A stand-alone Dutch text-to-speech system

Institute for Perception Research
PO Box 513, 5600 MB Eindhoven

RJHD/rjhd 91/01
17.01.1991

Rapport no. 775

A stand-alone Dutch text-to-speech system

R.J.H. Deliege

A stand-alone Dutch text-to-speech system

Part 1: Description.

R.J.H. Deliege

Contents

1 Introduction

This report describes a stand-alone text-to-speech system. This system is developed at the Institute for Perception Research (IPO), Eindhoven. The project was carried out within the national speech research programme "Analysis and Synthesis of Speech" (SPIN-ASSP) under the project-name "STASYS". The goal of this project was to come up with a microprocessor system, performing full text-to-speech conversion. For this purpose use could be made of two available components: grapheme-to-phoneme rules developed at the Institute of Phonetics of Nijmegen University together with a rule compiler that compiles the rules into a Pascal program and a speech synthesis method based on the concatenation of stored small speech fragments (diphones), developed at the Institute for Perception Research together with a speech synthesizer chip.

The possible applications of this system are manifold. It can be used everywhere where text in ASCII form is available and spoken output is wanted. The possible kinds of applications are shown in figure 1.

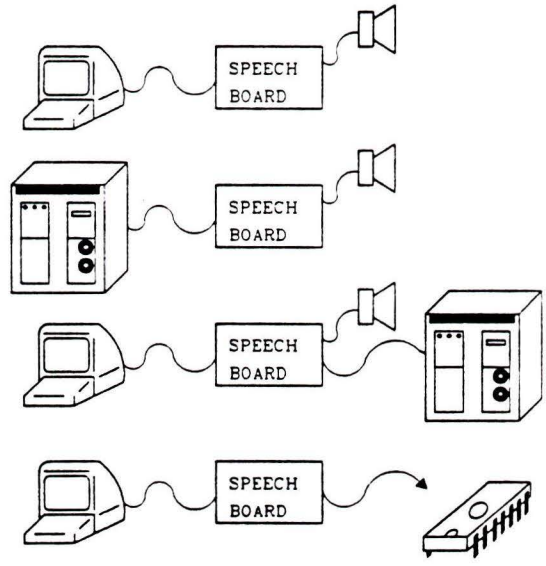


Figure 1: Possible applications.

The first possibility is human entered text through a terminal. This serves

mainly demonstration purposes. It can also be used as an educational tool, because in this setup feedback can be given on the various steps involved in text-to-speech conversion. To make the system real stand-alone, a keyboard and an LCD display can be connected instead of the terminal. A possible application for this complete stand-alone system is as a speech communication aid for the speech impaired.

The second possibility will perhaps be the most often used one: the input stems from some other device, for instance a computer, a modem or teletext. The system is then used as a speech generating back-end.

The third possibility is to place the system within an existing computer-terminal link to speak texts from and to the computer. This creates a speaking terminal.

The last possibility is to use the system as a speech programming device. Instead of analysis and coding of human speech, text can be entered and the resulting speech codes stored in the final application.

The system converts Dutch orthographic or phonetic input into speech. The conversion proceeds in a number of steps (figure 2):

- the conversion of conventional spelling into a pronunciation representation (phonetic alphabet);
- the assignment of word stress and sentence accents;
- the conversion of phonetic spelling into a sequence of prerecorded, LPC-coded speech segments (called "diphones");
- the computation of appropriate durations for the concatenated diphones;
- the computation of an appropriate intonation contour for the utterance;
- speech synthesis by means of a hardware formant synthesizer.

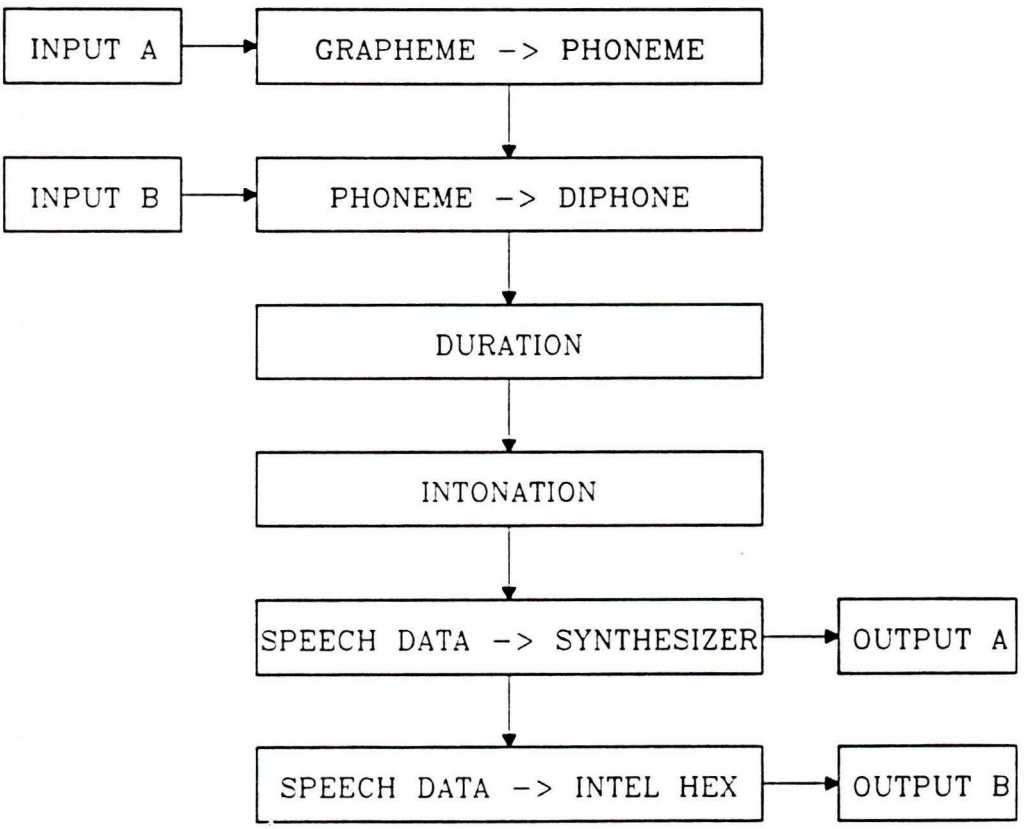


Figure 2: Text-to-speech conversion.

The input can consist of either text or commands. Commands are preceded by the special symbol "/". The input text can be interpreted in two ways, viz. as orthographic or as phonetic spelling. This interpretation can be selected by means of appropriate commands. Thanks to the phonetic-input option, it is possible to use the system in combination with other letter-to-

sound converters than the one incorporated into the system.

The orthographic input can be enriched by user-supplied accentuation markers, " ' ". In the absence of such markers, the system itself provides word stress and sentence accents. Sentence melody (intonation) is entirely computed by rule. A few duration rules are included, e.g. the lengthening of clause final syllables. The system has an exceptions lexicon that can contain the correct pronunciation of irregular forms, e.g. foreign names.

The following facilities are currently available to the user:

- a memory for the last nine sentences
- a permanent memory for nine sentences
- a screen editor
- an editor for the exceptions lexicon
- upload and download facilities for the exceptions lexicon
- a software-adjustable audio volume
- an adjustable talking speed
- the simulation of a female voice

The hardware of the system consists of a 68000 microprocessor, 512 kbyte EPROM, 64 kbyte RAM with battery backup, a Philips speech synthesizer (PCF8200), a double RS232 interface, all in CMOS, and a simple audio amplifier (TDA7052).

2 General

2.1 The board

Figure 3 gives an overview of the board with the location of the various connectors and jumpers.

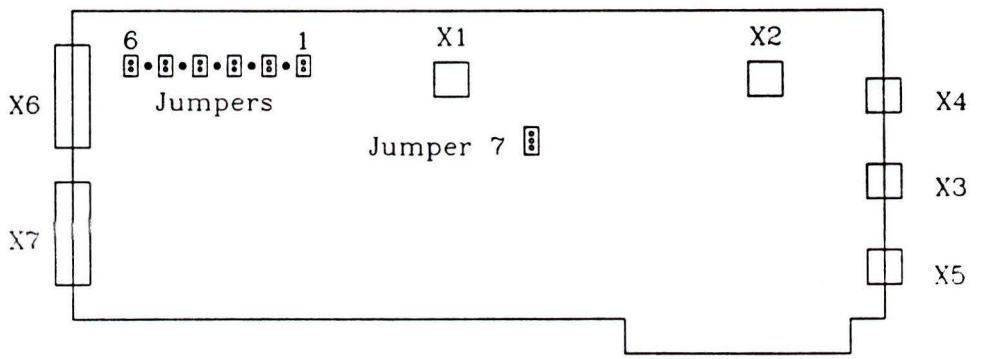


Figure 3: The board layout.

2.2 Connectors

X1 Power supply.

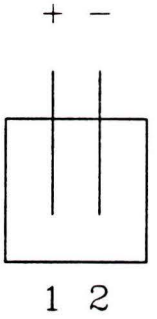


Figure 4: Pinning of power supply connector.

power supply: 7 - 15 volt unregulated.
power consumption: 80 mA (not speaking) to 250 mA (speaking).

When the board is plugged into a PC, power is taken from the PC's 12 volt supply.

X2 On/off switch.

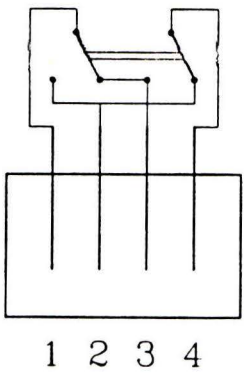


Figure 5: Connection of on/off switch.

The on/off switch does not switch the power line but resets the system when toggled. The position of the switch is checked after reset in order to determine if the system has to be active or has to be powered off. When this switch is not used a jumper has to be placed on pin 3 and 4 to assure that the active mode is selected after power-on reset.

X3 Terminal.

RS232 interface for use with a terminal or another device delivering the input texts and commands. For the screen mode of the program (see software) the terminal should be ANSI (VT100) compatible. The terminal settings should be:

baudrate 1200 or 19200 baud (jumper selectable)

data format 8 data bits, 1 stop bit, no parity

handshaking XON/XOFF

wrap autowrap on

The connector is a 9 pin female D connector. The pin assignment is given in table 1.

Table 1: Terminal connector pins.

pin	function
2	TxD
3	RxD
5	GND

X4 Host.

This second RS232 interface can be used in conjunction with a host computer. The possibilities offered, a transparent mode, up- or down-loading of the exceptions lexicon and speech data output are described in the software section. Settings should be similar to the terminal connection. The connector is a 9 pin male D connector. The pin assignment is given in table 2.

Table 2: Host connector pins.

pin	function
2	RxD
3	TxD
5	GND

X5 Loudspeaker.

An onboard amplifier is present, capable of driving an 8 Ω loudspeaker with 1.5 Watt. The output is short-circuit protected. The connector is a 3.5 mm stereo jack.

X6 Keyboard.

Connection for a matrix keyboard for input.

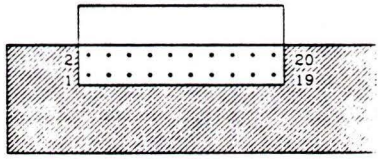


Figure 6: Pinning of keyboard connector.

Table 3: Keyboard connector pins.

pin	function	pin	function
1	INK0	11	OUTK2
2	INK1	12	OUTK3
3	INK2	13	OUTK4
4	INK3	14	OUTK5
5	INK4	15	OUTK6
6	INK5	16	OUTK7
7	INK6	17	OUTK8
8	INK7	18	OUTK9
9	OUTK0	19	NC
10	OUTK1	20	NC

X7 Display.

Connection for a Liquid Crystal Display for feedback.

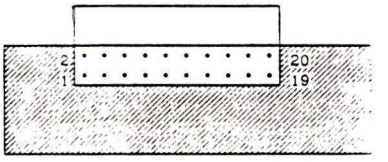


Figure 7: Pinning of display connector.

Table 4: Display connector pins.

pin	function	pin	function
1	D0	11	A3
2	D1	12	A4
3	D2	13	CLOCK
4	D3	14	SELECT
5	D4	15	$R/\overline{W}$
6	D5	16	$\overline{R}/W$
7	D6	17	NC
8	D7	18	NC
9	A1	19	GND
10	A2	20	Vcc

2.3 Jumpers

In the upper-left corner of the board there are six jumpers located to select various operation modes. These jumpers are described in table 5.

Table 5: Jumper row.

jumper	function	jumper present	jumper not present
1	program	text-to-speech	"Typestem"
2	text-to-speech mode	screen	application
3	terminal baudrate	1200	19200
4	host baudrate	1200	19200
5	time-out action	power-off	reset
6	operation	test	normal

1 In the text-to-speech mode the systems functions as a general text-to-speech system with two possible userinterfaces, in the "Typestem" mode it functions as a communication aid for the speech impaired, with an appropriate userinterface (see IPO report 724).

2 The screen mode is intended for interactive use of the system through a terminal, the application mode is better suited for the connection of the system to some kind of equipment other than a terminal.

5 The timer is set at 5 minutes. When the system is powered off, it can be switched on again by switching the on/off switch or by pressing a key on the keyboard (if connected).

6 **CAUTION:** The test mode destroys the whole RAM contents (settings, stored sentences and the exceptions lexicon). The test mode is not intended for use by persons unfamiliar with the system in detail.

Next to the processor there is one jumper (jumper 7) located to select the memory access time. When the lower two pins (towards the PC connector) are connected, the access time is 360 ns, when the upper two are connected the access time is 180 ns. This jumper can be set according to the access time of the EPROM's used.

3 Hardware

3.1 Introduction

The block diagram of the system is shown in figure 8.

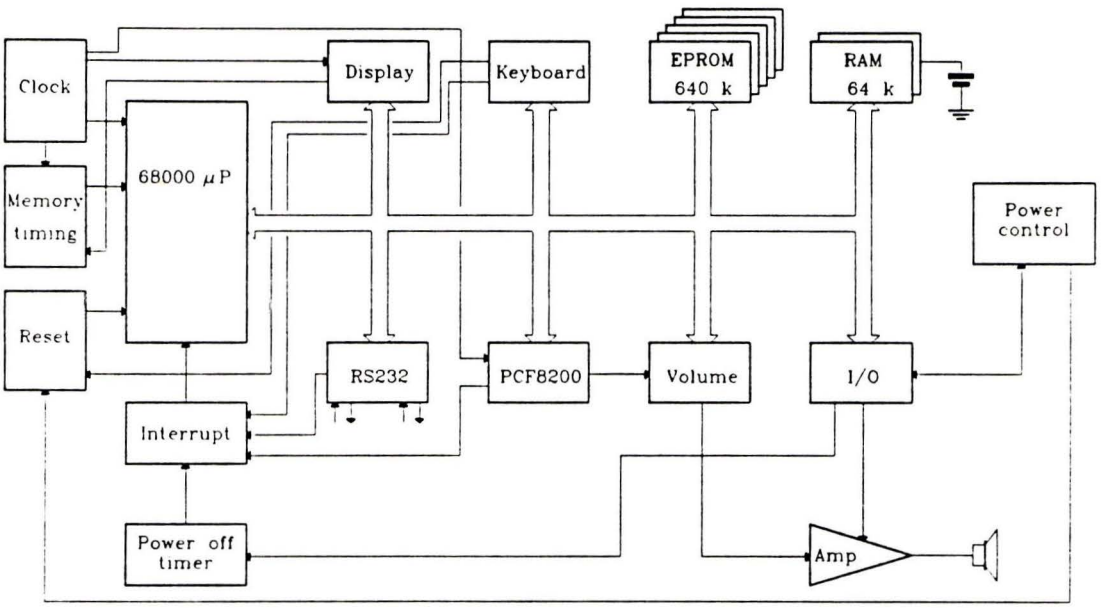


Figure 8: Block diagram.

The whole hardware is divided into nine pages in the schematic diagrams (see appendix A). This division will be followed when describing the hardware.

Page 0: processor
Page 1: clock, memory timing, LCD display
Page 2: reset timing, interrupt control, power off timer
Page 3: EPROM
Page 4: RAM
Page 5: serial interface
Page 6: power control, keyboard
Page 7: synthesizer, volume control, amplifier
Page 8: power regulation

3.2 Page 0

The processor is an 68HC000, the CMOS version of the 68000. The 68000 is chosen because of its 16 bit power and its easy and large addressing capacity. The CMOS version is chosen because of its low power consumption. The processor is running at 12 MHz, its maximum frequency.

3.3 Page 1

X101 together with 1/4 IC101 form a 24 MHz crystal oscillator. Its output is divided by 2 by 1/2 IC102, giving a processor clock of 12 MHz, by 4, giving a 6 MHz clock for the PCF8200, and by 16, giving a display clock of 1.5 MHz. The other half of IC102 is used to generate the $\overline{DTACK}$ pulse. This pulse is triggered by the address strobe ($\overline{AS}$). The clock signal for the $\overline{DTACK}$ timer can be jumper selected from 24 or 12 MHz. This results in a $\overline{DTACK}$ delay of 180 or 360 ns. This selection can be made according to the type of EPROM's used. Note: an access time of 180 ns is a little bit to fast for the PCF8200 (200 ns). In practice this will impose no problems, but to be absolutely sure or in case of problems 360 ns should be selected. The disadvantage, however, is a slower response time.

By asserting the $\overline{VPA}$ input when $\overline{AS}$ becomes active the processor is put into 6800 synchronous mode. This gives a longer access time (666 - 1332 ns).The only peripheral needing this longer time is the display (min. 350 ns for EPSON displays).

The $\overline{VPA}$ input is also used to put the processor in autovectoring mode. During a normal interrupt acknowledge the processor expects an interrupt vector on the databus. Because most of our interrupting devices (in fact all except the DACIA) have no built-in facility to generate such a vector, additional hardware would be required. As an alternative the processor can be put in autovectoring mode by asserting $\overline{VPA}$ during an interrupt acknowledge cycle. This cycle can be detected from the function code ($FC0..2$). When in autovectoring mode the processor uses an internal vector, one for each interrupt level.

The $R/\overline{W}$ signal has to be inverted for the PCF8200.

The connector for the LCD display has all signals available that could be needed because the actual display type was not chosen when the board was realized. All signal lines to and from the display are buffered in order to keep the high frequency processor lines as short as possible.

3.4 Page 2

IC201 generates the reset pulse for the processor ($\overline{RESET}$ & $\overline{HALT}$) and for the serial interface ($\overline{RES}$). The length of the reset pulse is 500 ms, by R203/C203. There are three events possible that trigger the reset pulse: power-up of the whole system (through R207/C205), switching the system on or off (through S201, C206/C207 and D204..207) or pressing any key on the keyboard when all output lines to the keyboard matrix are high (through signal PUK).

So it is obvious that the on/off switch S201 does not switch the actual power to the system. Instead, every switch action resets the whole system and the switch position is available to the processor (signal $SwPos$). So immediately after a reset the processor should test this line and according to its level power down the system or start processing. As will become clear the power can be switched on and off by the processor.

IC203 is the power off timer. It can be run or stopped by the processor through the signal $\overline{Enpot}$. The period time of the timer is 0.2 seconds, so a level change will occur after 0.1 second. As the output of the timer is used as an interrupt, this interrupt will occur 0.1 second after starting the timer (stopping the timer also resets it).

IC202 codes the interrupts to the three processor interrupt inputs. The interrupts are:
level 0: keyboard
level 1: speech synthesizer
level 2: serial channel 2 (host)
level 3: serial channel 1 (terminal)
level 4: power off timer
The priority assignment has mainly to do with the fact that when a certain interrupt level is disabled, also all interrupts below this level are disabled. The keyboard interrupt has to be disabled whenever a key is pressed on the keyboard (see the description of keyb.s), so this interrupt got the lowest priority. Because the speech synthesizer interrupt sometimes has to be disabled (during the beginning of an utterance) and the serial interrupts always are enabled, the serial interrupts got the higher priority. The power off timer got the highest priority because there will never be the need to disable this interrupt, the timer can be stopped instead.

3.5 Page 3

The connection of the five EPROM's is rather straightforward. The used EPROM's are the 27C1024, each 64k * 16bit, giving 640 kbyte memory. IC303 is the address decoder, it has room left for three additional EPROM's. The signals $\overline{UDS}$ and $\overline{LDS}$ are not used, at a byte read operation all 16 bits are presented. This imposes no problem because the processor only reads the correct byte.

3.6 Page 4

As RAM two chips (Sony CXK58256P, 32k * 8bit) are used, giving 64 kbyte of RAM space. IC401 is the address decoder, it has room for one additional RAM pair. The same address decoder decodes the addresses for the peripheral chips. The highest output ($Y7$) of the address decoder is not used because this signal is active during an interrupt acknowledge, when the processor makes $A4..A23$ high. The chip enable signal for the RAM's is gated with the $\overline{UDS}$ and $\overline{LDS}$ to allow for byte-size memory access.

The RAM's are provided with a backup battery (B101). This is a Varta ER1/2AASLF, Lithium battery, 3 V, lifetime 10 years, capacity 1Ah. The standby current of one RAM chip is typically 2 μA. So the theoretical lifetime of the battery is $10^6/4 = 250{,}000$ hours = 28.5 years. Because of the practical lifetime of 10 years, this battery has enough capacity. To switch this battery on, IC402 (Dallas DS1210) is used. When Vcc drops below 4.75 V, RAM power is switched to the battery instead to Vcc. At the same time the chip enable for the RAM's is made unconditionally high to prevent access to the RAM's. Be aware that the gates for $\overline{UDS}$ and $\overline{LDS}$ have to be connected to the RAM power instead of to Vcc, because their outputs have to be high during standby. Because the power supply of these gates during standby goes down to 3 V, IC405 must be a HC type (instead of HCT).

3.7 Page 5

IC501 is a double serial interface chip (DACIA: double asynchronous communications interface adapter), type Rockwell 68C552. The connection to the processor is rather straightforward, only the two $\overline{IRQ}$ outputs are open drain and need a pull-up resistor. The serial interface has its own clockgenerator for the receiver and transmitter clock. For this purpose X501 (3.6864 MHz) is present. From the serial connections only TxD and RxD are used (handshaking is done in software by XON/XOFF).

IC502 (Maxim MAX232) converts the levels from 0 and +5 V to -10 and +10 V and vice versa. The -10 and +10 V supplies are made internally in IC502.

3.8 Page 6

IC603, 605, 606 and 608 provide the processor with 16 input and 16 output lines because the processor has no input/output ports itself. For a matrix keyboard there are 10 output lines (IC606 and Q6..7 from IC603) and 8 input lines (IC608). Through eight diodes (D601..608) all input lines are orred to form the PUK (power up keyboard) and keyboard interrupt signals. When all ones are written to the output lines these signals (PUK and interrupt) become active when any key is pressed.

From IC603 three other outputs (Q0..2) are used for internal control:
Q0: $Vaudcon$: audio amplifier power
Q1: $Vcccon$: Vcc on
Q2: $\overline{Enpot}$: enable power off timer

From IC605 two inputs (A0..1) are used for internal control:
A0: $BatCon$: power level control
A1: $SwPos$: on/off switch position

The other inputs (A2..7) are equipped with jumpers.

The address decoder (IC401, page 4) generates one select signal for all input/output ports. 1/2 IC609 generates four select signals from this common select by using $A1$ and $R/\overline{W}$.

IC607 forms the power control circuit.

With 1/4 IC604 a power low detection circuit is made. With R611 the switching point can be adjusted.

Because some of the circuits are involved in the power control, these circuits have to be active even if the system is powered off. They are therefore supplied with $Vper$, which is present whenever there is power connected to the board.

3.9 Page 7

The PCF8200 speech synthesizer (IC701) is rather straightforward connected to the processor. The audio output is filtered by a low pass filter (R703..704/C706..707) and then led to the voltage divider R705..712. One

of the outputs of this divider is selected by the multiplexer IC702. The DC component in the PCF output signal is not blocked so the input signal for the multiplexer is always positive. The audio signal is then again low pass filtered (R717/C705) to remove some digital noise. The signal is then amplified by bridge amplifier IC704, capable of driving an 8 Ω loudspeaker with 1.5 Watt.

The audio amplifier is only powered when there is speech output. The power supply is switched by T701..702. C711..712 delay the switching off action.

3.10 Page 8

IC801 and 802 are two regulators for the 5 V power supply. IC 801 regulates $Vper$, which is present whenever power is connected to the board. IC802 regulates Vcc, the normal power supply for the board, which can be switched on and off. To allow for a higher maximum output current T801 is added. The current limits are 27 mA for $Vper$ and 215 mA for Vcc.

4 Grapheme-to-phoneme conversion

For some time research has been going on at various institutes to come up with (among other things) a grapheme-to-phoneme conversion system for Dutch. At least one of these systems has reached the point where its performance is good enough for practical use and which can easily be implemented in a microprocessor system. This system [Kerkhoff, Wester and Boves, 1984] has been developed at the Institute of Phonetics of the Nijmegen University. Before we can use this system in our application together with the diphone concatenation some changes had to be made.

This section describes this grapheme-to-phoneme conversion system as it became available to us, the changes made to it and some implementation points.

4.1 The original system

The system is rule-based, i.e. the grapheme-to-phoneme correspondences are formulated as a set of linguistic rules. This set of rules can be compiled into a Pascal program by means of a rule-compiler program that was developed for this purpose (called FONPARS). This process is illustrated in figure 9.

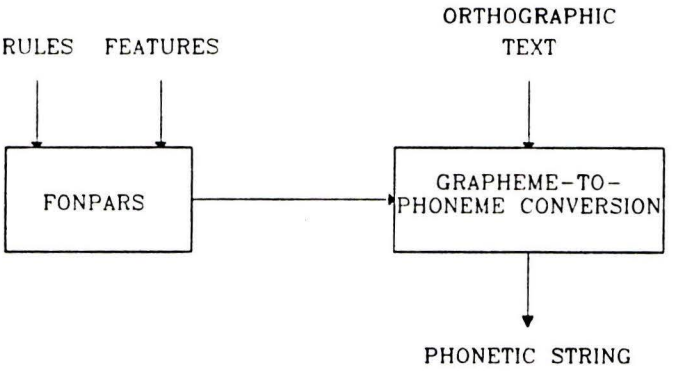


Figure 9: Use of FONPARS.

The format in which the rules are written is analogous to the notation described in the Sound Pattern of English (SPE) [Chomsky and Halle, 1968]. The general format of a rule is:

```
F -> C / L --- R
```

where: F = focus, C = change, L = left context, R = right context.

The capital letters in the rule format can represent so-called "phonological features". Phonological features are characteristics of sound segments, with which we can address groups of sound segments as a set. We can, for instance, address the vowels and consonants as sets via obvious features like [+voc] and [+cons] respectively. FONPARS needs to have access to a feature table in which the characteristics of sound segments are presented.

The complete rule format is described in Kerkhoff and Wester (1987). It allows among other things for: insertions, deletions, exchanges, feature specifications, optional elements, or-or statements and negations. Figure 10 gives an example of some rules. The first rule specifies the pronunciation of the graphemes **au** when preceded by the graphemes **rest** or **ch** and followed by **f** or **r**. The second rule specifies a phonetic variant of **n** when followed by zero or more symbols with feature **[-seg]** followed by a **r**, **g** or **x**.

```
au -> oo / {rest/ch} --- {f/r}
n -> ~ / --- [-seg]0 {r/g/x}
```

Figure 10: Example rules.

All rules are successively applied to the input string. For each rule the focus F moves from left to right through the string. Whenever the focus matches, the left and right contexts are evaluated. When these also match, the rule applies and the focus is replaced by the change C.
For grapheme-to-phoneme conversion a complete set of rules has been developed [Kerkhoff, Wester and Boves, 1984]. In this rule set we can distinguish various groups of rules. These groups are shown in Table 6.

Table 6: Grapheme-to-phoneme rules.

Function	Approx. number
roman numbers	10
arabic numbers	25
abbreviations	5
uppercase words	5
sentence accent	25
grapheme-to-phoneme	100
word accent	50
assimilation	20
syllable boundaries	10

For text-to-speech application this system is followed by a speech synthesis part. This synthesis part generates allophones by rules. These rules are written in a format close to that of the grapheme-to-phoneme rules.

4.2 The modified system

In order to use this grapheme-to-phoneme conversion together with the diphone-based speech synthesis, some modifications had to be made, for the following reasons:

1. The phonetic output of this system differs from our phonetic notation, both in coding and in the allophonic variations used.
2. The phonetic output is meant as input for an allophone synthesizer, rather than a diphone-based one. This sometimes results in other phoneme sequences.
3. The intonation part of the system already selects basic intonation patterns while we have our own selection system that expects the accent positions and punctuation marks as input.
4. The system, which is rule-based, does not support an exception lexicon. Such a lexicon is necessary, however, in our application (e.g., for names).

Ad 1). The phonetic output of this conversion (Table 7) had to be translated to the notation as used with the diphones (Table 8).

Table 7: Dutch phonemes (and some of their allophones) and their character representation for use with the allophone synthesis.

character representation	**example word**	**character representation**	**example word**
A	b*a*d	p	*p*ut
E	b*e*d	b	*b*ad
I	b*i*d	t	*t*ak
O	b*o*d	d	*d*ak
U	p*u*t	k	*k*at
a	b*aa*t	G	*g*oal
e	b*ee*t	f	*f*iets
o	b*oo*t	v	*v*at
i	b*ie*t	s	*s*ap
y	b*oe*k	z	*z*at
u	b*uu*rt	C	po*tj*e
@	b*eu*k	S	was*j*e
E:	*ai*r	Z	*j*aquet
EI	b*ij*t	x	la*ch*en
UI	b*ui*t	g	la*g*en
AU	b*ou*t	m	*m*at
&	d*e*	n	*n*at
!	a*n*jer	l	*l*at
N	la*ng*	r	*r*at
-	i*n*gaan	j	*j*at
*	a*an*was	w	*w*at
h	*h*ad		

Table 8: Dutch phonemes (and some of their allophones) and their character representation for use with the diphone synthesis.

character representation	**example word**	**character representation**	**example word**
SI	< *stilte* >	EW	l*eeuw*
GS	< *glottalstop* >	IW	k*ieuw*
II	l*ie*p	YW	d*uw*
I	p*i*t	P	*p*as
EE	l*ee*s	T	*t*as
E	l*e*s	K	*k*as
EH	mayon*ai*se	B	*b*as
AA	m*aa*t	D	*d*as
A	m*a*t	G	*g*oal
OO	r*oo*d	S	*s*ok
O	r*o*t	F	*f*ok
OH	z*o*ne	X	*g*ok
U	r*oe*t	Z	*z*eer
Y	f*uu*t	V	*v*eer
CC	p*u*t	M	*m*eer
C	d*e*	N	*n*eer
UH	fr*eu*le	NN	ma*n*dje
AU	k*ou*d	Q	ba*ng*
UI	m*ui*s	L	*l*ang
OE	k*eu*s	LL	da*l*
EI	r*ei*s	R	*r*ang
AI	det*ai*l	W	*w*ang
AJ	m*aai*t	J	*j*an
OI	h*oi*	H	*h*ang
OJ	h*ooi*t	PJ	boom*pj*e
UJ	r*oei*t	TJ	*tj*olk
ER	b*eer*	SJ	*sj*aak
OR	w*oor*d	DJ	*dj*atiehout
CR	k*eur*	ZJ	*j*ournaal
AW	k*auw*	DZ	mana*g*er

Because there is no one-to-one correspondence between these two notations a simple table conversion is not possible. In addition there are some am-

biguities, for instance E, I and EI are all valid phonemes. Therefore some kind of interface software is necessary. The rule format already used for the grapheme-to-phoneme conversion is perfectly suited to this task, so this translation is written as an additional set of rules. These rules are listed in figure 11.

```
(* Conversion of KUN phonetic symbols to IPO notation   *)
(*=====================================================*)
(* Remove double spaces *)
(* Remove spaces before a comma *)
# -> $ / --- {# / ,}
(* Remove space between accent marker and word *)
# -> $ / + ---
(* Use \ as word separator instead of space *)
# -> \
(* Add a space between all phonemes *)
$ -> # / ^#^ ---
E # I -> EI
U # I -> UI
A # U -> AU
(* Conversion KUN to IPO notation *)
C -> TJ
U -> CC / # --- #
a -> AA
e -> EE
o -> OO
i -> II
y -> U
u -> Y
@ -> OE
E # : -> EH
& -> C
N -> Q
*! -> NN
~ -> N
** -> N
h -> H
p # " # j -> PJ
```

```
p -> P
b -> B
t # " # j -> TJ
t -> T
d -> D
k -> K
f -> F
v -> V
S -> SJ
s # " # j -> SJ
s -> S
Z -> ZJ
z -> Z
x -> X
g -> X
m -> M
n -> N
l -> L
r -> R
j -> J
w -> W
" # -> $
(* Remedy against aankomst -> AAQKOMST etc. *)
Q -> N / [+voc] [+voc] # ---
```

Figure 11: Conversion rules from KUN to IPO phoneme notation.

Ad 2). Some rules have to be added after this translation because a number of phonetic variations used by the diphone synthesis are not covered by the phonetic output of the original system. These variations are the silence phoneme, glottal stop, thick L and some diphthongs. These rules are listed in figure 12.

```
(* Allophonic variants *)
AA # {I/J} -> AJ / --- [-ipo_seg]0 {[+ipo_cons] / \}
OO # {I/J} -> OJ / --- [-ipo_seg]0 {[+ipo_cons] / \}
O # {I/J} -> OI / --- [-ipo_seg]0 {[+ipo_cons] / \}
U # {I/J} -> UJ / --- [-ipo_seg]0 {[+ipo_cons] / \}
A # I -> AI / --- [-ipo_seg]0 {[+ipo_cons] / \}
```

```
AU # W -> AW / --- # [-ipo_seg]0 {[+ipo_cons] / \}
EE # W -> EW / --- # [-ipo_seg]0 {[+ipo_cons] / \}
II # W -> IW / --- # [-ipo_seg]0 {[+ipo_cons] / \}
Y # W -> YW
D # J -> DJ
J # -> $ / NN # [-ipo_seg]0 ---
# N # -> # NN # / --- [-ipo_seg]0 {SJ/TJ}
S -> Z / EH # ---
(* Thick L rules *)
# L # -> # LL # / --- [-ipo_seg]0 \ [-ipo_seg]0 ^L^
# L # -> # LL # / --- {S/T/D}
# L # -> # LL # / [+ipo_seg] --- [-ipo_seg]0 [+ipo_cons]
                                    [-ipo_seg]0 [+ipo_voc]
(* Add silence after a comma *)
$ -> # *# # *# / , ---
(* Add glottal stop between words *)
# -> # GS # / [+ipo_seg] # \ -L- [-ipo_seg]0 [+ipo_voc]
(* Add glottal stop in words (geopend, geacht) *)
# -> # GS # / C -L- [-ipo_seg]0 [+ipo_voc]
(* Allophonic variants *)
OO -> OR / --- # [-ipo_seg]0 R
EE -> ER / --- # [-ipo_seg]0 R
OE -> CR / --- # [-ipo_seg]0 R
(* Translate silence symbol *)
*# -> SI
```

Figure 12: Rules for allophonic variants.

In addition, some rules of the existing set had to be modified. These rules take care of effects that are already incorporated in the diphones (e.g., glide insertion).

Ad 3). The original system comes up with two types of accent information: the sentence accent (a pattern number before a word to be accented) and word accent (the accent position within a word). The software is changed in such a way that the sentence accents are removed and the word accents are kept in those words that had this sentence accent.

Ad 4). The existing system uses no exception lexicon. For practical applications this is a serious omission. However good the grapheme-to-phoneme rules will be, there will always be irregularities not covered by these rules in

a practical application (e.g., jargon, proper names). These irregularities can be covered by putting them together with their correct phonetic translation in a lexicon. In order to obtain maximum benefit from this lexicon, the user should preferably be able to change it. In this case he can tune the lexicon to his own application and vocabulary. Attention has to be paid to the translation of the input string, because the rules are designed to operate on the complete sentence, while the lexicon operates on words.

To incorporate these changes, the rule set (plus the additional self written rules) has been divided into four groups as given in Table 9.

Table 9: Grouped grapheme-to-phoneme rules.

Name	Rules
Front	roman numbers
	arabic numbers
	abbreviations
	uppercase words
Accent	sentence accent
Grafon	grapheme-to-phoneme
	word accent
	assimilation
End	allophonic variations

A new software framework has been developed using these four groups of rules as illustrated in figure 13. This translation strategy differs in two points from the straightforward translation by the rules. These two points are the possibility to skip the sentence (or sentence and word) accent rules and the presence of an exception lexicon.

The possibility to skip the accent rules is implemented because the accent rules often come up with accent positions that are far from ideal. This is due to the fact that proper intonation is dependent on the meaning of the sentence, which a machine cannot know without extensive semantic analysis, and on the complexity of the other factors that influence intonation, such as syntax. It is worth while, therefore, creating a possibility to let the user provide the accent positions. On the other hand, when the user does not want or is not able to provide this information, the accent positions generated by the rules are presumably better than no accents at all. The choice whether to use the accent rules has therefore to be made by the user and is implemented as follows. If the user supplies accents in the input, these

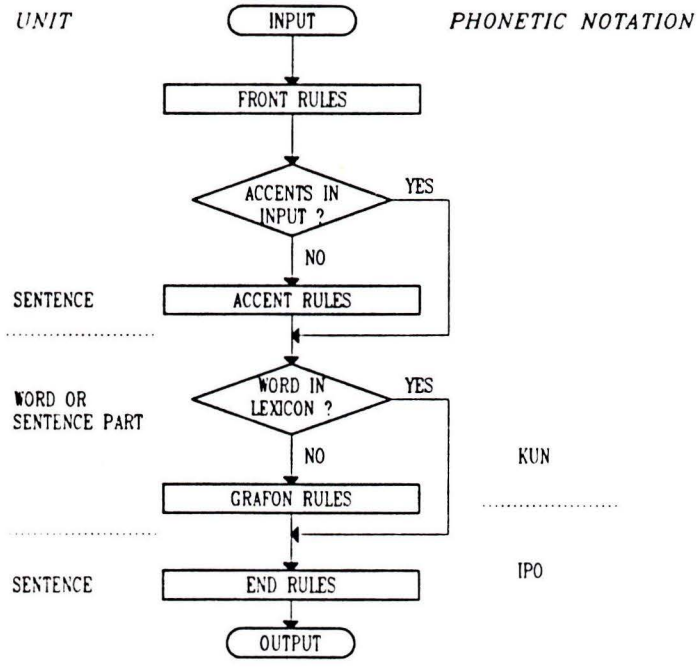


Figure 13: Grapheme-to-phoneme conversion.

accents are used, otherwise the rule-generated accents are used. Because the word accent rules (accent position within a word) function considerably better than the sentence accent rules, these word accent rules can be used most of the time. When the user places an accent symbol before a word, he indicates that this word has to be accented and the word accent rules determine the exact accent position. For cases where this goes wrong, he can place the accent symbol within the word, indicating the syllable to be stressed. This feature, however, was not foreseen during the development of the rules and therefore this placement of an additional symbol within a word, may cause the grapheme-to-phoneme rules to work incorrectly. Because this feature is only meant to be used in "emergency cases" no better solution is looked for.

As mentioned before, the exception lexicon operates on words, while the rules work on complete sentences. The rules are designed to work on complete sentences because 1) for calculating sentence accent positions the whole sentence is needed and 2) neighboring graphemes influence each other in the grapheme-to-phoneme conversion. Therefore the conversion by the

lexicon and by the group of grafon rules is carried out as follows. Each word in the input string is searched for in the lexicon, going from left to right through the string. This process continues until a word matches or the end of the string is reached. At that point the part of the string preceding the matching word (or the whole string in case the end was reached) is translated by the rules and the matching word is translated by the lexicon. The same process then starts again with the remaining part of the sentence. In this way the problems with the lexicon operating on words and the rules operating on sentences are minimized. The calculation of the accent positions (1) is still done on the whole sentence. The grapheme-to-phoneme conversion rules operate now on sentence parts (2). The only places where these rules miss the neighboring graphemes is where the sentence is interrupted by words present in the lexicon. In this way this missing information is kept to a minimum. The rules in the group of end rules often work across word boundaries, so this group of rules is processed separately from the grafon rules. In this way these rules always work on a complete sentence.

4.3 References

Chomsky N. and Halle M. (1968), *The sound pattern of English*, (Harper & Row, New York).

Kerkhoff J., Wester J. and Boves L. (1984), "A compiler for implementing the linguistic phase of a text-to-speech conversion system", *Linguistics in the Netherlands*, pp. 111-117.

Kerkhoff J. and Wester J. (1987), *Fonpars1 user manual, Part I: rule format*, internal publication Institute of Phonetics, Nijmegen University.

5 Software

5.1 General

The software is mainly written in Pascal. The use of a high level programming language has numerous advantages above assembly language. The software is easier to read and to maintain. The development of the software is much easier and can be done on another system, in this case a Vax, which has better debugging facilities and works much faster. The disadvantage, however, is that a high level language can result in less efficient code, both in size and in speed. Size has not to be a problem if the memory needed is still within acceptable limits. The same holds for the speed: as long as the system is fast enough (to allow for real-time processing), inefficiency is no problem. As it turned out during the development that the system was not fast enough, parts of the software had to be made more efficient. This was partly done by rewriting the Pascal code and partly by the use of assembly language. Assembly language is also used for those parts of the software that are hardware dependent. For the high level language Pascal was chosen for two reasons: the compiler for the grapheme-to-phoneme rules generates Pascal code and much software was already available on the IPO Vax in Pascal.

In order to maintain a clear overview over the software each functional part is put into a separate unit. This has the additional advantage of speeding up compile time because only the units that are changed have to be compiled. The Pascal code files have suffix *.p*, the assembly language files have suffix *.s*. The files present are:

accent_m.p accent rules

chip.p speech synthesis

dacia.h DACIA register declarations

data.s diphone data table

diagn.p diagnostic module

diphone.s diphone data table access

dir_io.s direct I/O to terminal and host

ds.h common declarations (constants and types) for Pascal units

ds.p main program

duration.p generation of prepause lengthening

edit.p screen editor

end_m.p end rules

even.s alignment of all PROM sections to even addresses

fd.p phoneme-diphone conversion

file_io.s dummy routines for file I/O

front_m.p pre-processing rules

gffon.p phoneme input

grafon.p grapheme-phoneme conversion

hardw.s hardware support

hash.s diphone hash table

head_em assembly body for whole program

hex.p speech data output in Intel Hex

into.p generation of pitch contour

keyb.s keyboard read

lex.s exception lexicon access and update

lex_edit.p maintenance of exception lexicon

parse.s fast assembly routines for grapheme-phoneme conversion

pcf.s PCF8200 control

rdstr.p read string

rules_m.p grapheme-to-phoneme rules

rules_m1.p grapheme-to-phoneme rules

rules_m2.p grapheme-to-phoneme rules

screen.h declarations of screen control routines

screen.p screen control

speech.s speech data for diagnostic module

strings.p string handling

term_io.s Pascal I/O

5.2 Data structures

5.2.1 Diphone tables

For the storage of all diphones a special data-structure has been chosen that allows for easy and fast access of each diphone. Just one table containing all diphones would be a waste of memory, because in this case all entries should be of equal length to allow addressing. This implies that all entries should have the length of the longest diphone. Instead a table is used where all diphones are placed immediately behind each other. To be able to locate each diphone in this table, a second table is present, giving the address in the first table for each diphone. The entry in this second table is found by a hash function on the diphone name. This is illustrated in figure 14.

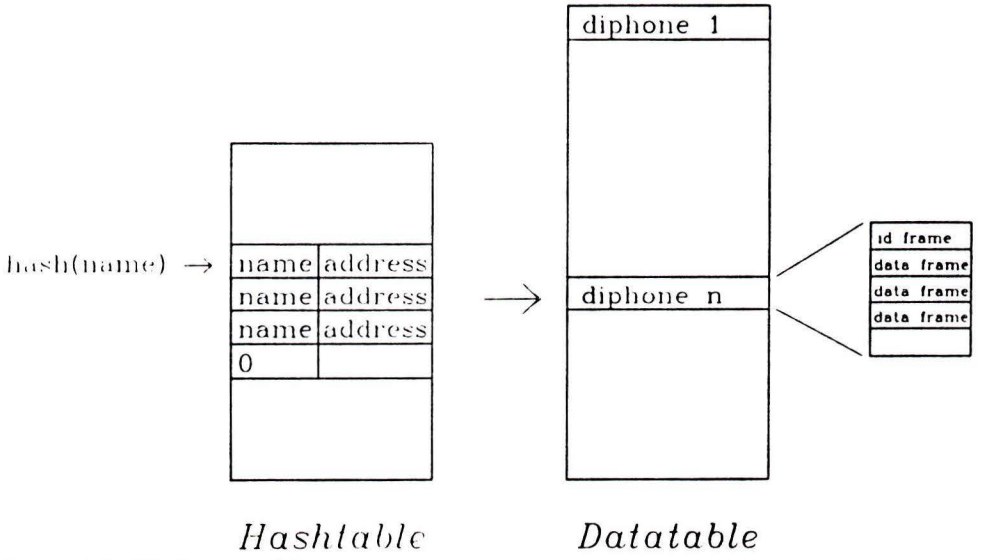


Figure 14: Diphone tables.

The construction of the tables goes as follows:

Take the hash function of the diphone name, this gives an entry in the hash table. If this entry is occupied (name $\neq$ 0), go to the next entry until a free place (name = 0) is found. Store the name of the diphone in the name field, and the address of the first free location in the data table in the address field. Place the speech data for this diphone behind the previous one.

The look-up of a diphone goes more or less in the same way. Take the hash function of the diphone name, this gives an entry in the hash table. From this entry on, compare the diphone name with the name in the name field of that entry, until they match or until the name field is zero. In the first case the address field gives the location of this diphone in the data table, in the second case this diphone is not present.

For optimal performance the hash table length should be about 1.5 times the number of diphones (to allow for enough zero entries) and should be a prime number (for an efficient hash function).

Each diphone in the data table is preceded by an identification frame. The bytes in this frame have the following meaning:

byte 1 diphone length in frames (excl. id frame)

byte 2 phoneme boundary

byte 3 end of beginsmoothing (0: no smoothing)

byte 4 begin of endsmoothing (length+1: no smoothing)

byte 5 frameduration in 0.1 ms

The Pascal declarations and constants are currently:

```
key_len  = 8;        { diphone name length in hash-table }
hash_len = 7;        { number of characters used in hash function}
priem    = 3001;     { size hash table }
priem_1  = 3000;     { priem -1 }
```

The hash function is:

```
FUNCTION HASH (s: str): integer;
VAR h, i: integer;
BEGIN { hash }
  h:= 0;
```

```
  for i:= 1 to hash_len do
    h:= (ord (s [i]) + h * 31) mod priem;
  hash:= h;
END; { hash }
```

5.2.2 Strings

There are a lot of strings declared, most of them as packed array's of char, but with different lengths. Two string types (*sentence* and *phonstr*) are a record containing a buffer (packed array of char) and a length (integer).

5.3 Memory map

The division of the memory area is as follows (Table 10):

Table 10: Memory map.

Address	Component
00000	EPROM0
20000	EPROM1
40000	EPROM2
60000	EPROM3
80000	EPROM4
A0000	EPROM5
C0000	EPROM6
E0000	EPROM7
100000	RAM0
110000	RAM1
120000	Speech synthesizer
130001	Volume
140001	LCD
150001	I/O
150002	Keyboard
160001	Serial interface
170000	Reserved

Note: EPROM5-7 and RAM1 are currently not used.

With the current software and exception lexicon (10000 bytes) the RAM area from 100000 (hex) to 10E490 (hex) is occupied. The stack comes down to 10E6C0 (hex), leaving about 500 bytes free between memory area and stack.

5.4 Unit descriptions

5.4.1 accent_m.p

The procedure *accent_mod*, which performs the sentence accent generation.

5.4.2 chip.p

The procedure *chip* starts the speech output, when the speech is running the interrupt service routine *pcf_interrupt* calculates and sends the consecutive frames.

The actions performed in *chip* are: first it waits until the previous speech is finished using the global flag *speaking*. Then it calls the routines *duration* and *intonatie* to fill the arrays with speed and pitch slope turning points. The diphone array is copied, so a next sentence can already be processed without disturbing the buffer for the current sentence. Then the PCF8200 interrupt is disabled, because the start-up is all done in this routine and not in the interrupt service routine. The interrupt routine is then called to calculate the first frame. The routine *start_pcf* sends the DAC factor. Then the pitch start is sent. After this the PCF8200 interrupt is enabled and this routine is finished.

The interrupt service routine *pcf_interrupt* does the following: the time is checked against the next duration turning point (in *durpnt*). The duration contour is stored as a number of points, each point consisting of a time and a new duration code. The last point has a time equal to the length of the utterance. If a duration turning point is passed, a command is sent to the PCF8200, incorporating this new duration code. The voice selection in the command byte is set according to the global variable *female*. Then it sends the frame calculated in the previous call and calculates a new frame. In this way there is more time available for calculation than when a frame is first calculated when a request occurs. Calculation of a frame is composed of the following steps: if the current frame is the first one of a diphone a check is made if smoothing is allowed. Smoothing is allowed when the previous diphone allows for endsmoothing and the new one allows for beginsmoothing and if the current frame has not the maximum frame duration. If smoothing is allowed the first frame of the new diphone is skipped and the second gets a longer frame duration. This should be the double of the original duration, but this is only true for original durations of 12.8 ms, as possible values are: 12.8, 25.6, 38.4 and 64 ms. The information about the number of frames that can be smoothed is not used, it is always one. Then the time *sprchip*

from the beginning of the utterance is checked against the next intonation turning point (in *kntpnt*). The intonation contour is stored as a number of points, each consisting of a time and a slope. The last point has a time equal to the length of the utterance. If the time passed a turning point, a new *freq1* (the frequency at the last turning point) is calculated using the formula:

$$f_{new} = f_{old} * 2^{\frac{slope*time}{12}}$$

where time is the time interval between the two turning points. To speed up this calculation the following approximation is used:

$$2^x = 1 + \ln(2) * x + 1/2 \ln(2)^2 * x^2$$

All times are in 0.1 ms, therefore the 12 in the formula changes to 120,000. For the actual frequency (*freq2*) the same formula is used:

$$freq2 = freq1 * 2^{\frac{slope*time}{120000}}$$

where time is the time interval since the last turning point (*sprint*). This calculation is only done when necessary, i.e. when the frame is a voiced one. Then the pitch interval (*freqdec*) between this frequency (*freq2*) and the frequency of the chip (*freqchip*) is calculated (see figure 15). The calculated value of *freqdec* is then compared with the border values in *pitable1*. These values are the borders between the possible pitch increment/decrement values for the PCF8200, which are stored in *pitable2*. So the calculated (exact) value is rounded of to the nearest value possible with the PCF8200. This value is then inserted in the frame that will be sent on the next call. The new frequency of the chip (*freqchip*) is then calculated by using the selected value from *pitable2*.

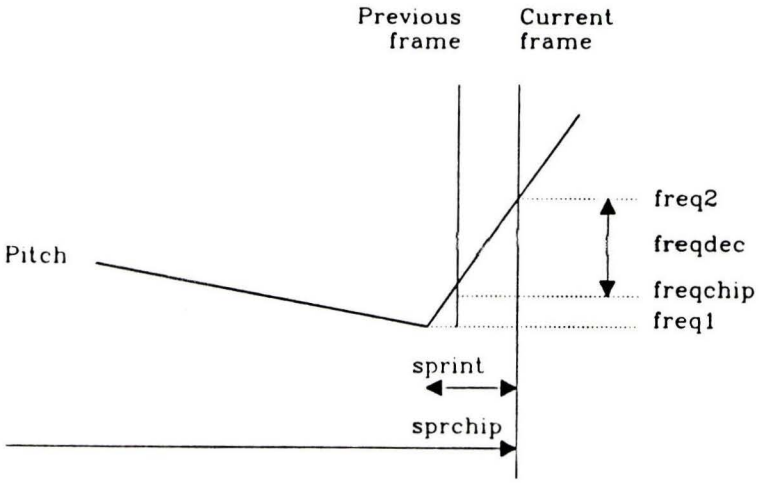


Figure 15: Pitch calculation.

5.4.3 dacia.h

Declaration of all DACIA registers.

5.4.4 data.s

The diphone speech data.

5.4.5 diagn.p

Hardware test module. This software is described in more detail in the technical report.

5.4.6 diphone.s

Routine *get_frame* to read one frame (5 bytes) from diphone data table. This could possibly be done from Pascal also, but then the length of this table should be known and stored as a constant in order to be able to declare this table as a Pascal array.

5.4.7 dir_io.s

Available procedures:

d_read read character from terminal (without echo)

d_h_read read character from host (without echo)

d_write write character to terminal

d_h_write write character to host

spec_write write special character to terminal (ASCII values between 0 and 255)

Available functions:

d_test test if a character is available from terminal

d_h_test test if a character is available from host

The I/O is synchronized by means of the XON/XOFF protocol. The meaning of the flags used for this purpose are:

wait1 output to terminal is stopped (XOFF received)

wait2 output to host is stopped (XOFF received)

wait3 inputbuffer from terminal full, XOFF was sent

wait4 inputbuffer from host full, XOFF was sent

The input is read from a buffer that is filled by the DACIA interrupt service routine. When one of the *d_read* routines empties his buffer and the corresponding wait flag was set, an XON is sent to signal that there is enough space available now to receive characters.

5.4.8 ds.h

Declaration of common Pascal constants and types.

```
maxacc    = 50;        { max number of accents }
max_kntp  = 200;       { 4 turning points per accent }
max_dur   = 50;        { max number of speed changes }
max_char  = 256;       { max stringlength }
maxacco   = 71;        { constants for grapheme-to-phoneme convertor}
maxfeat   = 16;
maxklank  = 50;
max_phon_char = 2;     { max number of char per phoneme }
maxfon    = 750; { length of phoneme string (max_char*(max_phon_char+1))}
LN2       = 0.69315;   { ln(2) for intonation formulas }
LN22      = 0.24023;   { 0.5 * sqr(ln(2)) }
right_margin= 60;      { screen width in screen edit mode }
key_len   = 8;         { diphone name length in hash-table }
hash_len  = 7;         { number of characters used in hash function}
priem     = 3001;      { size hash table }
priem_1   = 3000;      { priem -1 }
max_nr    = 4;         { max number in diphone name }
```

5.4.9 ds.p

This is the main program. The main program loop calls the various parts of the text-to-speech conversion:

kun_grafon grapheme-to-phoneme conversion

gffon phoneme input

fd phoneme-diphone conversion

chip speech synthesis (including duration and intonation)

Except this main program loop this unit takes care of the input, command parsing, execution of some (simple) commands and the initialization.

The text-to-speech routines are also called from the procedure *host*, in which the text going from terminal to host and/or vice versa can be spoken.

A number of variables declared here are global variables: they are also used in other units. In these other units they appear as argument in the unit declaration.

The variable *ram_check* is used to check RAM integrity. After initialization this variable is set to 123456. As long as it has this value, it is assumed that the RAM contents are intact.

This file contains also the main part of the Typestem software. This part is only compiled when a version is generated including the Typestem software.

5.4.10 duration.p

The procedure *duration* fills the duration points array *durpnt*. This array contains the times where the duration changes and the corresponding new duration values. At the moment only a prepause lengthening is implemented and there are two duration values possible: one for the prepause parts (*prepause_dur*), and one for the remaining parts (*mean_dur*). The diphone string is searched for the occurrence of SI phonemes, they signal the end of a prepause part. From such a point the diphone string is searched back until: the begin of the diphone string is reached, an accent is found, two vowels are found or a silence is found, whichever comes first.

5.4.11 edit.p

The procedure *edit* is a screen editor that edits a buffer of type *sentence*. The editor creates a window on the screen consisting of 5 lines of width *right_margin*. The use of a right margin less than the screen width has the advantage that the editor is not influenced by the terminal's auto-wrap setting. The use of the editor is like the Vax editor EDT, but only a limited number of functions are available:
cntrl H : begin of line
cntrl R : refresh screen
and the keypad functions Help, Next Word, End of Line, Delete Line, Delete Word and Delete Character.

This unit contains a few basic routines (*to_right, to_left, up, down, write_str*), which are used in the routines that perform the actual commands.

5.4.12 end_m.p

The procedure *end_mod*, part of the grapheme-to-phoneme converter.

5.4.13 even.s

This unit contains four alignment pseudo instructions to make sure that the length of the sections that are programmed in EPROM is even. This is necessary because the orientation in memory of two neighboring bytes for

the 68000 differs from the output generated by the assembler. Therefore a postprocessing program is used, swapping the bytes two by two, thus requiring an even number of bytes.

5.4.14 fd.p

The procedure *fd* has as input the array *fons*, containing the phonemes and special symbols for an utterance, and as output the arrays *plaats*, containing the diphones, and *accent*, containing the accent points. Each element of the array *fons* contains a phoneme or special symbol (';,?\.). The task of this procedure is twofold:
1) finding the corresponding diphones
2) separating the accent information from the phonemes
The first task is performed in two steps: first the names of two neighboring phonemes are combined to form a diphone name. In the straightforward cases the result is: <phoneme1>1<phoneme2>1. In some cases the second phoneme is altered because the beginning of that phoneme resembles close another one (e.g., D1AI1 would sound the same as D1A1). In the case of plosives a silence phoneme is used (e.g., C1K1 becomes C1SI1). All this is done in the procedure *make_diphone*. The second phase is the lookup of the diphone by *zoekdif*. This procedure looks for the diphone using its name as described in the section diphone tables. If it is not found the procedure *not_found* first tries to locate a diphone with the same name but with the last digit changed (e.g., D1A1, D1A2, D1A3 etc.) up to the constant *max_nr*. If this also results in no diphone found, a silence diphone (SI1SI1) is used instead.

The special symbols are distinguished in accent markers ('), comma markers (,;?) and symbols without accent information (\.). This information is stored in the array *accent* together with the position (diphone number).

It is also possible to enter numbers in the phoneme string to select diphone variants, e.g. A 33 X results in the diphone A3X3 being looked for.

This unit can produce some error messages, these are suppressed in the application mode.

5.4.15 file_io.s

This unit contains three dummy file I/O routines to satisfy the linker.

5.4.16 front_m.p

The procedure *front_mod*, which performs the text pre-processing.

5.4.17 gffon.p

The procedure *gffon* takes care of phoneme input. Its actions are twofold:
1) the addition of a silence phoneme at the begin and end of the utterance
2) the transformation of a string containing the consecutive phonemes into an array.

5.4.18 grafon.p

The procedure *kun_grafon* does the grapheme-to-phoneme conversion. The actual routines with the rules are not in this file because it would be too big to compile then.

The initialization part of this routine, building the feature sets, is only done when the RAM was corrupted. This is done to speed up the conversion. After application of the text pre-processing rules (*front_mod*) the input string is scanned for accent markers, if not found the accent rules (*accent_mod*) are called. Then each word is searched for in the lexicon (by *match*) and if found the preceding unprocessed text is processed by the grapheme-to phoneme rules (*rules* and *rules_mod*) and the phonetic transcription of the word found in the lexicon copied. At the end of each sentence any remaining unprocessed text is processed by the rules. Then the combinations of sentence- and wordaccents are turned into accents and sole wordaccents are removed. Accent signs are also moved forward until they are in front of a vowel. Then the rules in *end_mod* are applied on the whole string.

5.4.19 hardw.s

This unit contains a number of hardware interface routines:

ds_jumper text-to-speech software ?

scr_jumper screen mode ?

test_jumper test mode ?

set_vol write parameter to volume control

batt_empty power too low ?

power_off power off signal to power control

amp_on audio amplifier on

amp_off audio amplifier off

switch_pos test on/off switch position

read_jumpers setting jumper 1 to 6

timer_on run power off timer

timer_off stop power off timer

5.4.20 hash.s

The diphone hash table.

5.4.21 head_em

This (assembly; for some dark reason the .s suffix is not used here) unit is the first file that is read by the linker. It determines therefore to a large extent the total program layout. It is also used to map the software to the hardware.

The first part of this file is the declaration of all sections used. Because the linker determines the order of all sections when they are first encountered, all sections are declared here in the order wanted. The section layout is (Table 11):

Table 11: Section layout.

Address	Section	Function
0	.reset	reset vector
64	.level1_int	interrupt autovector 1
68	.level2_int	interrupt autovector 2
6C	.level3_int	interrupt autovector 3
70	.level4_int	interrupt autovector 4
74	.level5_int	interrupt autovector 5
400	.text	program code
	.roda	read only data
	.dif_hash	diphone hash table
	.dif_data	diphone data table
	.end_rom	dummy section
100000	.data	read/write data
	.bss	read/write data
	.lex	exception lexicon
	.end_ram	dummy section

The next part is the initialization. This is a mixture of self written code and code stemming from the original head_em. The latter parts (trap handlers, program parameters) are not used but are kept for compatibility reasons. First of all the output port is initialized, necessary for the power control to maintain power supply. The status of the output port is also kept in RAM to allow for bit manipulation operations. This is not possible directly on the output port, because bit manipulations read a byte, change a bit and write the bit back. The read operation would in this case read the input port however. Initially all ones are written to the keyboard latch to enable the keyboard. Then the on/off switch state is checked. If it is in the off position, all zeros are written to the keyboard latch to disable the keyboard and the power is switched off. Otherwise the initialization continues.

The PCF8200 speech synthesizer is checked if it is not busy which should be the case. If busy, a stop command is sent in order to try to make it ready.

Then the DACIA is initialized (routine *dacia_init*). The baudrate programmed is jumper selectable. In the same routine the input buffers and corresponding pointers are initialized.

This unit contains also some interrupt routines (DACIA and timer). The DACIA interrupt routines (*acia1_int and acia2_int*) first check if the incoming character is an XON or XOFF. These characters are not placed in the input buffer, but used to synchronize the output. They reset or set the

corresponding wait flag. All other characters are placed in the input buffer. When the input buffer becomes full (10 places left) an XOFF is sent to stop the input. An XON is sent when the input buffer is empty again (by *d_read or d_h_read*). The timer interrupt routine (*timer_int*) counts the number of interrupts in the variable *_time_out*. Interrupts occur every 0.1 second. This variable is cleared every time a character is entered from the terminal or keyboard. After a specified time (currently 5 minutes) an action is taken, depending on one of the jumpers a restart or power down.

5.4.22 hex.p

This unit is like chip.p, the main difference is that the speech data is not send to the speech synthesizer, but to the serial port for the host. The data is send in Intel Hex format. An Intel Hex record consists of:
a ":" as record start,
the number of data bytes (one byte),
the address (two bytes),
the record type (00: data, 01: End Of File),
the data bytes (only for data records),
the checksum (two's complement of the sum of all preceding bytes).
Each speech data frame or command is send as a separate record. The data is preceded by a header frame and terminated by a stop command. The header frame consist of (in accordance with the Philips proposal):

byte 1 & 2	length in bytes (incl. this frame)
byte 3	pitch end
byte 4	DAC factor
byte 5	pitch start

The length (in frames) of an utterance cannot be calculated by adding the individual diphones because smoothing will skip some frames. Therefore the whole utterance is processed twice, once to calculate the number of bytes and once to send them. The first time the pitch calculation is omitted because this information is not needed there.

5.4.23 into.p

The procedure *intonatie* generates an intonation contour in the array *kntpnt*. This array contains the times where the slope of the piecewise linear intonation contour changes and the new slopes.

First the declination is calculated using the formulas:

$$D = \frac{-1}{0.09 * t + 0.13} \qquad t \leq 4.82\ s$$

$$D = \frac{-1}{0.117 * t} \qquad t > 4.82\ s$$

where D is the declination in semitones/second and t is the sentence length in seconds. The endfrequency of the declination line is 75 Hz for the male voice and 180 Hz for the female voice.

Then the times for the accents are calculated from the accent positions given in diphone positions. Finally the actual patterns are calculated using the algorithm given in figure 16. The patterns used are showed in figure 17.

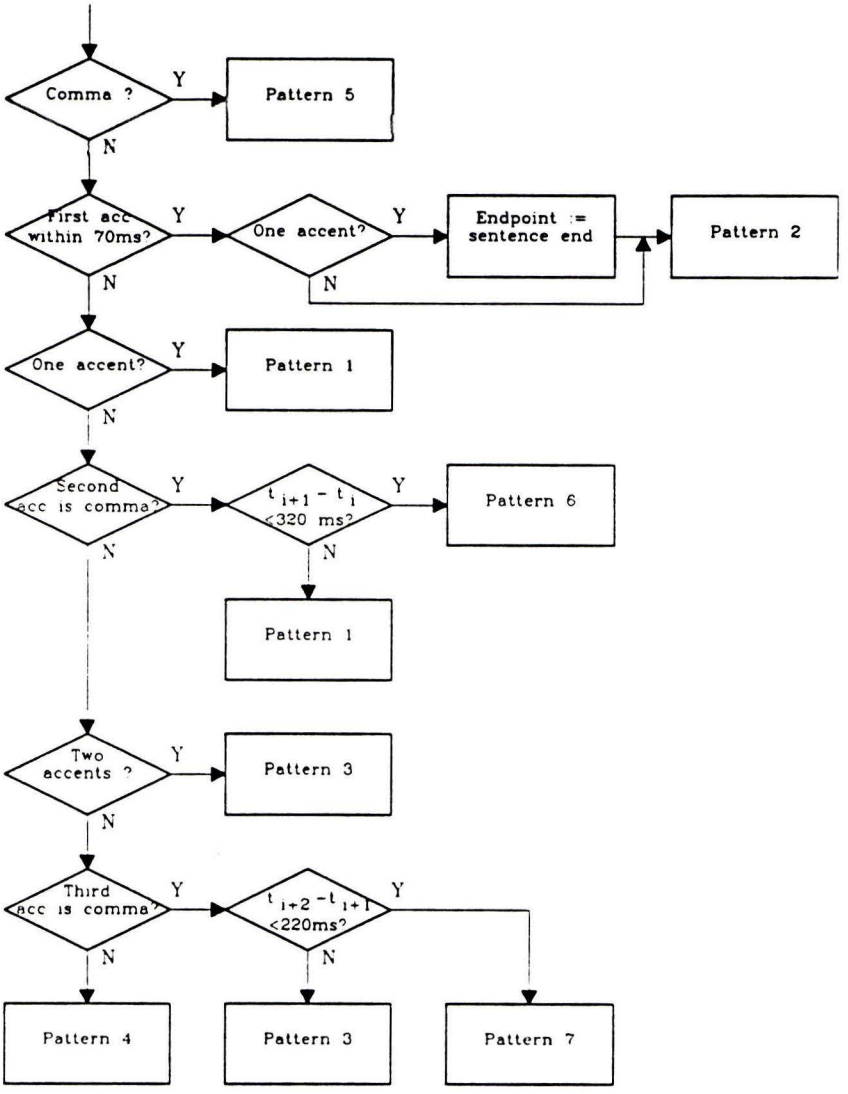


Figure 16: Intonation algorithm.

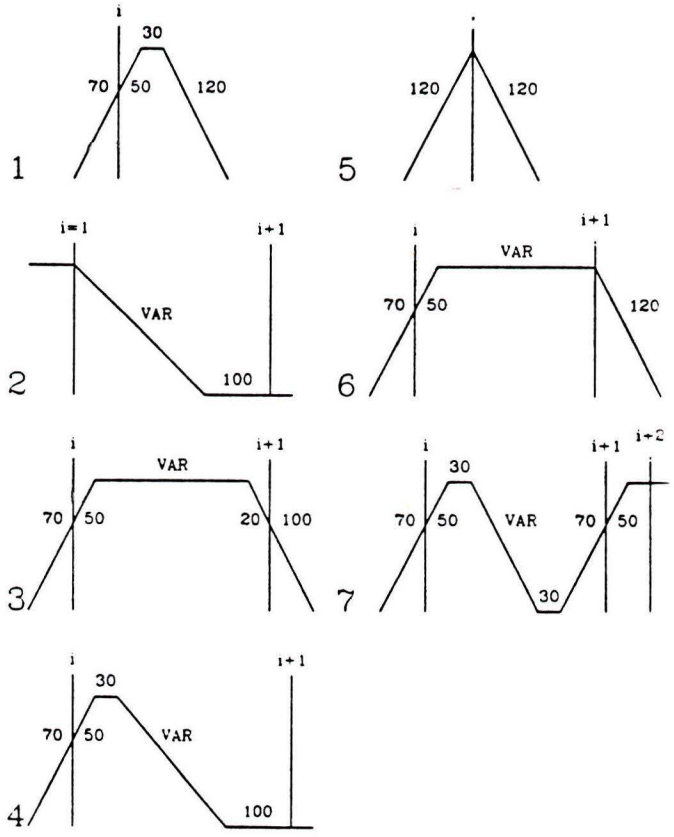


Figure 17: Intonation patterns.

5.4.24 keyb.s

This unit contains the read routine for the keyboard. This routine is an interrupt service routine, called whenever a key is pressed. To prevent this routine from being called all the time when a key is hold down, it starts with disabling the keyboard interrupt. Because this is an interrupt routine, it is of no use to change the status register. The original status register is namely saved on the stack and restored upon completion of the interrupt routine. Therefore the saved status register on the stack is modified. The interrupts are enabled again in the timer interrupt routine, but only when the PCF interrupts are already enabled (level = 1). Otherwise enabling the keyboard interrupt would also enable the PCF interrupt. In this way the

keyboard routine is called every 0.1 sec. when a key is hold down. To allow for debouncing the keyboard is scanned until 500 times the same value is read. Then the key pressed is searched for by making the latch outputs high one by one. The position of the pressed key is then translated into an ASCII code by a table conversion. The character thus found is compared with the previous character. When they are the same the new character is only entered into the input buffer when it is a cursor movement or delete key and the appropriate delay has passed. When this is true or when the characters or when the characters are not the same they are entered into the input buffer. Some characters are not directly copied into the buffer but translated in an appropriate escape sequence. This makes the keyboard input compatible with the terminal input. Some keys act different when used in text-to-speech or "Typestem" mode. In text-to-speech mode they provide some special characters (/+-*%&), while in "Typestem" mode they speak a sentence from the direct speech buffer.

5.4.25 lex.s

This unit contains five routines to access and control the exception lexicon.

match searches the lexicon for a given entry and if found (function result true) returns the phonetic representation for that entry.

next returns the entry that follows the given one. Returns false if the end of the lexicon is reached.

insert inserts the given entry at the correct place. Returns false if the lexicon is full or the entry is already present.

delete deletes the given entry (if present).

lex_init clears the lexicon.

Each entry in the lexicon is constructed as follows:
<entry length><grapheme length><graphemes><phoneme length><phonemes>

The last entry has <entry length> zero.

The entries are ordered alphabetically on their graphemes. This has the advantage that a search in the lexicon can be stopped at a certain moment and not the whole lexicon has to be searched. An example:

looking for	lexicon
blik	aap
	blok
	rene

The comparisons made are:

b ↔ a no match, next entry
b ↔ b match, continue with this word
l ↔ l match, continue with this word
i ↔ o o is greater than i, blik is certainly not present

5.4.26 lex_edit.p

This procedure is partly an adaptation of the lexicon edit program on the IPO Vax made by J.R. de Pijper. This part is rather straightforward. Added are the upload and download facilities to and from a Vax host computer. The way this uploading and downloading is done is:
- connect to the Vax (H command)
- run Vax interface program on Vax
- back to board (cntrl A)
- give retrieve or store command (R or S)
- enter Vax filename

When in progress, the uploading or downloading can be aborted by pressing any key. The protocol used is the following:
the board starts with an opening message:
!R<filename><CR> (retrieve) or
!S<filename><CR> (store)
The Vax answers with 3<CR> if the file cannot be found (retrieve) or created (store). Otherwise the lexicon is transferred using the following loop:
the sender sends 1<graphemes>!<phonemes>!<checksum><CR> and the receiver answers
A<CR> acknowledge if correct received
N<CR> nack if something went wrong (incorrect checksum)
When the answer was an acknowledge the next entry is send, otherwise the previous one is repeated (without limit). After the whole lexicon is send, the sender sends 2<CR>.

When a receive is aborted, the board sends a cntrl C to the Vax to stop the sending process. When a store is aborted, the board sends 2<CR> to

the Vax to signal end of lexicon.

5.4.27 parse.s

This unit contains the routines *corlengt* and *vgl* for the grapheme-to-phoneme converter. Because these routines are very often used they are rewritten in assembly language in order to speed up the grapheme-to-phoneme conversion. In corlengt it is possible to increment *eindverw* and *inplgt*. They could exceed the array length, so their new value is checked against the upper limit before updating these variables.

5.4.28 pcf.s

This unit contains the routines to control the PCF8200 speech synthesizer.

to_pcf sends one byte to the PCF

stop_pcf sends stop command to the PCF and waits until it is stopped

start_pcf tests if the PCF is busy (it should not). If it is busy the function returns false, otherwise the DAC factor is send and the function returns true

enable_pcf sets the interrupt mask to 0

disable_pcf sets the interrupt mask to 2 and stops the timer (to prevent the timer from enabling the interrupt again)

5.4.29 rdstr.p

The procedure *rdstr* reads a complete line of input (until a CR or LF) with echo and backspace possibility. It has three modes:
0 : untranslated input
1 : uppercase translated into lowercase
2 : lowercase translated into uppercase

5.4.30 rules_m.p

The procedure *rules_mod*, which performs the actual grapheme-to-phoneme conversion. The rule-procedures are not in this file, because it would be too big to compile then, but in the files *rules_m1.p* and *rules_m2.p*.

5.4.31 screen.h

This file contains the external declarations for all screen routines. This file can be included in units that use screen routines.

5.4.32 screen.p

This unit contains a lot of screen control routines. All routines use escape sequences according to the ANSI (VT100) standard.

erase_screen erase the screen from and including the cursor

appl_keypad put keypad into application mode

num_keypad put keypad into numeric mode

inv_video display the following characters in inverse video

high_intensity display the following characters in high intensity (VT200 only)

underline display the following characters underlined (VT200 only)

norm_video display the following characters normal

cursor_pos move cursor to given position

cursor_up move cursor up a given number of lines

cursor_down move cursor down a given number of lines

cursor_right move cursor right a given number of places

cursor_left move cursor left a given number of places

save_cursor save the current cursor position in terminal memory

restore_cursor restore the cursor to the saved position

delete_lines deletes the given number of lines starting on the line with the cursor

erase_line erase the complete line

scrolling_region set the scrolling region between the given limits

big_upper write the top half of one line in double height, double width characters

big_lower write the bottom half of one line in double height, double width characters

big write one line in double width characters

set_graphics put terminal in DEC graphics mode

set_ascii put terminal in text mode

cursor_off suppress the visible cursor (VT200 only)

cursor_on make the cursor visible (VT200 only)

5.4.33 speech.s

Speech data for the test utterance used by diagn.p.

5.4.34 strings.p

This unit contains some string routines. The Pascal used has no string type available, only a few operations on C-type strings. Therefore packed arrays of char are used instead, sometimes in a record together with a variable indicating the current length.

Stoa converts the C-type strings to the record form (*phonstr*). This allows for an easy filling of the record form with a constant.

Ktoa converts strings of type *key* into the record form *phonstr*.

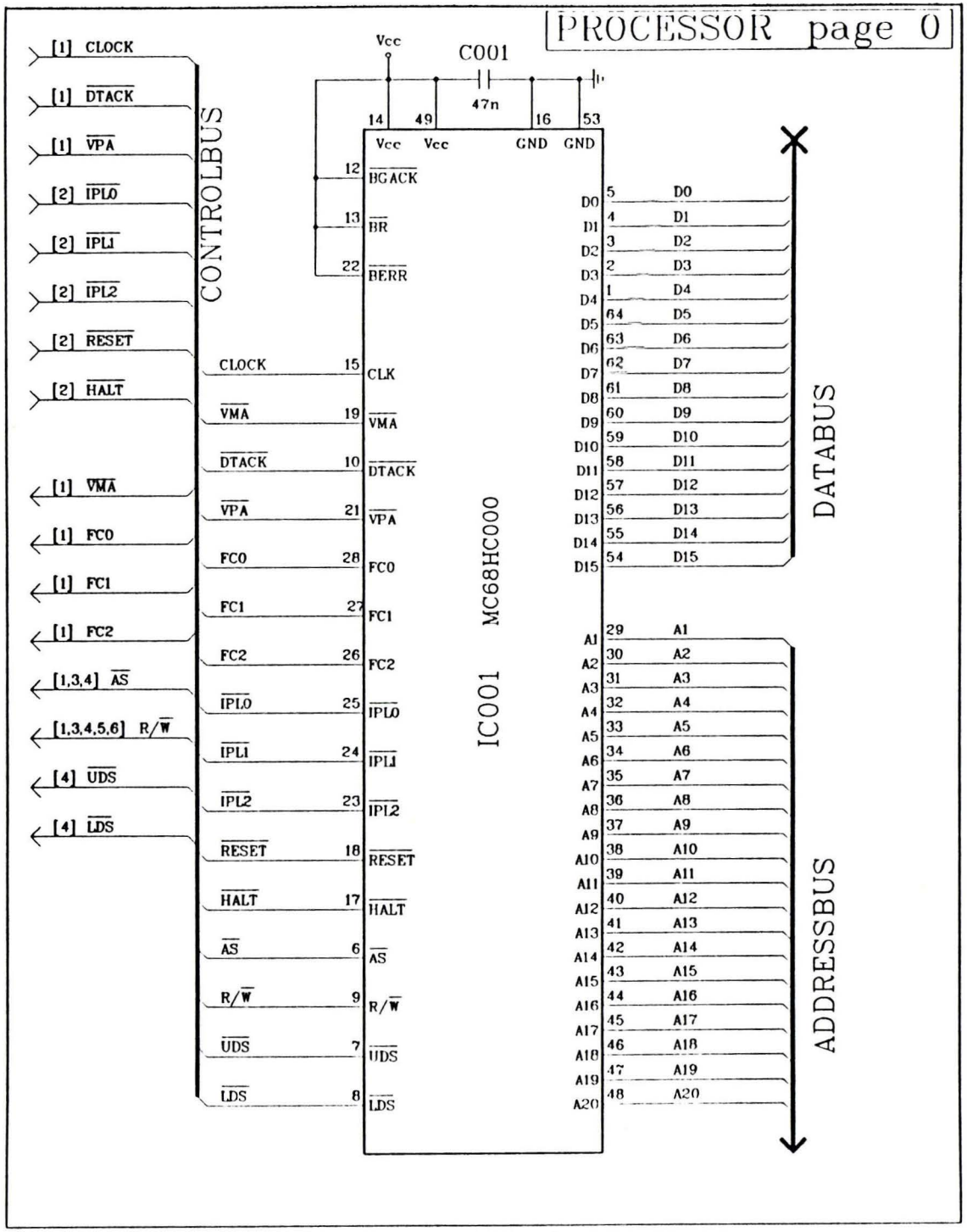
Concat concatenates two strings of type *phonstr*.

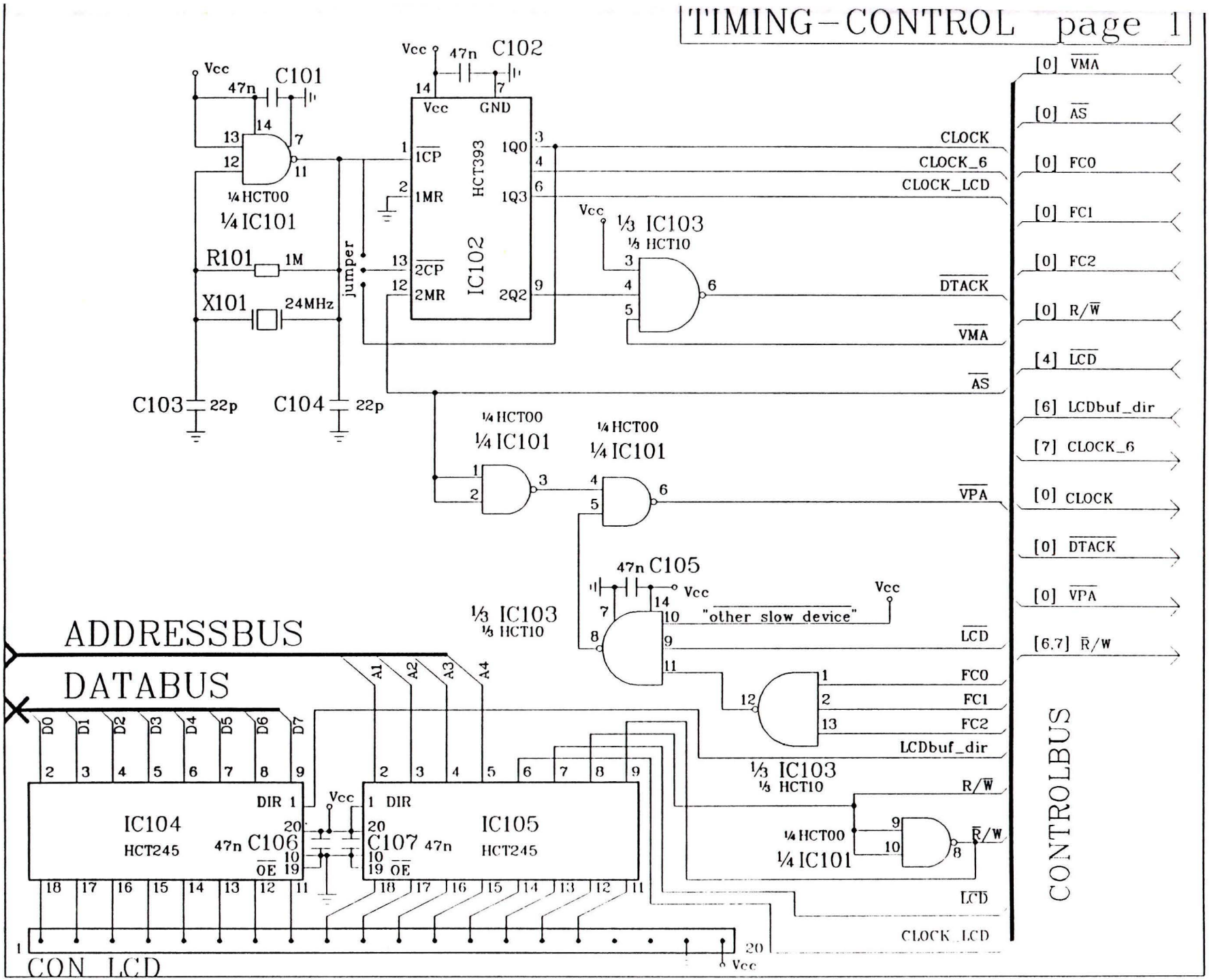
Compa compares two strings of type *phonstr*.

5.4.35 term_io.s

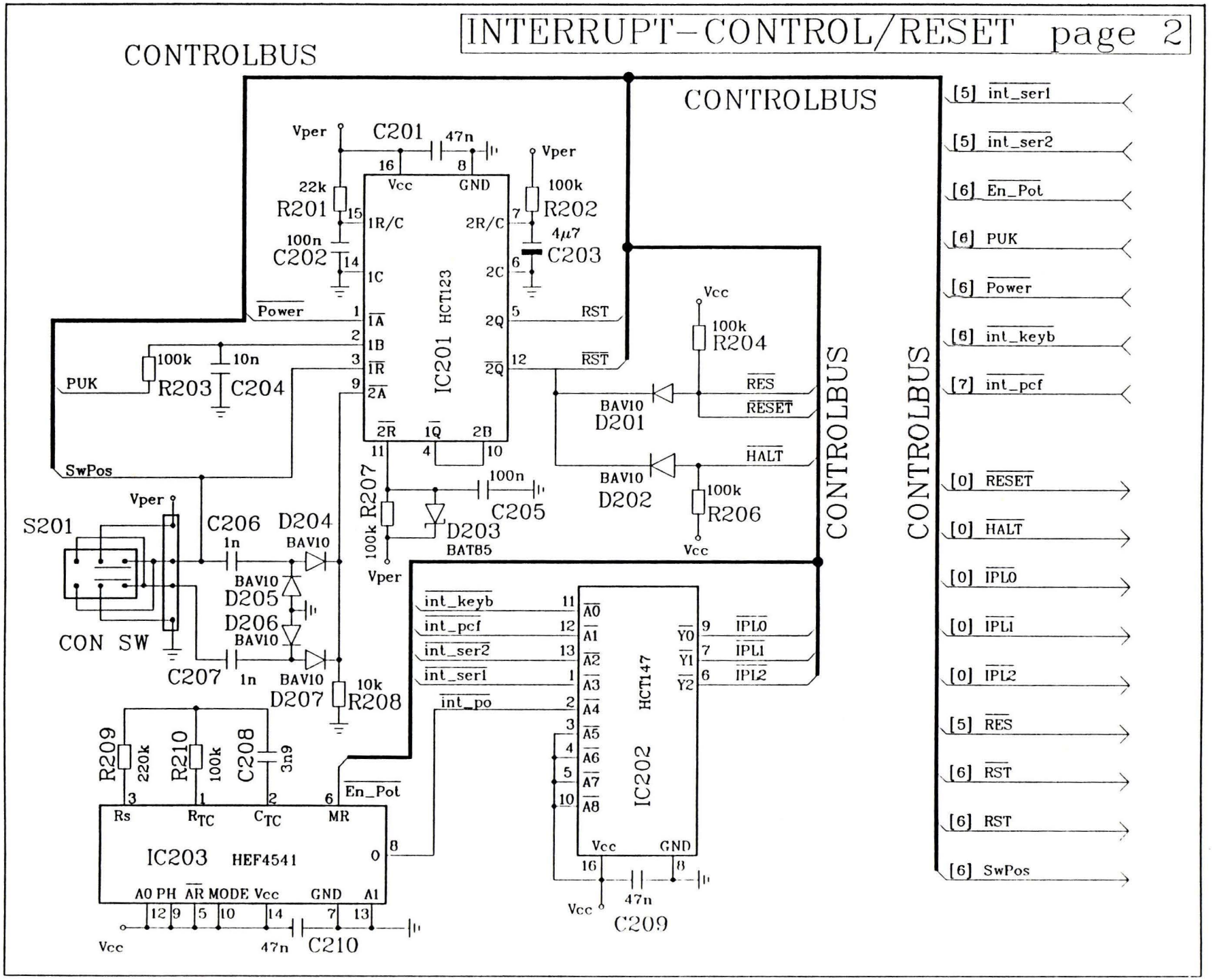
This unit contains the *write* system call, which is used by the Pascal write(ln). Pascal read is not used until now (instead *rdstr* and *d_read* are used), so the read system call is not implemented. In the writeln implementation *_wln* a CR is added.

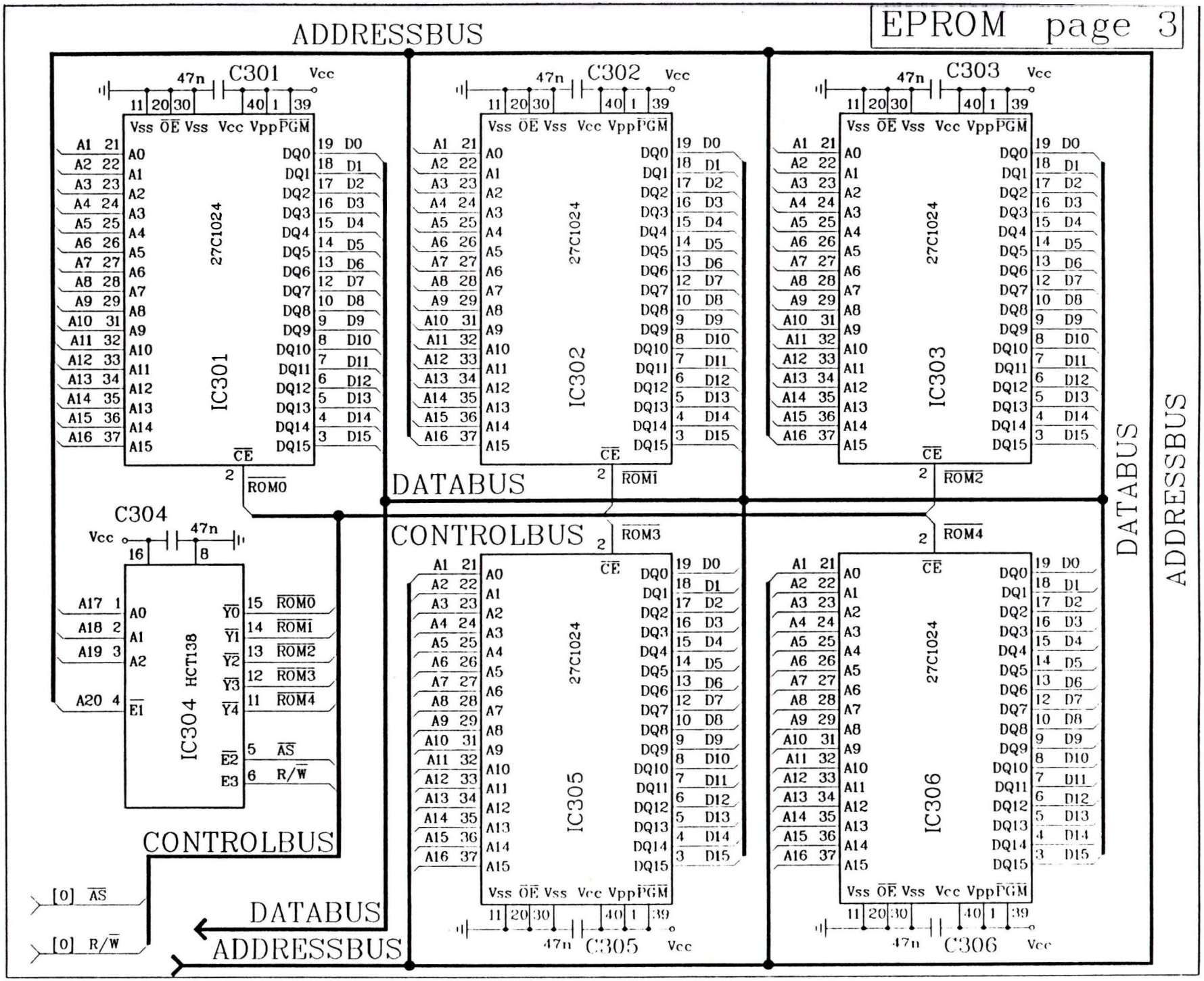
A Schematic diagrams

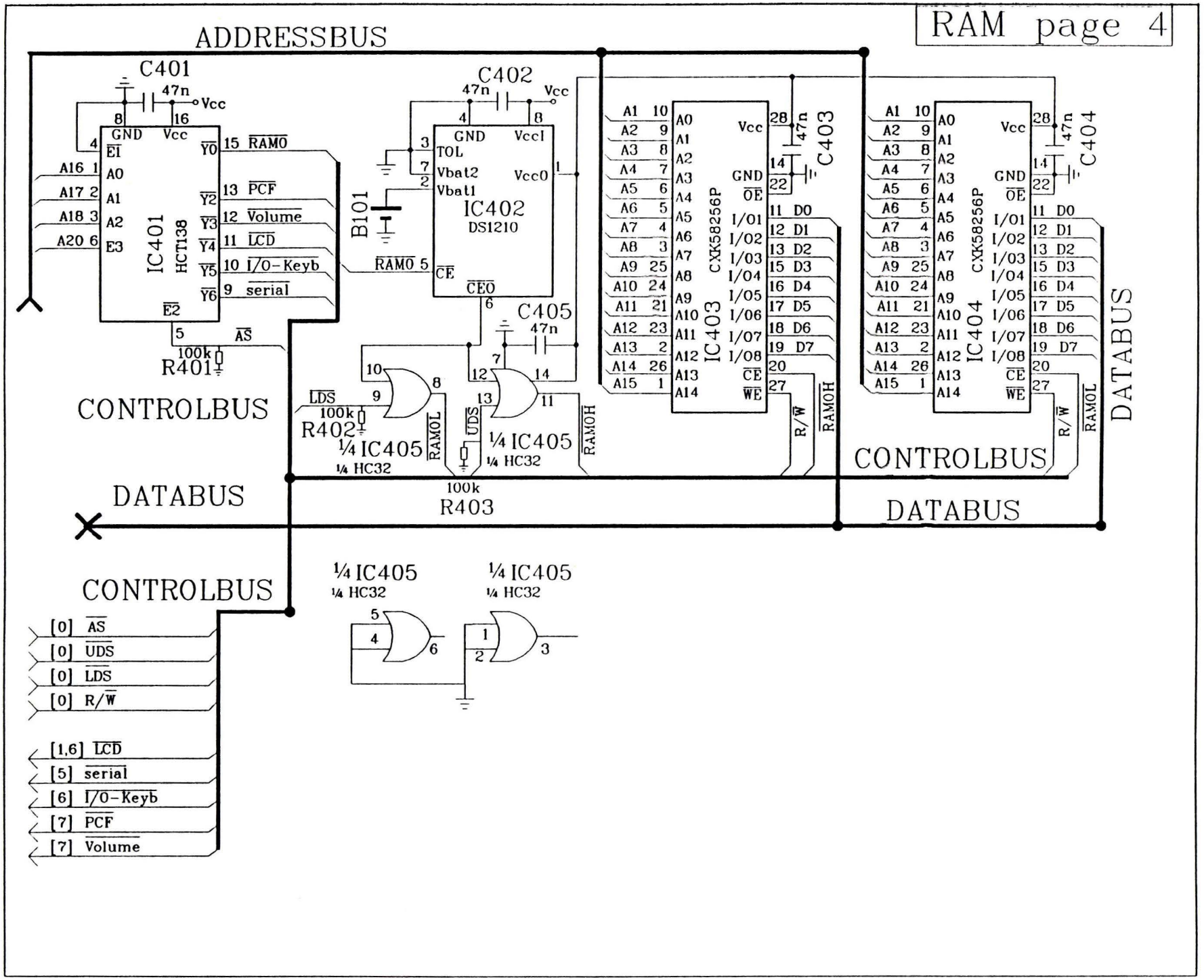

PROCESSOR page 0
Vcc
C001
47n
14 Vcc
49 Vcc
16 GND
53 GND
IC001 MC68HC000
CONTROLBUS
DATABUS
ADDRESSBUS
[1] CLOCK
[1] DTACK
[1] VPA
[2] IPL0
[2] IPL1
[2] IPL2
[2] RESET
[2] HALT
[1] VMA
[1] FC0
[1] FC1
[1] FC2
[1,3,4] AS
[1,3,4,5,6] R/W
[4] UDS
[4] LDS
12 BGACK
13 BR
22 BERR
CLOCK 15 CLK
VMA 19 VMA
DTACK 10 DTACK
VPA 21 VPA
FC0 28 FC0
FC1 27 FC1
FC2 26 FC2
IPL0 25 IPL0
IPL1 24 IPL1
IPL2 23 IPL2
RESET 18 RESET
HALT 17 HALT
AS 6 AS
R/W 9 R/W
UDS 7 UDS
LDS 8 LDS
D0 5 D0
D1 4 D1
D2 3 D2
D3 2 D3
D4 1 D4
D5 64 D5
D6 63 D6
D7 62 D7
D8 61 D8
D9 60 D9
D10 59 D10
D11 58 D11
D12 57 D12
D13 56 D13
D14 55 D14
D15 54 D15
A1 29 A1
A2 30 A2
A3 31 A3
A4 32 A4
A5 33 A5
A6 34 A6
A7 35 A7
A8 36 A8
A9 37 A9
A10 38 A10
A11 39 A11
A12 40 A12
A13 41 A13
A14 42 A14
A15 43 A15
A16 44 A16
A17 45 A17
A18 46 A18
A19 47 A19
A20 48 A20

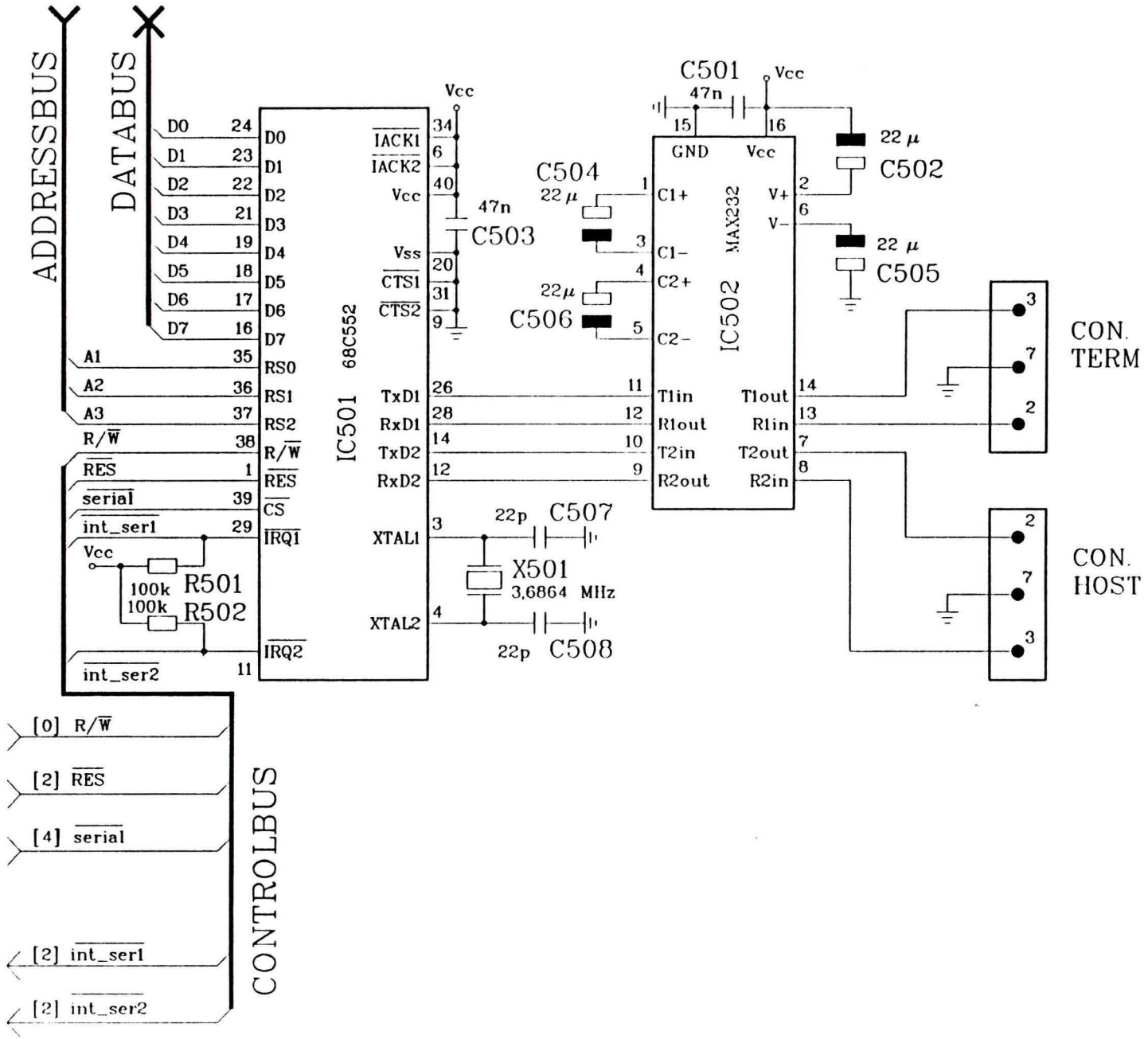

TIMING-CONTROL page 1
Vcc
47n C101
C102
14
7
Vcc GND
13
12
11
1/4 HCT00
1/4 IC101
R101 1M
X101 24MHz
jumper
C103 22p
C104 22p
1CP
1MR
2CP
2MR
HCT393
IC102
1Q0
1Q3
2Q2
CLOCK
CLOCK_6
CLOCK_LCD
1/3 IC103
1/3 HCT10
DTACK
VMA
AS
[0] VMA
[0] AS
[0] FC0
[0] FC1
[0] FC2
[0] R/W
[4] LCD
[6] LCDbuf_dir
[7] CLOCK_6
[0] CLOCK
[0] DTACK
[0] VPA
[6.7] R/W
CONTROLBUS
VPA
47n C105
"other slow device"
LCD
FC0
FC1
FC2
LCDbuf_dir
R/W
ADDRESSBUS
DATABUS
A1
A2
A3
A4
D0
D1
D2
D3
D4
D5
D6
D7
IC104
HCT245
DIR
47n C106
OE
IC105
47n C107
CON LCD
CLOCK_LCD

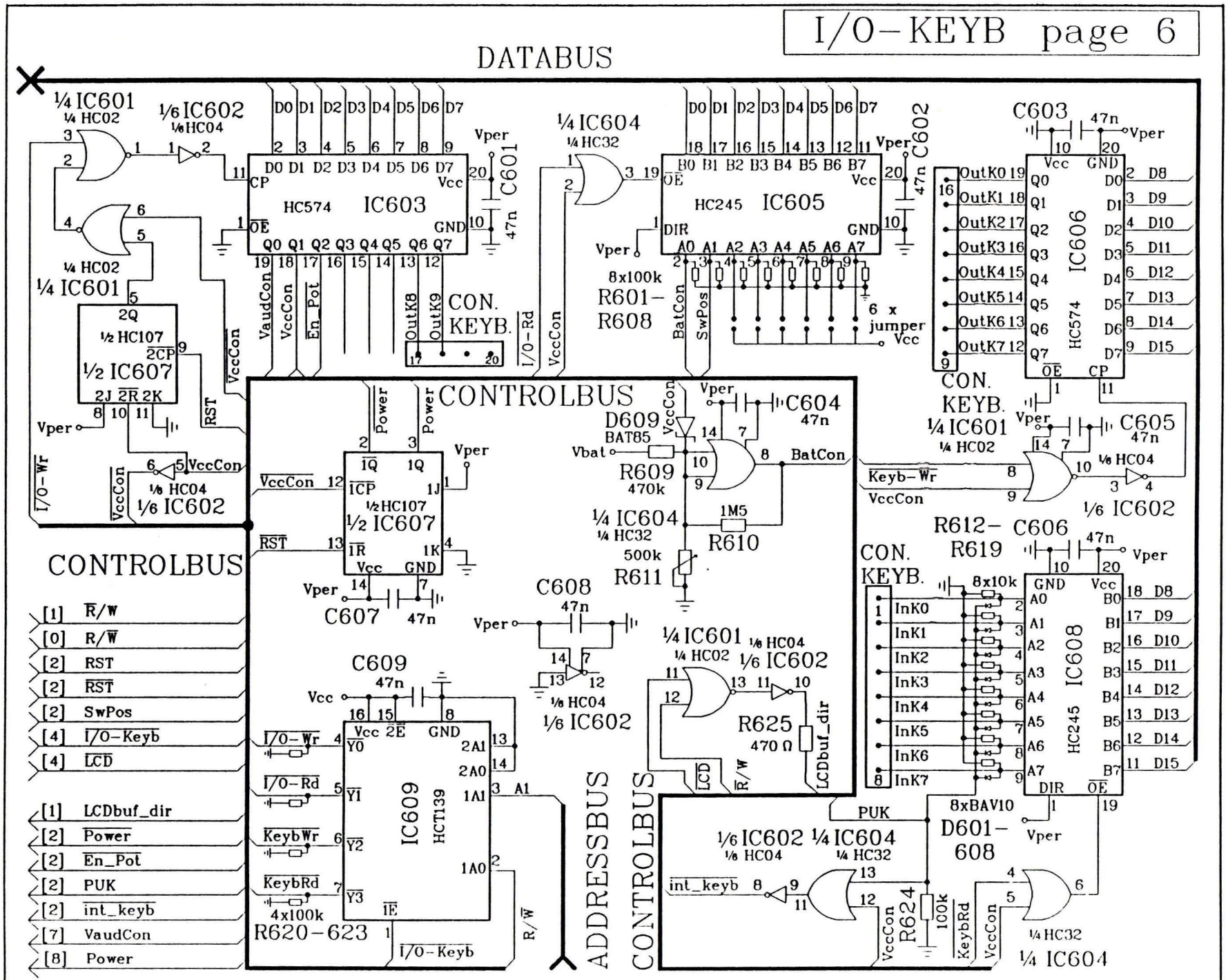
INTERRUPT-CONTROL/RESET page 2

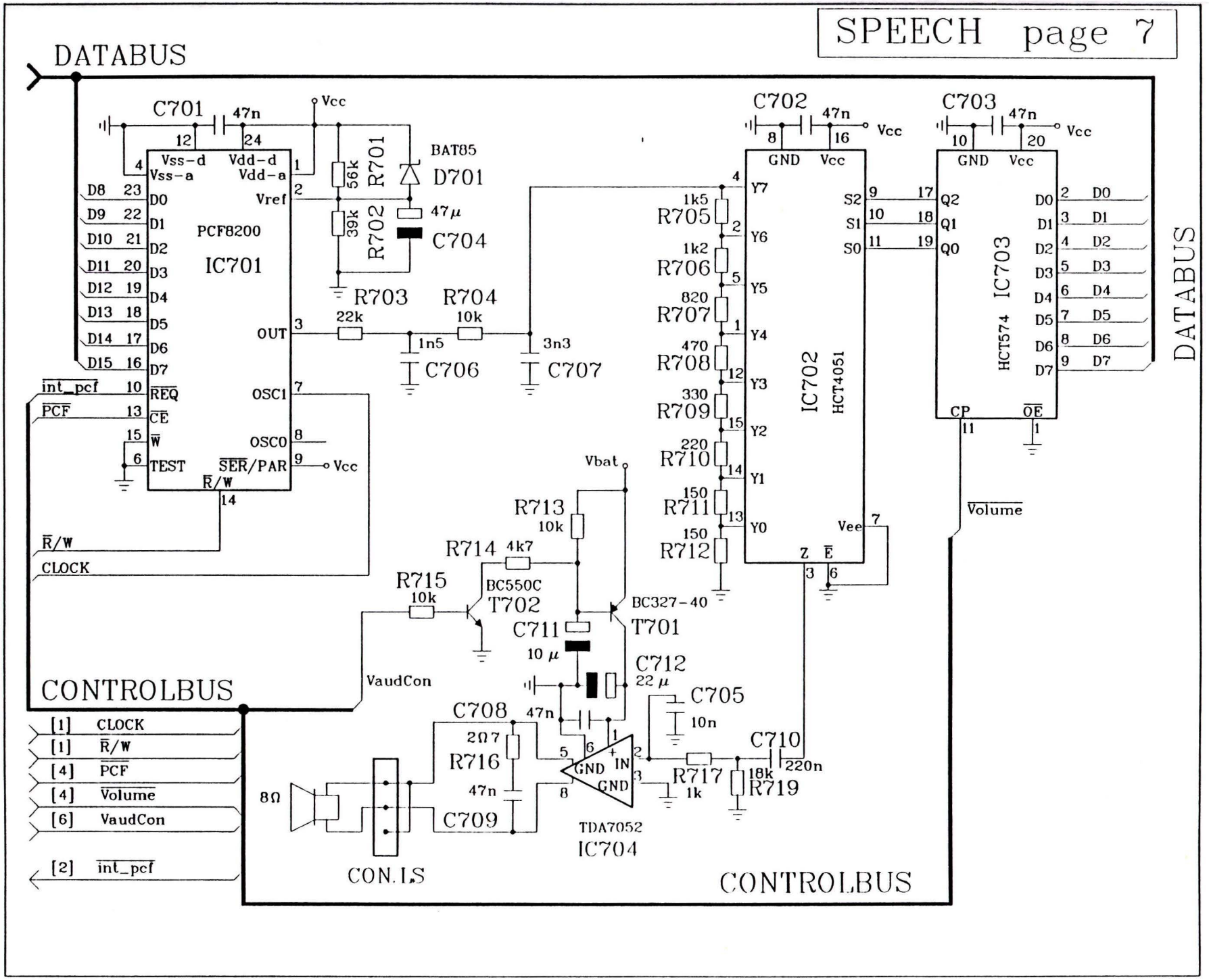



EPROM page 3
ADDRESSBUS
DATABUS
CONTROLBUS
C301
C302
C303
C304
C305
C306
47n
Vcc
IC301
IC302
IC303
IC305
IC306
27C1024
IC304 HCT138
ROM0
ROM1
ROM2
ROM3
ROM4
[0] AS
[0] R/W


ADDRESSBUS
C401
47n
Vcc
IC401
HCT138
GND
Vcc
EI
A0
A1
A2
E3
E2
Y0
Y2
Y3
Y4
Y5
Y6
A16
A17
A18
A20
RAM0
PCF
Volume
LCD
I/O-Keyb
serial
AS
100k
R401
C402
IC402
DS1210
Vcc1
TOL
Vbat2
Vbat1
Vcc0
B101
RAM0
CE
CE0
C405
R402
LDS
UDS
¼ IC405
¼ HC32
RAMOL
RAMOH
100k
R403
C403
C404
IC403
IC404
CXK58256P
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
GND
OE
WE
I/O1
I/O2
I/O3
I/O4
I/O5
I/O6
I/O7
I/O8
D0
D1
D2
D3
D4
D5
D6
D7
R/W
CONTROLBUS
DATABUS
[0] AS
[0] UDS
[0] LDS
[0] R/W
[1,6] LCD
[5] serial
[6] I/O-Keyb
[7] PCF
[7] Volume


ADDRESSBUS
DATABUS
IC501 68C552
IC502 MAX232
D0 24
D1 23
D2 22
D3 21
D4 19
D5 18
D6 17
D7 16
A1 35
A2 36
A3 37
R/W 38
RES 1
serial 39
int_ser1 29
int_ser2 11
IRQ1
IRQ2
CS
R/W
RES
IACK1 34
IACK2 6
Vcc 40
Vss 20
CTS1 31
CTS2 9
TxD1 26
RxD1 28
TxD2 14
RxD2 12
XTAL1 3
XTAL2 4
Vcc
47n C503
C501 47n
C502 22 µ
C504 22 µ
C505 22 µ
C506 22µ
C507 22p
C508 22p
X501 3,6864 MHz
R501 100k
R502 100k
GND
C1+
C1-
C2+
C2-
V+
V-
T1in
R1out
T2in
R2out
T1out
R1in
T2out
R2in
CON. TERM
CON. HOST
[0] R/W
[2] RES
[4] serial
[2] int_ser1
[2] int_ser2
CONTROLBUS


DATABUS
CONTROLBUS
ADDRESSBUS
¼ IC601
¼ HC02
⅙ IC602
⅙ HC04
HC574 IC603
HC245 IC605
HC574 IC606
HC245 IC608
HCT139 IC609
½ IC607
½ HC107
¼ IC604
¼ HC32
C601
C602
C603
C604
C605
C606
C607
C608
C609
47n
Vper
Vcc
GND
CON. KEYB.
8x100k
R601-R608
6 x jumper
R609
470k
1M5
R610
500k
R611
R612-R619
R620-623
4x100k
R624
100k
R625
470 Ω
D609
BAT85
Vbat
D601-608
8xBAV10
PUK
int_keyb
VaudCon
VccCon
En_Pot
OutK8
OutK9
I/O-Rd
I/O-Wr
I/O-Keyb
KeybWr
KeybRd
LCDbuf_dir
BatCon
SwPos
Power
RST
LCD
R/W
[1] R/W
[0] R/W
[2] RST
[2] RST
[2] SwPos
[4] I/O-Keyb
[4] LCD
[1] LCDbuf_dir
[2] Power
[2] En_Pot
[2] PUK
[2] int_keyb
[7] VaudCon
[8] Power


DATABUS
C701
47n
Vcc
PCF8200
IC701
Vss-d
Vss-a
Vdd-d
Vdd-a
Vref
OUT
OSC1
OSC0
REQ
CE
W
TEST
SER/PAR
R/W
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
D14
D15
int_pcf
PCF
R701
56k
R702
39k
BAT85
D701
47µ
C704
R703
22k
R704
10k
1n5
C706
3n3
C707
C702
IC702
HCT4051
GND
Vcc
Vee
Y7
Y6
Y5
Y4
Y3
Y2
Y1
Y0
S2
S1
S0
Z
E
R705
1k5
R706
1k2
R707
820
R708
470
R709
330
R710
220
R711
150
R712
150
C703
IC703
HCT574
Q2
Q1
Q0
CP
OE
Volume
R/W
CLOCK
Vbat
R713
10k
R714
4k7
R715
10k
BC550C
T702
BC327-40
T701
C711
10 µ
C712
22 µ
C705
10n
C708
47n
2Ω7
R716
47n
C709
C710
220n
R717
1k
R719
18k
8Ω
CON.1S
TDA7052
IC704
GND
IN
VaudCon
CONTROLBUS
[1] CLOCK
[1] R/W
[4] PCF
[4] Volume
[6] VaudCon
[2] int_pcf
CONTROLBUS

SUPPLY page 8