μ
C10	47 n	C35	4 μ 7	C60	22 p
C11	47 n	C36	100 n	C61	22 p
C12	47 n	C37	1 n	C62	47 n
C13	47 n	C38	1 n	C63	47 n
C14	47 n	C39	100 n	C64	1 μ
C15	47 n	C40	3 n 9	C65	1 μ
C16	47 n	C41	47 n	C66	1 μ
C17	47 n	C42	10 n	C67	1 μ
C18	47 n	C43	22 p	C68	1 μ
C19	47 n	C44	22 p	C69	1 μ
C20	47 n	C45	22 μ	C70	100 n
C21	47 μ	C46	22 μ		
C22	47 n	C47	22 μ		
C23	47 n	C48	22 μ		
C24	47 n	C49	47 n		
C25	47 n	C50	47 n		

Table 3: Integrated circuits

D1	68HC000	D16	74HC107	D31	74HCT393
D2	74HCT138	D17	74HC02	D32	74HCT10
D3	74HCT138	D18	74HC04	D33	ICL7663
D4	27C1024	D19	74HC245	D34	ICL7663
D5	27C1024	D20	74HC574	D35	74HC245
D6	27C1024	D21	74HCT123	D36	74HC245
D7	27C1024	D22	HEF4541	D37	27C1024
D8	74HC32	D23	74HCT147		
D9	DS1210	D24	R68C552		
D10	HM62256	D25	MAX232		
D11	HM62256	D26	PCF8200		
D12	74HCT139	D27	74HCT574		
D13	74HC32	D28	74 HCT4051		
D14	74HC245	D29	TDA7052		
D15	74HC574	D30	74HCT00		

Table 4: Resistors

R1	4 k 7	R26	10 k	R51	10 k
R2	100	R27	10 k	R52	4 k 7
R3	2 E 7	R28	10 k	R53	10 k
R4	180 k	R29	100 k	R54	150
R5	62 k	R30	100 k	R55	150
R6	18	R31	470 k	R56	220
R7	1 M 6	R32	500 k potm	R57	330
R8	560 k	R33	1 M 5	R58	470
R9	100 k	R34	100 k	R59	820
R10	100 k	R35	100 k	R60	1 k 2
R11	100 k	R36	4 k 7	R61	1 k 5
R12	100 k	R37	22 k	R62	2 E 7
R13	100 k	R38	10 k	R63	100 k
R14	100 k	R39	10 k	R64	100 k
R15	100 k	R40	100 k	R65	100 k
R16	100 k	R41	220 k	R66	470
R17	100 k	R42	100 k		
R18	100 k	R43	100 k		
R19	100 k	R44	1 M		
R20	100 k	R45	22 k		
R21	10 k	R46	56 k		
R22	10 k	R47	10 k		
R23	10 k	R48	39 k		
R24	10 k	R49	1 k		
R25	10 k	R50	18 k		

Table 5: Semiconductors

V1	BC337-40	V8	BAV10	V15	BAV10
V2	BAV10	V9	BAV10	V16	BAV10
V3	BAV10	V10	BAV10	V17	BAT85
V4	BAV10	V11	BAV10	V18	BC550C
V5	BAV10	V12	BAV10	V19	BC327-40
V6	BAT85	V13	BAV10	V20	BAT85
V7	BAV10	V14	BAV10		

Table 6: Connectors

X1	2 pin	X4	9 pin female	X10	3.5 mm stereo jack
X2	20 pin header	X5	9 pin male		
X3	4 pin	X6	20 pin header		

2.3 Parts count

Table 7: Capacitors

4	22 p	2	10 n	1	4 μ 7 (tant. 6v)
2	1 n	38	47 n	1	10 μ (tant. 16v)
1	1 n 5	3	100 n	5	22 μ (elco 16v)
1	3 n 3	1	220 n	3	47 μ (tant. 6v)
1	3 n 9	6	1 μ (tant. 6v)	1	100 μ (elco 16v)

Table 8: Integrated circuits

5	27C1024	1	74HCT123	2	ICL7663
1	68HC000	2	74HCT138	1	MAX232
1	74HC02	1	74HCT139	1	PCF8200
1	74HC04	1	74HCT147	1	R68C552
2	74HC32	1	74HCT393	1	TDA7052
1	74HC107	1	74HCT574		
4	74HC245	1	74HCT4051		
2	74HC574	1	DS1210		
1	74HCT00	1	HEF4541		
1	74HCT10	2	HM62256		

Table 9: Resistors

2	2 E 7	1	1 k 5	1	220 k
1	18	3	4 k 7	1	470 k
1	100	13	10 k	1	500 k (potm)
2	150	1	18 k	1	560 k
1	220	2	22 k	1	1 M
1	330	1	39 k	1	1 M 5
2	470	1	56 k	1	1 M 6
1	820	1	62 k		
1	1 k	22	100 k		
1	1 k 2	1	180 k		

Table 10: Semiconductors

3	BAT85	1	BC327-40	1	BC550C
14	BAV10	1	BC337-40		

The given diode types are examples of possible types, for the BAV10 any general purpose silicium diode will do and for the BAT85 any Schottky diode (low voltage drop).

Table 11: Various components

1	crystal 3.6864 MHz, fundamental mode, small case
1	crystal 24 MHz, fundamental mode, small case
1	battery 3V
1	9 pin D connector, angled, female
1	9 pin D connector, angled, male
2	20 pin header, angled
1	3.5 mm stereo jack receptable with switch
6	2 pin jumper
1	3 pin jumper
1	4 pin angled connector
1	2 pin angled connector
1	DIL IC socket 64 pins
6	DIL IC socket 40 pins
2	DIL IC socket 28 pins
1	DIL IC socket 24 pins

2.4 Order information non-standard components

27C1024 CMOS 1 Mbit (64k * 16) EPROM, 40 pin DIL
Accesstime preferrably below 180 ns, because available memory access times on the board are 180 and 360 ns. Tested are:
NEC D27C1024D-15 (150 ns), Intra Electronics BV, Nuenen
AMD AM27C1024-205DC (200 ns), Arcobel, Oss

68HC000 CMOS 68000 microprocessor, 12 MHz, 64 pin DIL
Hitachi HD68HC000P12, Arcobel, Oss

DS1210 Non volatile controller, 8 pin DIL
Dallas DS1210, ALCOM Electronics BV, Capelle aan den IJsel

HM62256 CMOS 32k static RAM, low power, 28 pin DIL
Accesstime preferrably below 180 ns, because available memory access times on the board are 180 and 360 ns. If available, the low power version should be used because of the battery backup. Tested are:
Hitachi HM62256LP12 (120 ns, low power), Arcobel, Oss
Sony CXK58256P-10L (100 ns)

ICL7663 Programmable positive voltage regulator, 8 pin DIL
Maxim ICL7663ACPA, Techmation Manudax Electronics BV, Heeswijk-Dinther

MAX232 5V powered dual RS232 transmitter/receiver, 16 pin DIL
Maxim MAX232EPE, Techmation Manudax Electronics BV, Heeswijk-Dinther

PCF8200 Speech synthesizer, 24 pin DIL
PCF8200P, Philips, Eindhoven

R68C552 CMOS Dual Asynchronous Communications Interface Adapter, 40 pin DIL
Rockwell R68C552P, Microtron, 's Graveland

TDA7052 Bridge load audio amplifier, 8 pin DIL
TDA7052, Philips, Eindhoven

Battery 3V Lithium battery
Varta ER1/2AA SLF, Varta BV, Utrecht

3 Software

3.1 Introduction

The software is written in Pascal and 68000 assembler. To compile, assemble and link this software a development environment is needed. The environment originally used is a Philips PMDS-II (Philips Microcomputer Development System) with PCP software (Philips Compiler Package). Later on this development environment became also available on other hardware platforms. Because of the advantage of using standard, general available hardware, a Personal Computer is currently used as development environment. Due to a change of supplier, the software is now called TCP (Tasking Compiler Package). The package used is CP0101-01-EF, Pascal 68000 for DOS. Because MS-DOS has no built-in MAKE facility, the MAKE from Turbo C or Pascal is used. Due to the file naming restrictions in MS-DOS, the file names on the PMDS and PC are sometimes different.

3.2 Program generation on the PMDS

To compile and/or link the program, two additional files are important. *Makefile* is a description file for the make command. With this command and the correct *Makefile* all actions to be taken to produce a loadfile can be automated. The description file *descr* is a steering file for the compiler/linker software.

All software is divided into five directories:

PCF all units for the VME rack with corresponding *descr* and *Makefile*.

STASYS those units that are different for the stand-alone board with corresponding *descr* and *Makefile*.

TIEP modified main program and additional units for Typestem program with corresponding *descr* and *Makefile*.

DIAGN stand-alone version of diagnostic program.

LIB changed libraries for EPROM version.

To generate a program, go to the correct directory (PCF, STASYS or TIEP) and give a *make ds* command. This produces a Philips type loadfile. This file is converted to extended Intel hex format for the Data I/O eprom programmer by the command *dhex ds*. This command also swaps the bytes within a word as needed. The number of sections to be programmed is the section number of section *.end_rom*. This program produces a number of output files (prom0.hex, prom1.hex...), each of which can be programmed in an EPROM.

There are two tricky points in this business:

1. Not all the files that are linked together are specified in the *Makefile*. Some files are specified in the descriptor file *descr* instead. For the file *head_em* this is done to make it the first file that is processed, the other files contain general routines that are almost always needed. The presence in the descriptor file makes it unnecessary to specify them each time.

2. The Philips supplied libraries are not used because they contain initialised constants in section *.data* that is located in RAM area. Therefore these libraries are copied to the own directory **LIB**, the sections named *.data* changed to *.roda* and then recompiled (using the script *Makelib*).

3.3 Program generation on the PC

To compile and/or link the program, two additional files are important. *Makefile* is a description file for the make command. With this command

and the correct *Makefile* all actions to be taken to produce a loadfile can be automated. The description file *descr* is a steering file for the compiler/linker software.

The main directory for the software is **TNS**. In this directory the output conversion program *dhex* is located. In the subdirectory **DS** the text-to-speech software is located and in subdirectory **LIB** the changed libraries. The Tasking supplied libraries are not used because they contain initialised constants in section *.data* that is located in RAM area. Therefore these libraries are copied to the own directory **LIB**, the sections named *.data* changed to *.roda* and then recompiled.

A loadfile can be built by the command *make <target>*, where target is one of the following:

ds text-to-speech software

typestem text-to-speech software and Typestem user-interface

ds_tst text-to-speech software without grapheme-phoneme conversion and speech-data for faster turnaround during test of program changes not involving grapheme-phoneme conversion or speech

ty_tst the same as *ds_tst* with Typestem user-interface

The name of the loadfile created is the name of the given *<target>*. This file is converted to extended Intel hex format for the Data I/O eprom programmer by the command *\tns\dhex <target>*. This command also swaps the bytes within a word as needed. The number of sections to be programmed is the section number of section *.end_rom*. This program produces a number of output files (prom0.hex, prom1.hex...), each of which can be programmed in an EPROM.

4 Diagnostic program

4.1 Introduction

For testing the hardware of the system a diagnostic program is provided. With this program the hardware can be tested and/or controlled piece by piece.

This diagnostic program is available in two versions: stand-alone and incorporated in the text-to-speech software.

4.2 Connections

The minimal connection required is the power supply. For use of the interactive mode (see further) a terminal (VT200 compatible) should be connected. All other connections have only to be made if their function has to be tested. The stand-alone diagnostic program is not affected by the setting of the jumpers 1 – 6, the baudrate for the serial ports is therefore fixed at 19200 baud. The built-in version is activated when jumper 6 is placed and uses jumper 2 and 3 for selecting the baudrates.

4.3 Power-up action

After power-up (or reset) two actions are automatically performed.

The program starts by executing RESET instructions for about 15 seconds. A RESET instruction makes the reset-pin of the processor low for some time. The purpose of this is to check the functioning of a minimal system: processor, reset circuitry, clock, memory timing and EPROM. The function can be tested by connecting a voltmeter or oscilloscope to pin 18 of the processor. The voltmeter should show a value somewhere between 0 and 5 volt, the oscilloscope should show a pulse train.

After this action a RAM test is performed. The results of this test can be displayed on the terminal in the interactive mode of the program. There is however one big pittfall in this: if RAM is not working the program will never reach the interactive mode because for example a working stack is needed. The RAM test consists of two parts: first all words are filled with the same, fixed pattern that is then checked for. After this each word is filled with its own address and checked.

After the RAM test the program displays an opening screen on the terminal. When any key is pressed the interactive mode is entered.

4.4 Interactive mode

4.4.1 EPROM

For testing all EPROM sockets four additional EPROM's are provided. The program checks only the first word of each EPROM. This test is only available in the stand-alone version.

4.4.2 RAM

The results of the power-up RAM test are displayed:
Error code 1 means error during the test with fixed pattern,
Error code 2 means error during the test with the addresses.
If an error is found, its address is also displayed.

In RAM, two words are used as a test pattern for the RAM backup. This test pattern can be set or cleared, and its presence at power-up time is displayed. After setting this pattern, switching off the system (for some while) and switching it on again should reveal the pattern to be found.

4.4.3 Serial

To test the serial ports a screenfull of characters can be sent to either the terminal or host connector. The test the input from these ports, the program waits for a character from the corresponding port. The test of the terminal port is not that usefull because it is already used in the interactive mode.

4.4.4 Audio

This menu controls the speech synthesizer, volume control and amplifier. The amplifier can be manually switched on and off, it is always switched on during speak actions. The volume can be adjusted. A test sentence can be spoken or the synthesizer can produce silence for some time. The latter function can be used in conjunction with the manual switching of the amplifier to detect any background noise.

Note: the synthesizer interrupt is not used in this test.

4.4.5 Timer

The timer can be started or stopped. When it is running a changing symbol should be visible on the terminal screen.

4.4.6 Keyboard

This menu shows the row and column number of any key that is pressed on the separate matrix keyboard.

Note: the keyboard interrupt is not used during this test.

4.4.7 Display

Not yet implemented.

4.4.8 Power

This menu displays the status of the battery control and the on/off switch. Watch out: toggling the on/off switch resets the system. The battery control display can be used to adjust the power-low detection. The system can be powered off by command. Resetting the system or pressing any key on the separate keyboard should switch it on.

Adjustment procedure for power-low detection:
Feed the board with a normal power supply (9 - 15 volt). Adjust R32 so that supply ok is displayed. Feed the board with a supply of 7.5 volt. Adjust R32 untill supply low is displayed.

4.4.9 Jumpers

The status of the jumper row (1 - 6) is displayed. Jumpers that are present are shown in inverse video.

5 Auxillary programs

The following auxillary programs are available:

Name	Language	System	Function
dhex	C	PMDS, PC	load-file conversion
pcfdif	Pascal	Vax	creation diphone tables
68000lex	DCL	Vax	up/download lexicon
lexcom	Pascal	Vax	up/download lexicon
update	Pascal	Vax	lexicon editor
speak	C	PC	speak file
speak	DCL	Vax	speak file
rules	DCL	Vax	grapheme-phoneme rules compiler
fonpars	Pascal	Vax, PC	grapheme-phoneme rules compiler

Note: C and Pascal on the PC are Turbo-C and Turbo-Pascal.

5.1 Dhex

Dhex is a conversion program that converts an a.out type loadfile into ex-

tended Intel hex format and swaps the upper and lower byte within each word. The PCP/TCP supplied conversion utilities cannot be used because they dont do the byte swap and produce only one output file. In our case, we should find some way to divide this (very big) file over more than one EPROM. The conversion takes place in two steps as illustrated in figure 2.

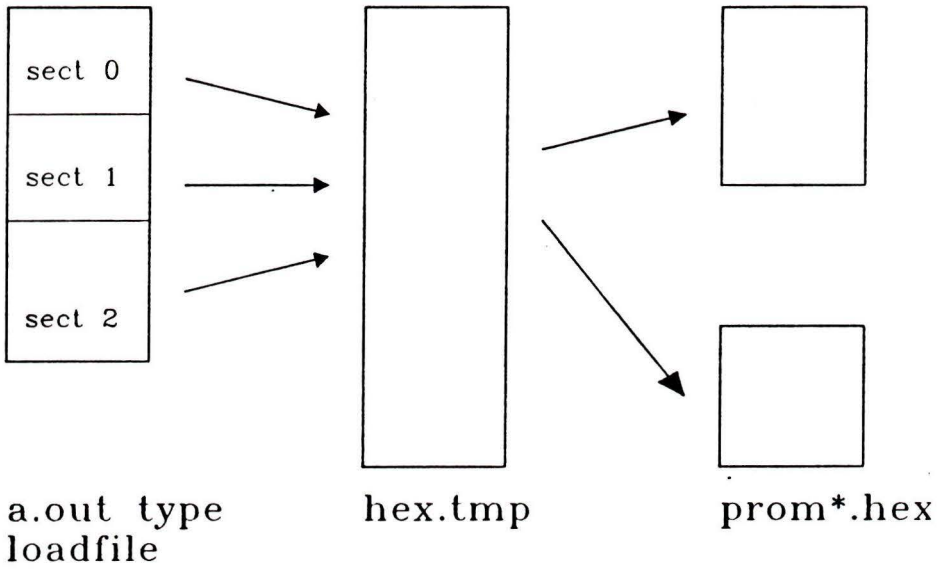

Figure 2: Operation of dhex.

The first step consists of extracting the data from the loadfile. This data is then written into a pure binary file (*hex.tmp*). This binary file contains the memory image starting at address zero. The bytes are already swapped in this file. This conversion step requires the sections in the loadfile to be in ascending order (considering their start addresses). Possible gaps between sections are filled with FF (hex).

The second step is the conversion of the binary file into one or more Intel hex files. Each file has a starting address of zero.

The intermediate file is removed at the end of the program. The program is available on the PMDS (Unix) and PC (MS-DOS). These two implementations differ in two points: under Unix, words and longwords cannot be read as a whole from the loadfile because the bytes are in the wrong order. Therefore they are read byte by byte and then the total value is calculated. Under MS-DOS (Turbo C), files can be opened in two modes: text or binary, depending on the variable *_fmode*. The loadfile and the intermediate file are

binary files, while the Intel hex files are text files.

5.2 Pcfdif

Pcfdif is a conversion program that quantizes the diphones and builds the diphone tables for the text-to-speech software. As explained in section 5.2.1 of the other report the diphone storage consists of two tables: a hashtable and a datatable. *Pcfdif* builds these two tables in assembly source format from an indexed diphone file. The hashfunction used in this program should be the same as the one in the text-to-speech software. The same holds for the hashtable length and the key length that have to be entered and the constants *priem* and *hash_len* in *ds.h*. *Pcfdif* asks several parameters, the current values are:

diphone file	:	dif:zelle_stand.ind
key length	:	7
table length	:	3001
hashfile name	:	hash.s
datafile name	:	data.s

5.3 68000lex and Lexcom

68000lex is a DCL command file for running the lexicon up/download program *lexcom*. Its function is to turn input echo off during transfers. The program *lexcom* performs the actual transfer and file handling. Lexica are stored on the Vax as indexed files. These indexed files are organised in the same way as all other lexica at IPO. Therefore these files differ in two aspects from the lexicon on the board: a third field is available for word-class and zero characters are used in stead of spaces because these files are then automatically condensed.

5.4 Update

Update is an editing program for the lexica on the Vax. It functions similar to the lexicon editor on the board with the following exceptions: at the beginning the file-name of the lexicon is asked for, there are separate update and insert commands and all changes made are logged in a separate file. Its implementation is the same as the board version except for the file access.

5.5 Speak (PC)

Speak is a program to operate the board via a PC. For interactive use a VT100 compatible terminal emulation program (e.g., *Kermit*) is better suited, because *speak* does not perform such an emulation. The main application of *speak* is to send a file to the board. *Speak* without any parameter will enter the interactive mode, while *speak* followed by a filename will speak that file.

All sentences (and commands) in the file have to be terminated by a period (so a sentence can be more than one line).

5.6 Speak (Vax)

Speak is a command file to speak a file from the Vax when connected in transparant mode. Sentences are ended by a return. To avoid the insertion of returns after 80 characters, the wrap feature of the Vax is turned off.

5.7 Rules and Fonpars

Rules is a DCL command file to call the rulecompiler *Fonpars*. *Fonpars* was developed by Joop Kerkhoff and Jip Wester of the Institute of Phonetics of the University of Nijmegen. *Fonpars* uses two inputs: a file with phonological features, always called *features.dat*, and the phonological rules from the standard input device. The command file *rules* takes as an argument the name of the file with phonological rules and assigns it to the standard input. The output is a program called *parser.pas*. Because we use a different framework in our software, the *eve* editor is called after creating *parser.pas*. The essential information can then be extracted from *parser.pas* and inserted in the corresponding file. As is explained in section 4.2 of the other report we divided the whole set of rules into four parts: front, accent, grafon and end. On the Vax these four modules are compiled separately and linked together. On the PMDS these modules are included in the source file of the main module at compile time, because the use of global variables is more difficult with this compiler. On the PC the four modules have to be compiled separately because of system limitations. The grafon part even has to be further divided into sub-parts. The file names are:

Module	Rules	Vax file	PMDS file	PC file
main	-	kun_grafon.pas	kun_grafon.p	grafon.p
front	front_rules.dat	front_mod.pas	front_mod.p	front_m.p
accent	accent_rules.dat	accent_mod.pas	accent_mod.p	accent_m.p
grafon	grafon_rules.dat	rules_mod.pas	rules_mod.p	rules_m.p
				rules_m1.p
				rules_m2.p
end	end_rules.dat	end_mod.pas	end_mod.p	end_m.p

The procedure to update a module is as follows:

- update the corresponding rule file.
- compile the rules: *@rules <rule-file>*.
- in the editor (entered automatically) get the corresponding module and replace the procedure and execution block (see figure 3) with those from *parser.pas*.

```
program parser;
      .
      .
procedure regel1;                    }
      .                              }
      .                              }      PROCEDURE BLOCK
procedure regeln;                    }
      .
begin
      .
  regel1; ..... regeln;              }      EXECUTION BLOCK
      .
end.
```

Figure 3: Structure of parser.pas.