

## **Copyright Warning & Restrictions**

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be “used for any purpose other than private study, scholarship, or research.” If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of “fair use” that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

**Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation**

Printing note: If you do not wish to print this page, then select “Pages from: first page # to: last page #” on the print dialog screen

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

## **ABSTRACT**

### **TELEPHONE-ACCESSED CONTROLLER USING CEBUS FOR DEVICE CONTROL OVER POWER LINE**

by

**Gerald Aska**

The CEBus standard has made it possible for devices developed by different manufacturers to communicate over the power line. Further, the standard allows analog adjustments of devices besides transmitting and receiving binary information. This Controller extends the distance from which these devices can be controlled.

To extend the distance of communication with a device, a telephone line interface was developed that allows the user to communicate with a CEBus device via the Controller. The Controller responds to Central Office signaling and opens its communication channel to allow the user to provide it with the commands by using the telephone keypad. The Controller interprets these commands and sends the appropriate information over the power line to the device specified in the command.

To make the Controller user friendly a voice circuit has been included. This circuit provides all the prompts and responses to guide the user for proper operation of the Controller.

**TELEPHONE-ACCESSED CONTROLLER USING CEBUS FOR DEVICE  
CONTROL OVER POWER LINE**

by  
**Gerald Aska**

**A Thesis  
Submitted to the Faculty of  
New Jersey Institute of Technology  
in Partial Fulfillment of the Requirements for the Degree of  
Master of Science in Electrical Engineering**

**Department of Electrical and Computer Engineering**

**August 1998**

**APPROVAL PAGE**

**TELEPHONE-ACCESSED CONTROLLER USING CEBUS FOR DEVICE  
CONTROL OVER POWER LINE**

**by  
Gerald Aska**

---

Dr. Constantine N. Manikopoulos, Thesis Advisor Date  
Associate Professor of Electrical and Computer Engineering, NJIT

---

Dr. Stanley S. Reisman, Committee Member Date  
Professor of Electrical and Computer Engineering, NJIT

---

Dr. Yun-Qing Shi, Committee Member Date  
Associate Professor of Electrical and Computer Engineering, NJIT

## BIOGRAPHICAL SKETCH

**Author:** Gerald Aska  
**Degree:** Master of Science  
**Date:** August 1998

### **Undergraduate and Graduate Education:**

- Master of Science in Electrical Engineering,  
New Jersey Institute of Technology, Newark, NJ, 1998
- Bachelor of Science in Engineering Technology,  
New Jersey Institute of Technology, Newark, NJ, 1996

**Major:** Electrical Engineering

To my dear wife, Pauline, and mother-in-law, Cynthia, for their invaluable support during my studies at NJIT. To my daughter, Abigail, from whom time was taken so that I could complete my studies.

## ACKNOWLEDGMENT

There are a number of persons that I must give special thanks for their contribution to my success in completing this thesis. I thank Dr. Constantine N. Manikopoulos for allowing me to work under his supervision and for providing me with the information that gave me the basic understanding of CEBus. Thanks to Dr. Stanley S. Reisman and Dr. Yun-Qing Shi for consenting to be committee members. Special thanks also to Christopher Onjian for his responsiveness to my questions on CEBus-related issues.



## TABLE OF CONTENTS

Chapter	Page
1 INTRODUCTION .....	1
1.1 Objective .....	1
1.2 Background Information .....	2
2 SPREAD SPECTRUM .....	4
3 HARDWARE DESCRIPTION .....	10
3.1 Power Line Interface .....	12
3.2 Telephone Line Interface .....	18
3.3 Ring Detector .....	23
3.4 Tone Detector.....	29
3.5 Voice Module.....	34
4 SOFTWARE IMPLEMENTATION .....	42
4.1 Bank Memory Implementation .....	42
4.2 Software Development.....	49
4.3 CEBus Message Transfer using CEBench.....	52
4.3.1 Layers Definitions .....	52
4.3.2 CEBench Modifications .....	55
5 USER INTERFACE .....	60
6 CONCLUSION AND FUTURE IMPROVEMENTS .....	65
APPENDIX A CONTROLLER APPLICATION SOURCE CODE.....	68
APPENDIX B CONTROLLER APPLICATION FLOWCHARTS .....	103

**TABLE OF CONTENTS**  
**(Continued)**

<b>Chapter</b>	<b>Page</b>
APPENDIX C COMPILE BATCH FILE.....	113
APPENDIX D CONTEXTS, OBJECTS AND IVS .....	114
APPENDIX E ALTERA SOURCE CODE.....	120
APPENDIX F SCHEMATIC DIAGRAMS .....	122
APPENDIX G PARTS LIST .....	128
REFERENCES .....	132

## LIST OF TABLES

<b>Table</b>	<b>Page</b>
2.1 Spread spectrum symbol code .....	6
3.1 DLL functions implemented by CETHinx part of CENode PL .....	13
3.2 CENode/Host processor commands.. .....	14
3.3 I/O lines and their uses.....	15
3.4 U.S. telephone line operating parameters and limits .....	19
3.5 Truth table of Multivibrator .....	25
3.6 Status register bit description.....	31
3.7 Functional encode/decode table.....	31
3.8 Control logic to access registers .....	33
3.9 Sythesizer Phonemes .....	39
3.10 Phonemes Attribute Tokens.....	40
3.11 V8600 command summary .....	41
4.1 Data transfer between modules in hierarchical program.....	51
5.1 Valid and Invalid conditions for a command code .....	62

## LIST OF FIGURES

<b>Figure</b>	<b>Page</b>
2.1 Spread spectrum signal for a 1 bit .....	5
2.2 Representation of two consecutive 1 bits.....	7
2.3 Bit representaion for the preamble .....	8
2.4 Preamble and preamble EOF signaling .....	8
2.5 Beginning of data using $\phi 1$ and $\phi 2$ for symbols.....	9
3.1 High-level block diagram of Controller .....	10
3.2 Second-level block diagram of Controller .....	11
3.3 CENode PL block diagram.....	13
3.4 Timing diagram of Attention Sequence .....	17
3.5 Timing diagram of read Command Sequence .....	17
3.6 Timing diagram of write Command Sequence.....	18
3.7 Ringing cadence at Tip and Ring.. .....	21
3.8 Ringing cadence at output of DAA.. .....	21
3.9 Telephone Line Interface Circuit.....	22
3.10 Multivibrator used in Ring Detector Circuit .....	24
3.11 Counter circuit used in Ring Detector Circuit.....	26
3.12 Timing relation of Multivibrator and ring cadence of DAA. ....	27
3.13 Tone Detector Circuit.....	30
3.14 V8600 logic symbol .....	35

**LIST OF FIGURES**  
**(continued)**

<b>Figure</b>	<b>Page</b>
3.15 V8600 status flags .....	36
3.16 Microprocessor interface example.....	36
3.17 Parallel Printer Port interface example.....	37
3.19 RS-232 Serial interface example .....	37
4.1 Altera bank memory implementation program (part1) .....	44
4.1 Altera bank memory implementation program (part 2).. .....	45
4.2 Address range of each bank in memory chip.. .....	46
4.3 Memory map of Controller (part 1).....	47
4.3 Memory map of Controller (part 2).....	48
4.4 Hierarchical arrangement of software program.....	50
4.5 CEBench components of CEBus device. ....	53
4.6 Contexts, Objects and IVs used for the Controller (part 1).....	56
4.6 Contexts, Objects and IVs used for the Controller (part 2).....	57
5.1 Operation flowchart of Controller (part 1).....	63
5.1 Operation flowchart of Controller (part 2).....	64

## CHAPTER 1

### INTRODUCTION

#### 1.1 Objective

The objective of this thesis is to present the design and implementation of a Controller that is accessible from the telephone line and send control signals via the power line to a controlled device.

Since the CEBus standard is moving towards becoming a popular standard it will be inevitable that users will need a device that allows them to be able to control devices in the home from a remote location. Currently, X-10 allows remote control of device in the home. The problem is that X-10 is a proprietary standard and only a few devices can communicate using this standard. CEBus on the other hand is an open standard and because it flexibility and capability, that exceeds that of X-10, it is expected that many manufactures will either start or developing home automation devices, using CEBus, adding to the number of existing manufacturers. The Controller, that has been developed, satisfies the need for users to be able to communicate with any CEBus device from a remote location.

In the process of designing the Controller five stages were developed. The first stage is the Telephone Line Interface. It is responsible for line impedance matching and generating and receiving central office signaling. All user code entered from a telephone keypad enters this Interface. The Ring Detector forms the second stage of the Controller. It detects the ring signal from the Telephone Line Interface after it has been reformatted. This circuit determines the number of rings the Controller will allow before responding. The third stage is the Voice Module, which sends synthesized voice prompts and

responses to the user. These prompts and responses guide the user through the operation of the Controller. The fourth stage is the Tone Detector, which detects the code the user enters from the telephone keypad. The dual-tone signal is then converted to a 4-bit digital code so that the microcontroller can interpret it. The final stage of the Controller is the Power Line Interface. This stage is responsible for converting digital-coded information to spread spectrum format in the transmitting mode and from spread spectrum to a digital format in the receiving mode. Spread spectrum is the technology used for power line communication. Since no device currently exist that allows remote control of CEBus devices the design was started from an high-level block diagram and proceeded to the component level design and implementation. Also, the software started with a high-level flowchart to the writing of many lines of code.

An important consideration used in the development of this device is the ease of use. This is the reason for including a synthesized voice feedback to the user. The user knows when to enter a code and receives a feedback on what happen when it was processed. The Controller receives four-digit access and command codes. It was also designed to allow the user to be able to use an extension to control a device that may be in another room of the house that the user is in. This feature was also implemented.

## **1.1 Background Information**

As the market for home automation increases more manufacturers will become involved in developing home automation devices. To avoid the confusion and hardship for consumers to find devices that are compatible to those that may already be in the home it became important to develop a standard among home automation manufacturers. The

standard developed by the Electronics Industries Association (EIA) for home automation is the Consumer Electronic Bus (CEBus) standard. Since it was developed by a standard organization it will facilitate the interoperation of CEBus devices from various manufacturers.

The Controller presented in this thesis incorporates the CEBus standard making it possible to communicate with a CEBus device from a remote location.



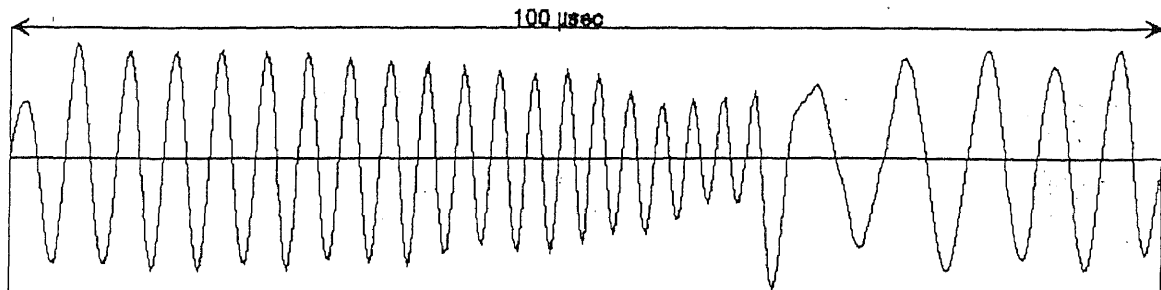
## CHAPTER 2

### SPREAD SPECTRUM

The spread spectrum waveform is a 100kHz to 400kHz bandpass-filtered signal used in CEBus for communication over the power line. A 1 bit is represented by the waveform shown in Figure 2.1. The waveform starts at 200kHz and sweeps to 400kHz then suddenly changes to 100kHz and sweeps to 200kHz. The total sweep from 200kHz through to 200kHz again takes 25 cycles and a duration of 100 $\mu$ sec. This sweep is called a chirp. To meet FCC standard the out of band frequency above 400kHz must have an amplitude of less than 1mV. This is to ensure that the signal does not affect any AM radios that might be connected to the power line. The amplitude of the signal less than 100kHz must be less than 5mV to prevent interference with air and marine radio navigation.

The symbols, 1 and 0, are transmitted along the power line using SUPERIOR and INFERIOR states. Before any packet of data is transmitted on the power line a preamble must precede it. The SUPERIOR state of the preamble is that shown in Figure 2.1 and the INFERIOR state is the absence of the SUPERIOR state. In other words, the type of modulation used for the preamble is Amplitude Shift Keying (ASK). The preamble is a series of ones and zeros used in the sensing of the presence of another node transmitting. The physical layer does this by receiving its own signal. When it sends a signal (SUPERIOR state) it expects to get back a signal and when it does not send a signal

(INFERIOR state) it does not expect to receive a signal. However, if it gets a signal during its INFERIOR state it is an indication that another node is transmitting and that it



**Figure 2.1** Spread spectrum signal for a 1 bit.

must back off. This detection method is called carrier sense multiple access (CSMA). If after all the eight bits of the preamble has been transmitted and no other node has been detected this node sends a preamble EOF (End of Frame) signal to let the other nodes know that it is about to transmit data.

In the data portion of the packet symbols are represented by two SUPERIOR states (phase 1 and phase 2). The signal shown in Figure 2.1 is called SUPERIOR phase 1 ( $\phi_1$ ). SUPERIOR phase 2 ( $\phi_2$ ) is  $180^\circ$  out phase with SUPERIOR  $\phi_1$ . Therefore the modulation method used in the data portion of the packet is Phase Reversal Keying (PRK). In the data portion of the signal there can be no absence of the signal as in the preamble. When the receiver detects a valid signal it then locks on to the signal for the duration of the packet. If the signal became absent the receiver would assume that it has

received the whole packet. However, the packet would eventually be discarded after processing.

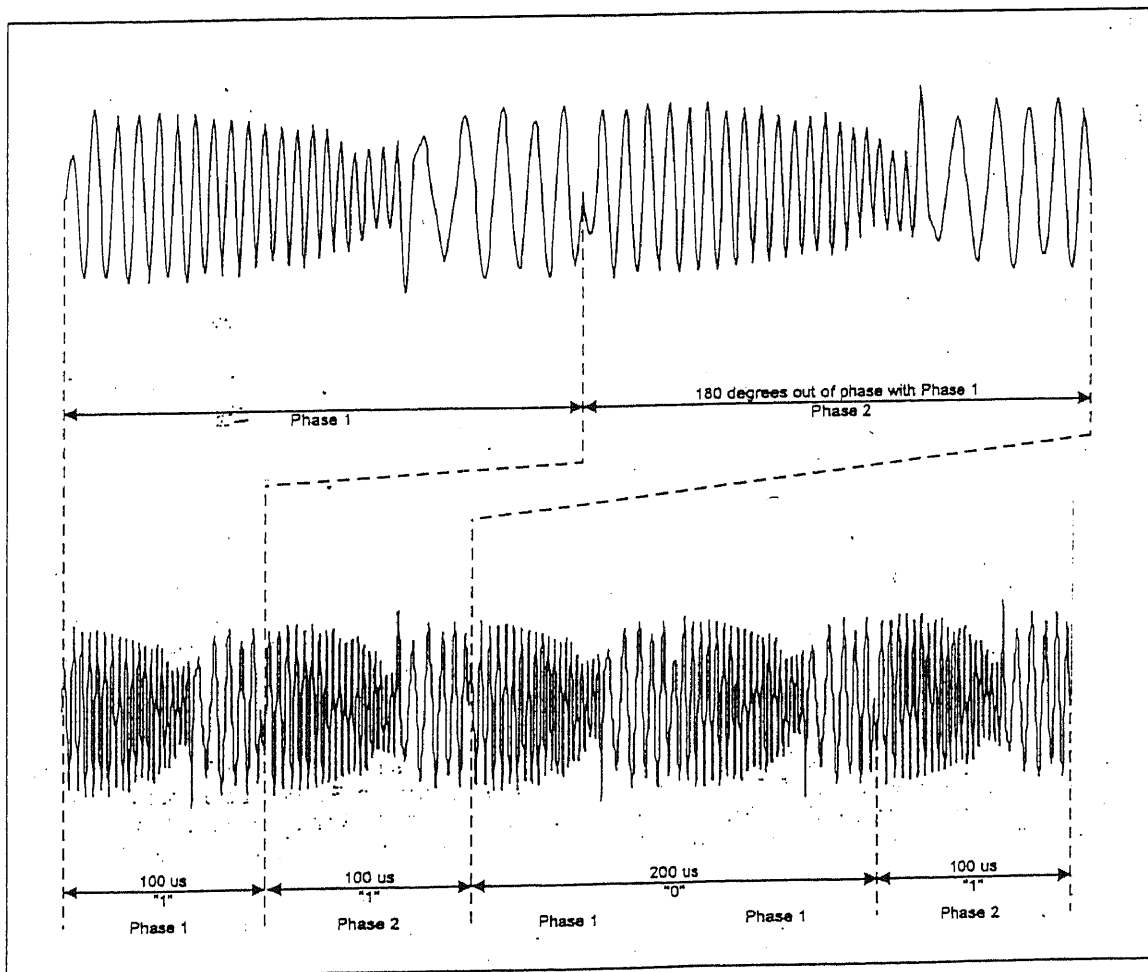
What is important in spread spectrum to represent a bit is the duration of the signal and not the phase. Table 2.1 shows the duration of the signal to represent 0, 1, End

**Table 2.1** Spread spectrum symbol code.

SYMBOL	PREMBLE		DATA	
	UST	Timing	UST	Timing
1	1	114 $\mu$ s	1	100 $\mu$ s
0	2	228 $\mu$ s	2	200 $\mu$ s
EOF	8	800 $\mu$ s	3	300 $\mu$ s
EOP	N/A	N/A	4	400 $\mu$ s

of Frame (EOF), and End of Packet (EOP) for different parts of a packet. A 100 $\mu$ s duration of the signal is called a Unit Symbol Time (UST). Earlier, it was mentioned that a 1 bit is represented by the signal shown in Figure 2.1, however an inverted or 180° out of phase signal may also represent a 1 bit provided that the duration of the signal is 100 $\mu$ s. This inverted symbol is called a INFERIOR or SUPERIOR  $\phi$ 2 (phase 2) state and the non-inverted SUPERIOR state is called the SUPERIOR  $\phi$ 1 state. Spread spectrum alternates between these phases to represent a series of 1 bits. Figure 2.2 shows the spread spectrum signal to represent two consecutive 1 bits. Note that after the first 100 $\mu$ s signal the phase is reversed to represent the next 1 bit. This phase alternating also applies to the 0 symbol which has a duration of 200 $\mu$ s. Also for a 0 bit, whichever phase is representing the signal that phase is continued for the duration of the symbol. The chirp will just be repeated.

As shown in Table 2.1, the duration of the signal that represent a symbol for the preamble is different from that of the data portion. Since a chirp last for only  $100\mu\text{s}$  there is an absence of signal even before the end of the symbol. The states are alternated just as in the data portion of the packet. Also, to represent a 0 bit there are two consecutive absences of the signal. The pattern is shown in Figure 2.3.



**Figure 2.2** Representation of two consecutive 1 bits.

Figure 2.4 shows an example of preamble and the states used to represent the series of bits (01101001). The preamble EOF (PRE\_EOF) is a series of eight ones and is represented by eight SUPERIOR states. The SUPERIOR state is represented by  $\phi 1$  and the INFERIOR state by “---“ in the preamble. After the PRE\_EOF has been transmitted the data portion of the packet follows. This is shown in Figure 2.5.

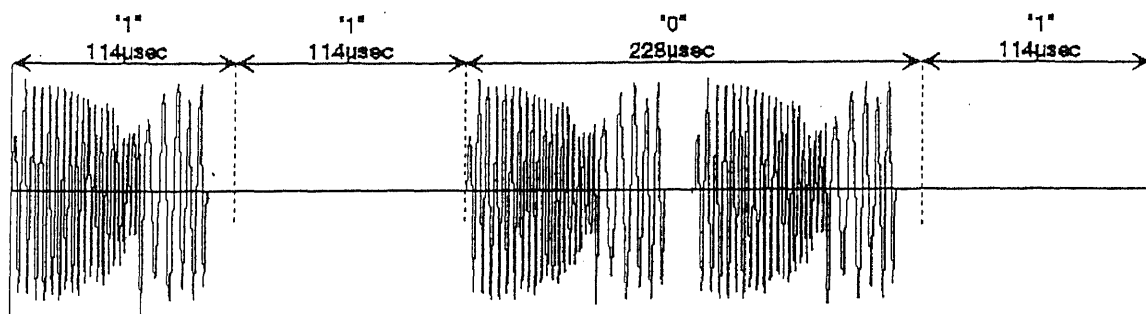


Figure 2.3 Bit representation for the preamble.

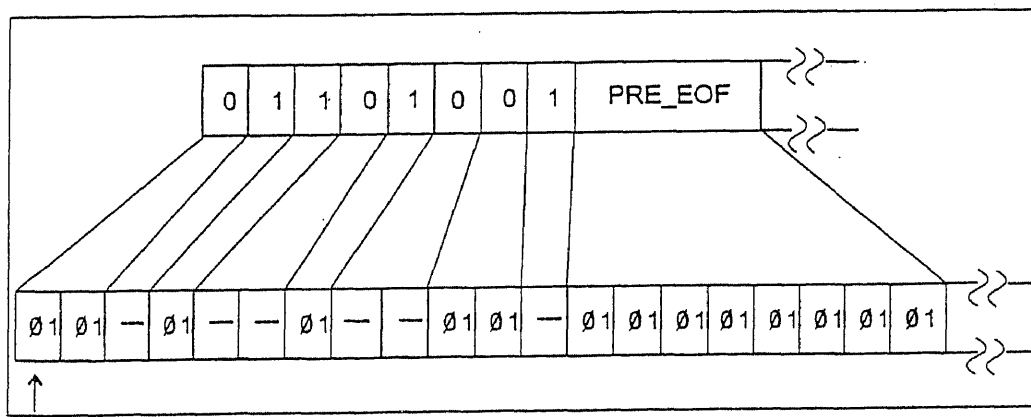
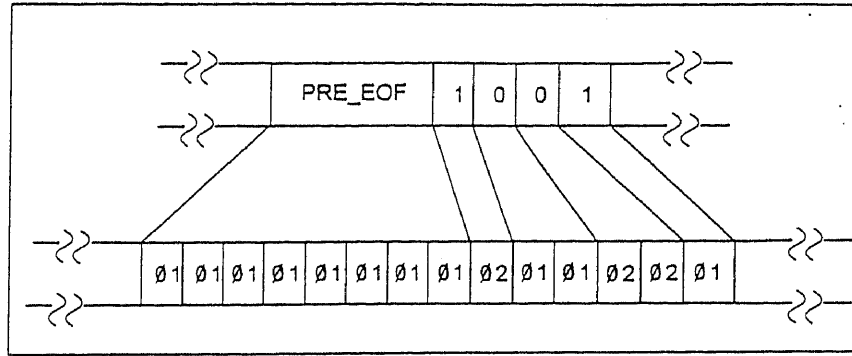


Figure 2.4 Preamble and preamble EOF signaling.

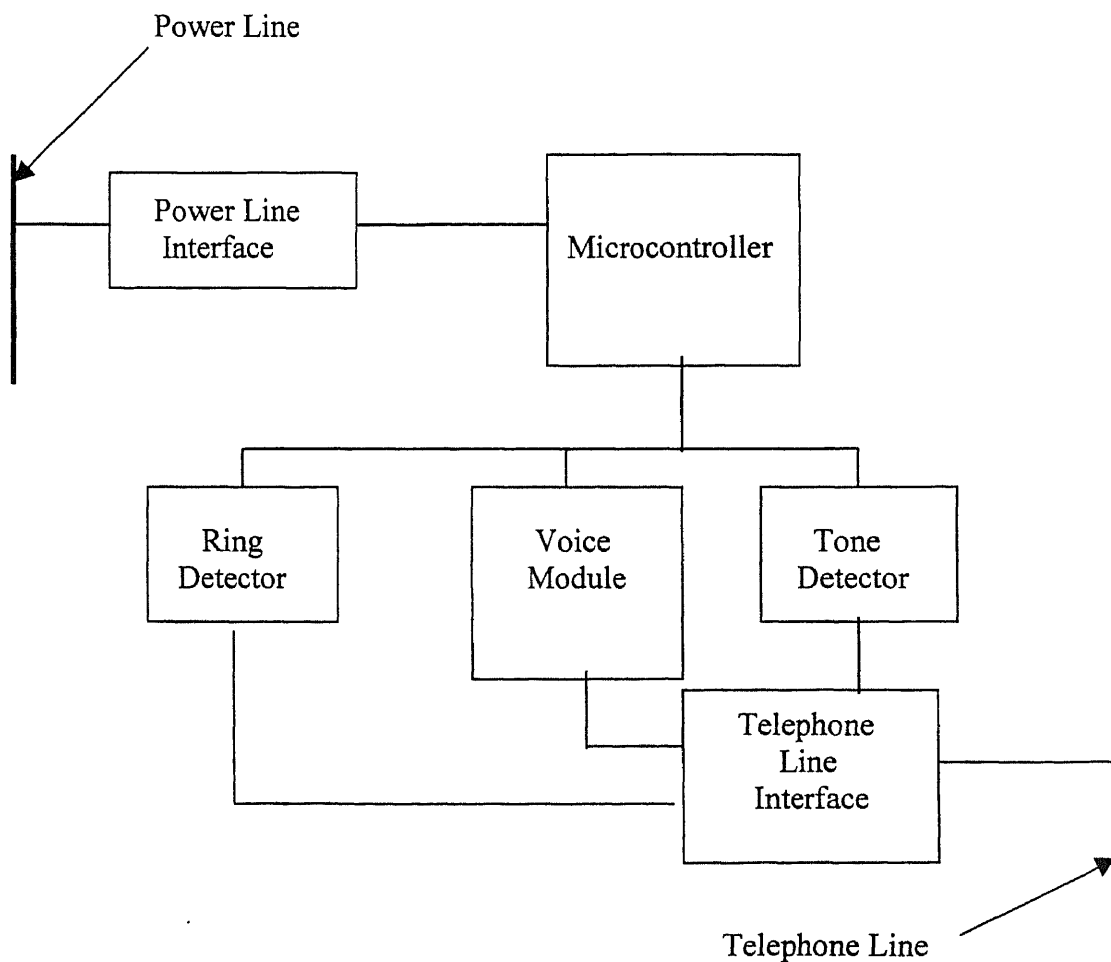


**Figure 2.5** Beginning of data using  $\phi 1$  and  $\phi 2$  for symbols.

## CHAPTER 3

### HARDWARE DESCRIPTION

The high-level block diagram of the Controller is shown in Figure 3.1. The Telephone Line Interface receives signaling information from the telephone company central office and DTMF code from the user at the other end of the line. The Ring Detector counts the number of rings. When the set number of rings are detected the circuit signals the microcontroller. The microcontroller then accesses the telephone line and sends data to



**Figure 3.1** High-level block diagram of Controller.

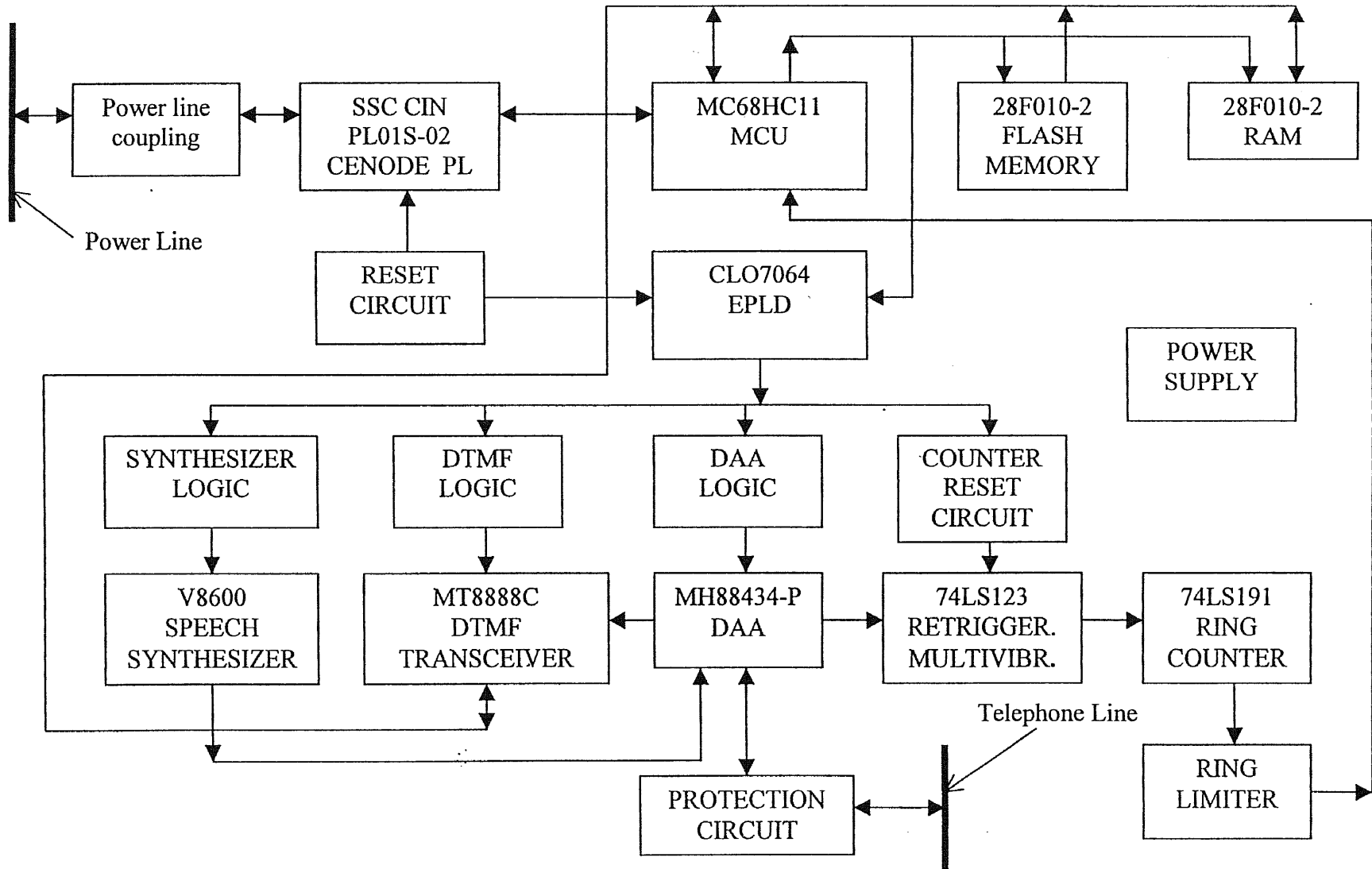


Figure 3.2 Second-level block diagram of Controller.



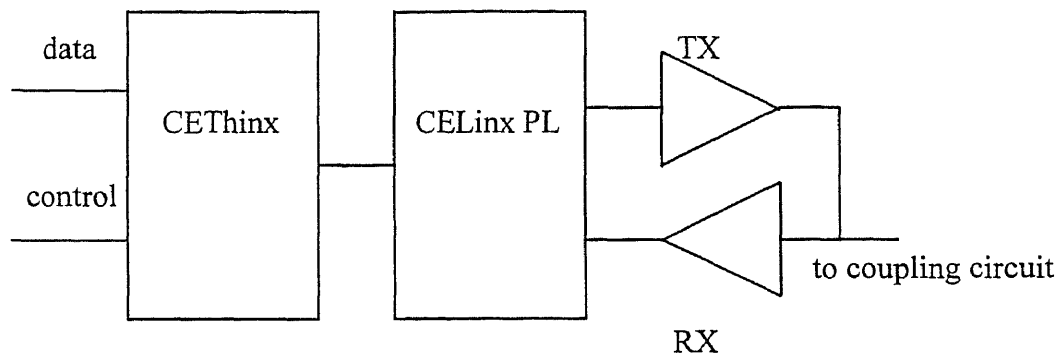
the voice module. The Voice Module transmits an intelligible audio prompt to the user requesting some action. The user responds by pressing appropriate keys on the telephone keypad. The Tone Detector decodes these entries. The inputs to the Tone Detector are DTMF tones. The decoded tones are analyzed by the microcontroller to determine the action to take. The tones must provide information as to which device to operate and what kind of operation to perform. This communication between the controller and the controlled device takes place over the power line. The interface to the power line is provided by the Power Line Interface. Figure 3.2 shows a more detailed block diagram of the Controller. Refer to Appendix F for schematic diagrams. The following sections provide details of the functions of the hardware.

### **3.1 Power Line Interface**

The Power Line Interface consist of a toroid impedance matching transformer, a 60 Hz blocking capacitor, two clamping diodes and the CENode PL Network Interface Card (SSC CIN PL01S-02) made by Intellon. The CENode PL Network Interface Card contains all of the circuitry and processing to implement EIA-600 Data Link Layer and Physical Layer of the CEBus Power Line Specification.

A block diagram of the CENode PL is shown in Figure 3.3. The CENode PL contains the CETHinx that implements the Data Link Layer and the CELinx PL that implements the Physical Layer of the Controller. The CELinx generates and detects spread spectrum waveform. In the detection mode of the CELinx PL the received spread spectrum is compared to an internal representation of the signal. Once a match is detected

the receiver locks on to the signal. This is why no absence of the signal can occur in the information portion of a packet. The receiver can detect signals of amplitudes between 5mV and 7V in the presence of power line noise.



**Figure 3.3** CENode PL block diagram.

The DLL part of the CENode PL is implemented by the CETHinx. The table below (Table 3.1) lists the functions of CETHinx in both transmitting and receiving modes [5]:

**Table 3.1** DLL functions implemented by CETHinx part of CENode PL.

CETHINX FUNCTIONS	
TRANSMITTING MODE	RECEIVING MODE
Generate preamble	Separate packet fragments
Compress data using leading zero suppression	Detect packet errors
Generate FCS for error checking	Compare received and generated FCS
Generate acknowledgement packet	Recognize System and MAC addresses
Access the power line medium using the CSMA/CDCR protocol	Reject duplicate packets

The CENode to Host interface supports 15 commands. The development software used for the Controller sends these commands to the CENode. Table 3.2 shows these commands [4].

**Table 3.2** CENode/Host processor commands.

Command Value	Command Code	Command Name	Description
00H	FIE	Force Interface Error	Used by Factory Testing ONLY.
01H	RST	Reset	Host interface reset of CENode.
02H	LR	Layer Management read	Layer management read of configuration information.
03H	LW	Layer Management write	Layer management write of configuration information.
04H	IR	Interface Read	Host read of interface flags.
05H	CW	Control Wrote	Host write to data link control parameters.
06H	SR	Status Read	Host read of transmit, receive and statistics status information.
07H	--	Reserved	Reserved.
08H	PR	Packet Receive	Host read of received CENode packet.
09H	PT	Packet Transmit	Host write of CENode packet.
0AH	RRI	Read Receive Information	Host read of buffered received packet NDPDU field from CENode.
0BH	WTI	Write Transmit Invoke	Host write of transmit NDPDU field to CENode.
0CH	RRH	Read Receive Header	Host read of data link NDPDU header information from received.
0DH	WTH	Write Transmit Header	Host write of the NDPDU header information for subsequent transmit packets.
0EH-45H	--	Reserved	Reserved.
46H	WR46	Write Register 46	Host write of data link communications access control parameters.
47H-83H	--	Reserved	Reserved.
84H	RR4	Read Register 4	Host read of receive status flags.
85H-FFH	--	Reserved	Reserved.

The logical interface between the Host processor and the CENode uses 13 I/O lines. Eight are bi-directional data lines, four are handshaking, and one is the Reset input. These 13 lines and their uses are listed in Table 3.3 [4].

**Table 3.3** I/O lines and their uses.

<b>Mnemonic</b>	<b>Name</b>	<b>Use</b>
D0-D7	Data pins	Bi-directional data lines. In high impedance state when not in use for transfer.
HSTST*	Host Strobe	Falling edge active strobe from host to CENode. Used by host to initiate transfer of command to CENode in conjunction with HSTWR* signal, and used to indicate data byte available in host write mode, and to acknowledge data byte transfer in host read mode.
HSTWR*	Host Write	Active low. Used to indicate a write mode in conjunction with a Host Strobe signal. HSTWR* inactive (high) with HSTST* indicates end of host write mode, and end of any write command transfer.
DLLST*	CENode Strobe	Falling edge active. (Falling edge active interrupt or latched flag at host). Used in conjunction with DLLWR* by CENode to get host attention and request a command sequence. Also used to indicate data available to host during a host read mode (with DLLWR* active), and to acknowledge the data byte transfer in host write mode.
DLLWR*	CENode Write	Active low. Used to indicate an attention request to the host in conjunction with a DLLST* signal if no transfer in progress. DLLWR* inactive (high) with DLLST* indicates the end of read command data transfer.
Reset	CENode Reset	Reset signal line for CENode. The host may assert this line low (open collector drive) to provide a hardware reset to the CENode.

The time allowed to service a DLLST\* signal from the CENode with a HSTST\* signal from the host is up to 1msec. Beyond this time duration the CENode will time-out. The exception to this rule occurs when DLLST\* is asserted for an Attention Sequence. In this case the HSTST\* response cannot be timed out. The DLLST\* is a fixed pulse of approximately 6 $\mu$ sec duration. The maximum supported transfer rate between the CENode and the host processor is approximately 40Kbytes per second.

The CENode uses an Attention Sequence to cause the host to execute a command sequence. To prevent a race condition, the CENode executes a non-interruptible Attention Sequence by first asserting DLLWR\* and then waiting approximately 15 $\mu$ sec while checking for a HSTST\* signal. If a HSTST\* is seen during the time, DLLWR\* is dropped and the host command sequence is performed. If after approximately 15 $\mu$ sec, no HSTST\* has been seen, the CENode asserts a DLLST\* signal. The next HSTST\* is then interpreted as an acknowledgement of the Attention Sequence. The HSTWR\* should be asserted prior to this time by the host and the CENode will drop the DLLWR\* signal in response to the HSTST\* signal. Figure 3.4 shows the timing diagram for the Attention Sequence.

Although the CENode can cause the host to initiate a command sequence, the host can do it independently and asynchronously. In either case, the host starts a command sequence by putting the command on the data bus and asserting HSTWR\*. If the host is not responding to an Attention DLLST\* it must, in a non-interruptible sequence lasting not more than 15 $\mu$ sec, check that DLLWR\* is not asserted before asserting HSTST\*. If DLLWR\* is asserted it must wait for the Attention DLLST\*, then with a HSTST\*

acknowledge it and indicate a command on the data bus and continue with the command sequence. The host may have to wait up to  $5\mu\text{sec}$  for the initial DLLST\* acknowledging the command. Figures 3.6 and 3.7 show the timing diagrams for the read and the write command sequences respectively [4].

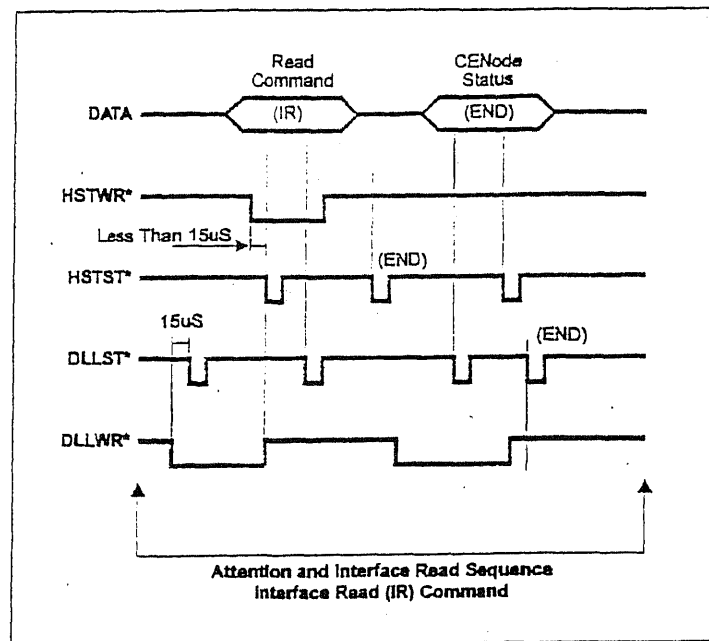


Figure 3.4 Timing diagram of Attention Sequence.

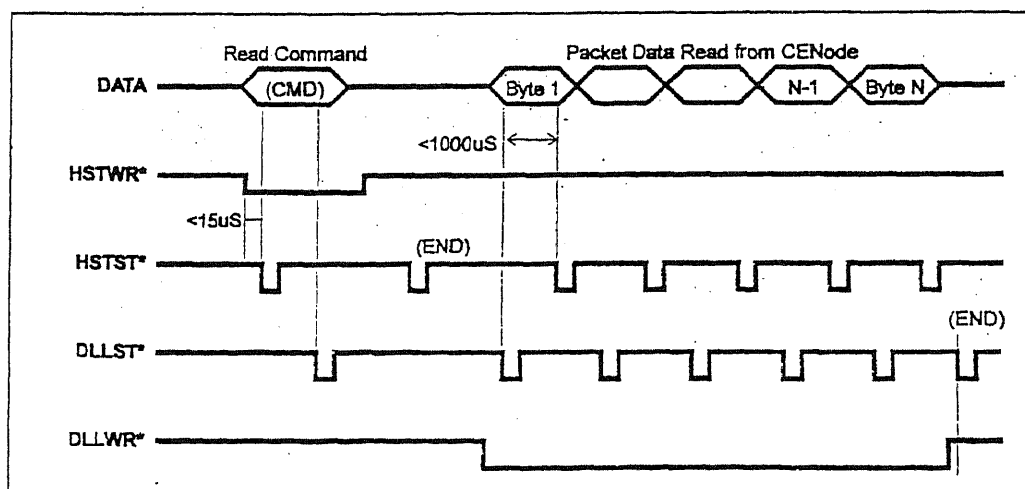


Figure 3.5 Timing diagram of read Command Sequence.

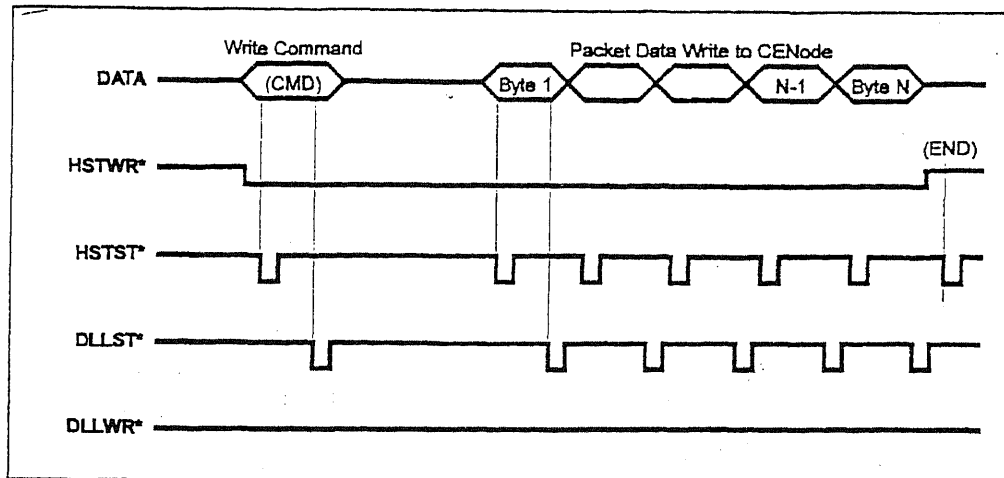


Figure 3.6 Timing diagram of write Command Sequence.

### 3.2 Telephone Line Interface

The Telephone Line Interface is an MH88434-P Data Access Arrangement (DAA) manufactured by Mitel. The device provides isolation to comply with North American standards -- FCC Part 68.304 and DOC CS03 2.2 [1]. Table 3.4 lists the operating parameters and limits for telephone equipment in the United States [16].

To protect the device from damage due to over voltage a P3203AB sidactor (SD1) is connect across Tip and Ring as shown in Figure 3.9. Damaging transient voltage may occur due to lightning surges of up to 1000 volts and induced voltages from, or short circuits to, utility electrical power lines. The sidactor is a very fast clamping device. When it senses high voltages on the line it provide a low impedance in a matter of nanoseconds to protect communication equipment. It will hold the line voltage low until the high voltage that triggered the device reaches a safe level.

**Table 3.4** U.S. telephone line operating parameters and limits

PARAMETER	Typical Values	Limits
Common Battery Voltage	-48 V dc	-47 to 105 V dc
Operating Current	20 to 80 mA	20 to 120 mA
Subscriber Loop Resistance	0 to 1300 ohms	0 to 3600 ohms
Loop Loss	8 dB	17 dB
Distortion	-50 dB total	N/A
Ringing Signal	20 Hz, 90 V rms	16 to 60 Hz, 40 to 130 V rms
Receive Sound Pressure Level	70 to 90 dBspl*	130 dBspl
Telephone Set Noise		less than 15 dBmC**

\*dBspl = dB sound pressure level.

\*\*dBmC = dB value of electrical noise referenced to -90dBm measured with C message weighting frequency response.

A resistor, zener diode and capacitor in series are connected across Tip and Ring as a dummy ringer. Its purpose is to provide a load across Tip and Ring.

When Loop Control (LC\*) is at logic 0, a line termination is applied across Tip and Ring. At this logic level the device is in the off-hook state and DC loop currents will flow through the DAA from the central office. The line termination consists of a DC line termination and an AC input impedance. The DC termination is dependent on the loop current and is approximately 300Ω. Zext represents the additional impedance required for



proper impedance matching of the DAA to the line impedance. The following formula was used to calculate the value of  $Z_{ext}$ :

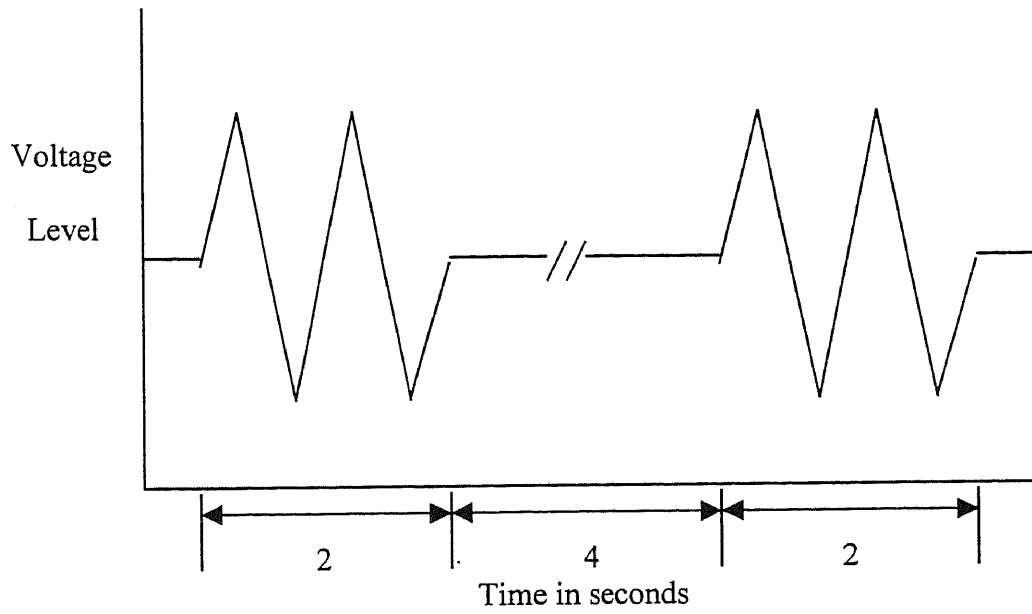
$$\text{Input impedance } Z_{in} = \frac{Z_{ext} + Z_{int}}{10} \quad (3.1)$$

Where  $Z_{int}$  is the internal impedance of the DAA and is equal to  $1.3k\Omega$ .  $Z_{in}$  (the impedance of the telephone line) is equal to  $600\Omega$ .

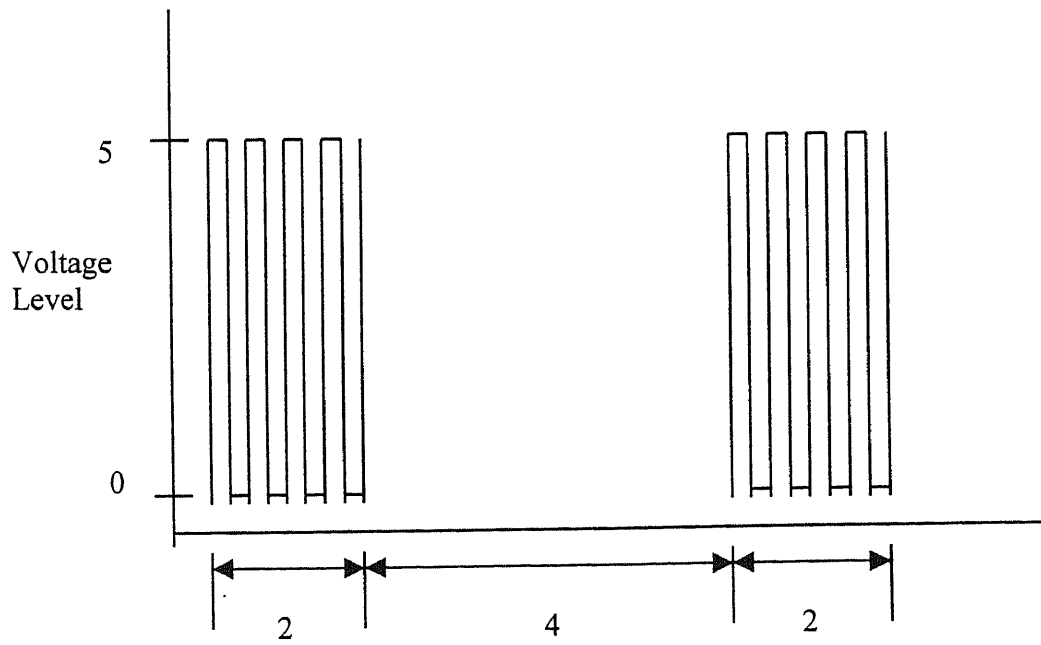
The resistor  $R_2$  sets the sensitivity of the ring voltage detection circuit. Using a  $300k\Omega$  resistor sets the sensitivity to approximately  $20V_{rms}$ . The ring signal from the central office is typically  $90V_{rms}$ . The ring frequency from the central office is between  $16Hz$  and  $60Hz$ . However, the frequency is doubled ( $32Hz$  to  $120Hz$ ) when it comes out of the DAA at the Ring Voltage/Loop Current (RVLC\*) pin. This output ring voltage is at TTL level. Another consideration of the ring signal is the ringing cadence. The ringing cadence for the United States and Europe is 2 seconds ring and 4 seconds silence. Figure 3.7 shows the ring signal from the central office and Figure 3.8 shows the ring voltage at the output of the DAA.

The DAA converts the balanced two-wire input at Tip and Ring to a ground-referenced signal at VX. It also converts the ground-referenced signal at VR to a balanced two-wire signal across Tip and Ring. The transmit (VX) and receive (VR) signals are biased at  $2.5V$ . During full duplex transmission an internal cancellation circuit prevents the signal sent out on TIP and Ring from re-entering on VX.

The transmit gain of the DAA is the gain from the differential signal across Tip and Ring to the ground referenced signal at VX. Resistors  $R_3$  and  $R_4$  alters the gain of

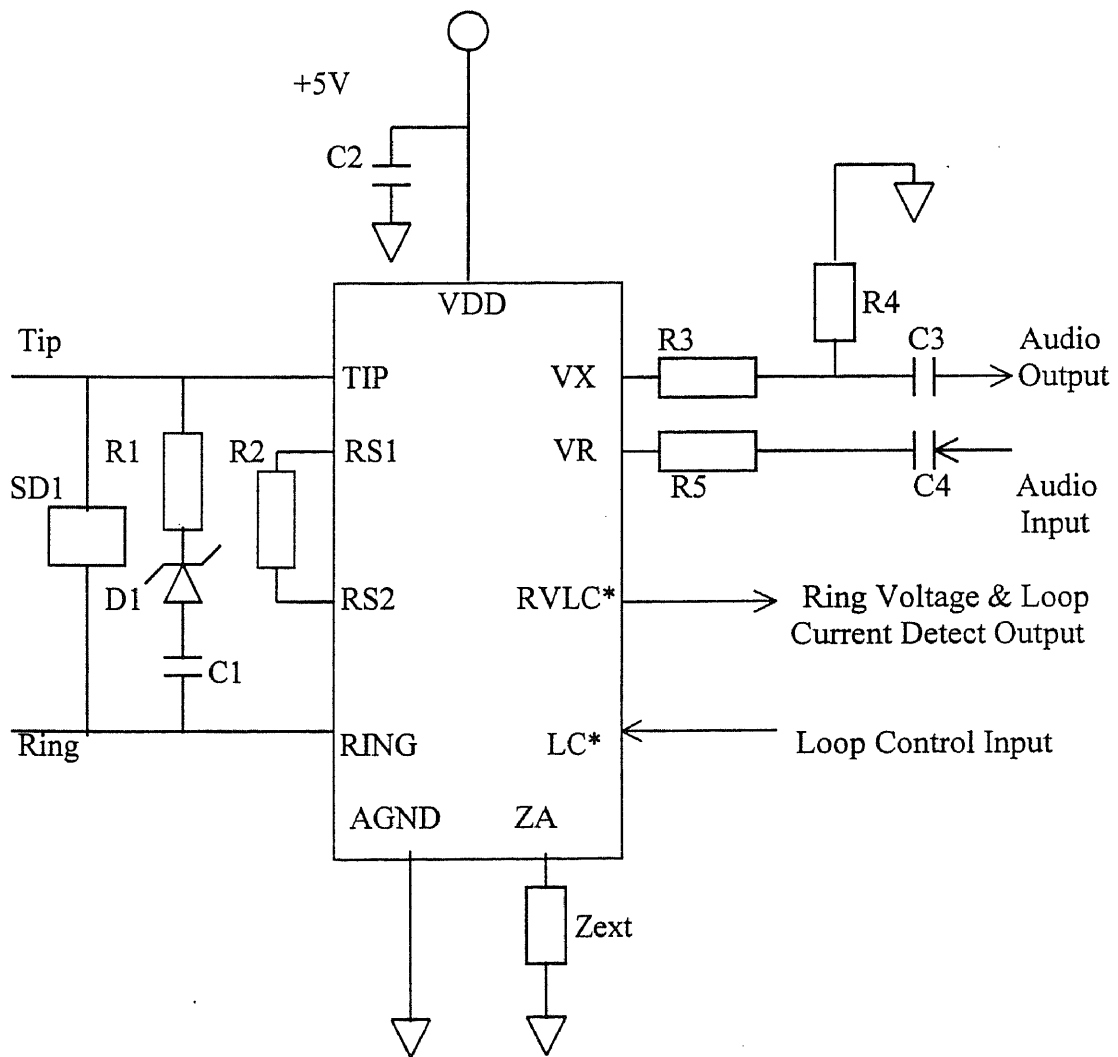


**Figure 3.7** Ringing cadence at Tip and Ring.



**Figure 3.8** Ringing cadence at output of DAA.

the device. The gain of the device was reduced by 50% (3dB) using a voltage divider consisting of two 2k $\Omega$  resistors. The receive gain of the device is the gain from the ground referenced signal at VR to the differential signal across Tip and Ring. The input resistance at VR (to ground) is 47k $\Omega$ . A 100k $\Omega$  resistor was used to reduce the gain by 50% (3dB).



**Figure 3.9** Telephone Line Interface Circuit.

The DAA shown in Figure 3.9 is capable of monitoring the line condition across Tip and Ring. The Ring Voltage/Loop Current (RVLC\*) detect pin indicates the status of the device. The output is a logic 0 when loop current flows indicating that the device is in the off-hook state. The pin will also go low when an extension phone goes off-hook. This feature was used to implement the extension phone mode of the Controller. In the dial-up mode the RVLC\* pin will output ringing voltage which distinguishes it from the extension mode.

When the Controller is on-hook, 48V is across Tip and Ring. When the line is accessed (LC\* low) a low DC impedance of 100 to 400 $\Omega$  causes a loop current to flow. At the time the ring signal is present no loop current flows.

### 3.3 Ring Detector

The major components of the ring detector are the 74LS123 retriggerable monostable multivibrator and the 74LS191 Up/Down Binary Counter. Figure 3.10 shows the circuit diagram for the multivibrator and Table 3.5 shows its truth table [17]. Resistor R1 and capacitor C1 essentially determines the basic output pulse duration. For C1 less than or equal to 1000pF the pulse duration is calculated from:

$$t_w = K * R1 * C1 \quad (3.2)$$

When C1 is greater than or equal to 1 $\mu$ F, the output pulse duration is calculated from:

$$t_w = 0.33 * R1 * C1 \quad (3.3)$$

For the given equations, as applicable:

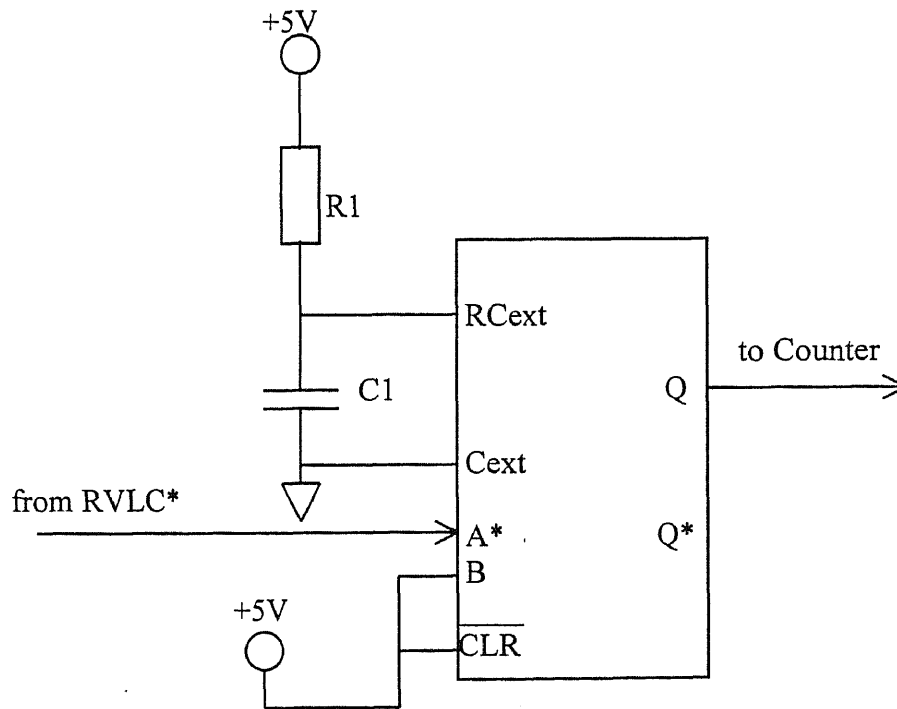
K is the multiplier whose value depends on the expected pulse duration. This value was obtained from the data sheet.

R1 is in  $K\Omega$

C1 is in pF

$t_w$  is in ns







Connecting the Cext pin to ground gives maximum noise immunity even though the device is connected to ground internally. Due to the timing scheme used by the device, a switching diode is not required to prevent reverse biasing when using electrolytic capacitors [17].



**Figure 3.10** Multivibrator used in Ring Detector circuit.

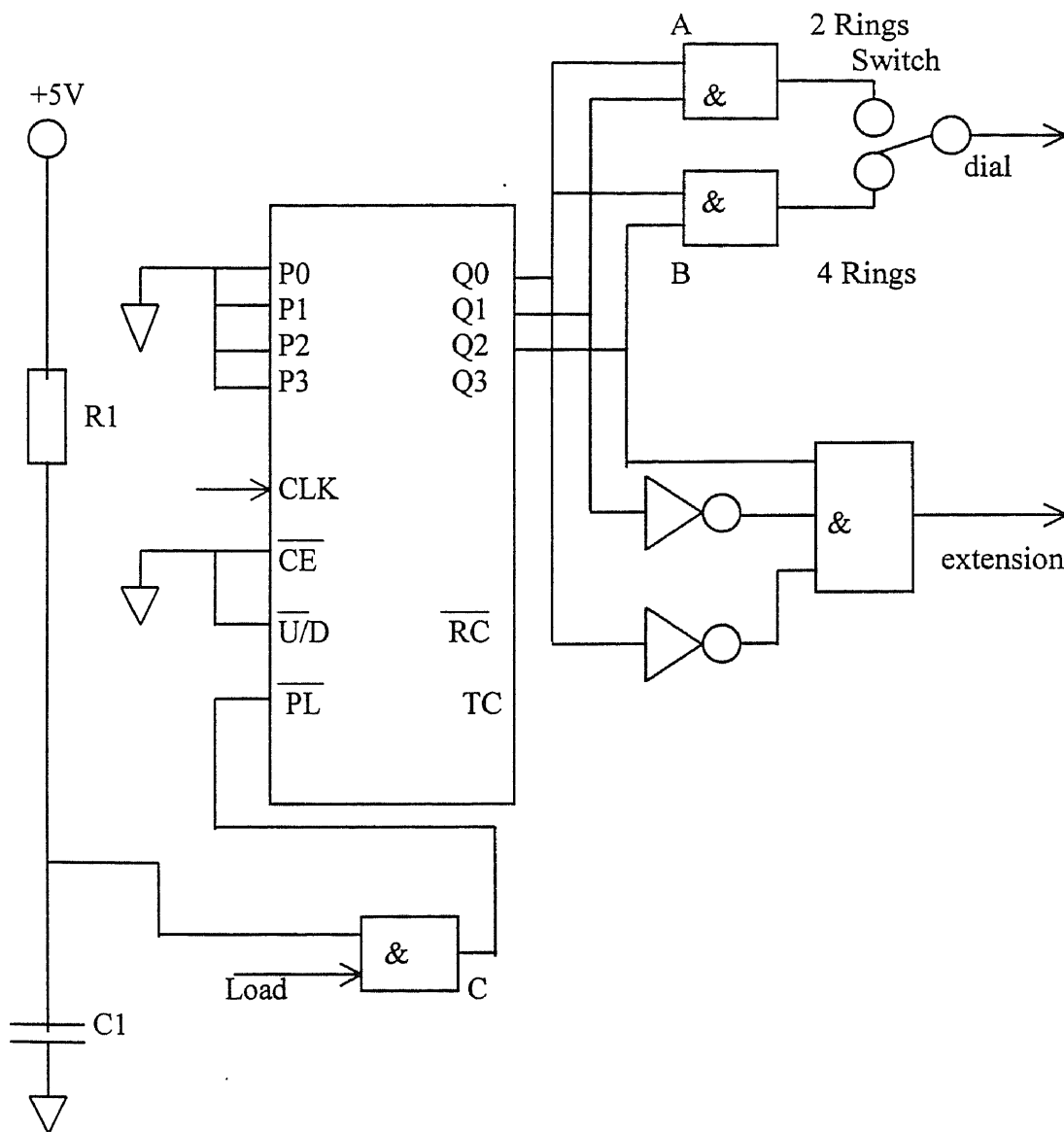
A pulse of 2sec was calculated for the Controller. This duration was used so that the pulsed would reflect the ringing cadence. Each time the ringing signal is present the output of the 74LS123 goes to logic 1 to increment the count. Doing so we can set the number of times the Controller will “ring” before seizing the telephone line. As Figure 3.11 shows the number of rings can be set to either 2 or 4 using AND gates. We will get back to the detail of this circuit a little later. For now we want to look at the response of the multivibrator circuit in relation to the ringing cadence at the output off the DAA. Recall that the multivibrator used is retriggerable. Therefore when the ring stops and the silent period begins the output of the multivibrator will be at logic 1 for 2 more seconds after the ring stops (although it rose to logic 1 at the beginning of the ring). Figure 3.12 shows

**Table 3.5** Truth table of Multivibrator.

INPUTS			OUTPUTS	
CLEAR	A	B	Q	Q*
L	X	X	L	H
X	H	X	L	H
X	X	L	L	H
H	L			
H		H		
	L	H		

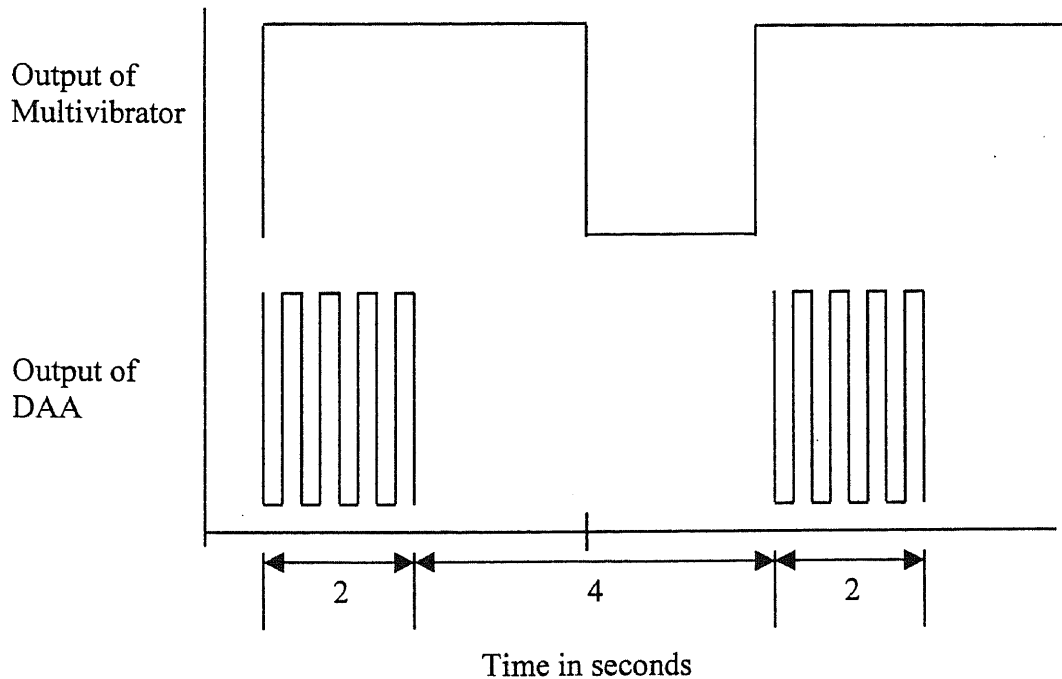
this timing relation. The 74LS191 counter has a positive edge clock. When counter receives a positive-going edge from the multivibrator it will increment at the beginning of each ring.

The counter begins to operate at power-up. Since at power-up the capacitor has no charge the upper plate of the capacitor is virtual zero. The Load input of AND gate C in



**Figure 3.11** Counter circuit used in Ring Detector circuit.

Figure 3.11 is also initialized to zero. Therefore the input on the PL\* pin of the 74LS191 counter will be zero. This will cause the inputs on pins P0, P1, P2 and P3 to be loaded into the counter setting the output of counter (Q0, Q1, Q2 and Q3) to zero. When “ringing” begins it causes the counter to increment. Note the dial and extension outputs of the counter circuit. The extension output is low when the count is 1. This will indicate to the microcontroller that there is a request for access to the Controller. The microcontroller will then seize the telephone line to begin communication.



**Figure 3.12** Timing relation of multivibrator output (top) and ringing cadence at output of DAA (bottom).

There was one problem here that needed to be solved. The extension output of Figure 3.11 should only be active when an extension phone is off-hook. However, it does



become active even in the dial-up mode since the counter will eventually reach a count of 1. If this problem is not solved then we will never get to a count beyond 1 and there will be no distinction between dial-up access and extension phone access. That is, the telephone line will be seized at the count of one. This will cause the ringing signal from the central office to stop, ending the count at one. What makes this problem serious is that for extension access no access code is required and anyone can have access to any device in the home although the user would be using a remote telephone for access. To solve this problem a twelve-second software delay was added. Since only the extension phone can cause the RVLC\* pin to be at logic 0 for 12 seconds then the microcontroller cannot be confused as to which mode of access to provide. In other words, the count of 1, using a remote telephone, will not remain at 1 long enough for the Controller to accept it as a local access. After each access to the Controller the microcontroller resets the counter by setting the Load input on AND gate C to a logic 0.

Because of the timing relation between the multivibrator output and the DAA output, setting the number of dial-up rings was not straightforward. To set the Controller to “ring” 2 times the counter had to be set for a count of 3. To explain the reason for this we must look back at Figure 3.12. Note that the first high output of the multivibrator occurs when the ring signal goes to logic 0 causing the counter to immediately go to a count of 1. The output then goes low for 2 seconds before the next ring. After the first silent period of the ring signal and at the beginning of the second ring the counter will go to a count of 2 because of the second high at the output of the multivibrator. A count of 3 will occur just at the start of the third ring. In fact the microcontroller will respond to this

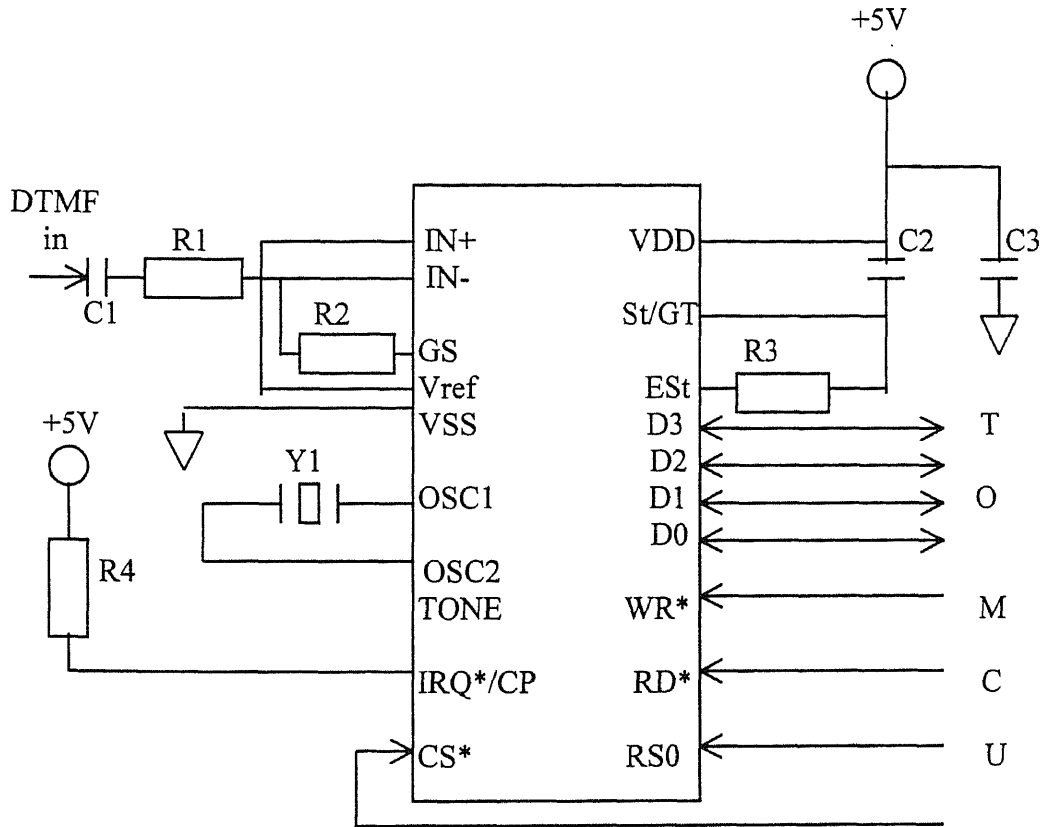
count of 3 so fast that the ring signal will be cut off before it can even make a complete cycle. Therefore only 2 rings actually occur for a count of 3 at the output of the counter.

### 3.4 Tone Detector

The MT8888C Integrated DTMF Transceiver is responsible for detecting and converting DTMF tones. Figure 3.13 shows the Tone Detector circuit diagram used in the Controller. The MT8888C detects a valid dual tone by a steering circuit. Before loading the Receive Data Register (RDR) with the corresponding 4-bit data the steering circuit in conjunction with the C2 and R3 checks the duration of the tone. This is done so that voice tones (which are of shorter duration) are not interpreted as valid tone pairs. When a valid tone pair is received Est goes to logic 1 which in turn drives St/GT to logic 1. Provided that the signal is maintained for the valid period, St/GT will reach the threshold voltage of the steering circuit. This is enough time for the steering logic to register the corresponding code in the register. After the code is registered GT outputs a logic 1 which will remain as long as Est stays at logic 1. There will be a short delay to allow the output latch to stabilize. After this delay the steering output flag goes to logic 1 indicating that a valid tone pair has been received and registered. The condition of a valid tone pair detection can be obtained by reading a logic 0 on bit b3 of the status register. Table 3.6 [1] shows the representation of each bit in the status register. Although the steering circuit rejects signals that are too short to be considered valid, it will tolerate signal interruptions too short to be considered a valid pause [1].

The input signal, from the DAA, enters the MT8888C on IN-. In Figure 3.13 the input is connected in the single-ended mode. Capacitor C1 and resistors R1 and R2

determine the gain of the input amplifier and GS provides the feedback path. Capacitor C1 also provides dc voltage blocking.



**Figure 3.13** Tone Detector Circuit.

The MT8888C is capable of generating sixteen standard DTMF tone pairs with low distortion and high accuracy. All frequencies are derived from crystal Y1 that must be 3.579545 MHz. The sinusoidal waveforms for the individual tones are digitally synthesized using a row and column programmable divider and switched capacitor D/A converters. The standard set for the tolerance of the individual tones in North America is Register. The individual tones generated are referred to as low group (LOW FREQ.) and

**Table 3.6** Status register bit description.

BIT	NAME	STATUS FLAG SET	STATUS FLAG CLEAR
b0	IRQ	Interrupt has occurred . Bit 1 (b1) or bit 2 (b2) is set.	Interrupt is inactive. Cleared after Status Register is read.
b1	TRANSMIT DATA REGISTER (BURST MODE ONLY)	Pause duration has terminated and transmitter is ready for new data.	Cleared after Status Register is read.
b2	RECEIVE DATA REGISTER	Valid data is in the Receive Data Register.	Cleared after Status Register is read.
b3	DELAYED STEERING	Set upon the valid detection of the absence of a DTMF signal.	Cleared upon the detection of a valid DTMF signal.

**Table 3.7** Functional Encode/Decode Table.

LOW FREQ.	HIGH FREQ.	DIGIT	D3	D2	D1	D0
697	1209	1	0	0	0	1
697	1336	2	0	0	1	0
697	1477	3	0	0	1	1
770	1209	4	0	1	0	0
770	1336	5	0	1	0	1
770	1477	6	0	1	1	0
852	1209	7	0	1	1	1
852	1336	8	1	0	0	0
852	1477	9	1	0	0	1
941	1336	0	1	0	1	0
941	1209	*	1	0	1	1
941	1477	#	1	1	0	0
697	1633	A	1	1	0	1
770	1633	B	1	1	1	0
852	1633	C	1	1	1	1
941	1633	D	0	0	0	0

high group (HIGH FREQ.) tones. Typically, the high group to low group amplitude ratio is 2dB to compensate for high group attenuation on long loops.

In certain telephony applications it is required that DTMF signals generated are of a specific duration determined either by the particular application or by any one of the exchange transmitter specification currently existing. Standard DTMF signal timing can be accomplished by making use of the burst mode. The transmitter is capable of issuing symmetric bursts/pauses of predetermined duration. This burst/pause duration is  $51 \text{ ms} \pm 1 \text{ ms}$ , which is a standard interval for auto-dialer and central office applications. After the burst/pause duration expires the appropriate bit is set in the Status Register which indicates that the transmitter is ready for more data. The timing described above is available when the DTMF mode is selected in Control Register A (CRA).

The MT8888C is capable of transmitting single tones from either the low or high group. To accomplish this bit b2 of Control Register B (CRB) must be set to logic 1 and the device must be in DTMF mode.

The MT8888C incorporates an Intel microprocessor interface that is compatible with a 16 MHz 80C51 microcontroller. However, this device was interfaced to the Motorola MC68HC11 in which additional logic had to be use for RD\* and WR\* signaling. The MC68HC11 microcontroller has only one pin for both the read and write signals in which a high indicates a read and a low for write. The MT8888C however has two pins, one for read and the other for write. Hence the use of addition logic. This chip was used due to availability. The DTMF chip for the MC68HC11 is the MT8880C.

The microprocessor interface of the MT8888C provides access to five internal registers. The read-only Receive Data Register contains the decoded output of the last valid DTMF digit received. Data entered into the write-only Transmit Data Register will determine which tone pair will be transmitted. Transceiver control is accomplished with two control registers – CRA and CRB – which have the same address. CRB can be accessed only when a logic 1 is written to bit b3 of CRA. The following control register write will be to CRB. The third control register write will be to CRA again. If CRB needs to be accessed again then a logic 1 must be written to b3 of CRA again. Table 3.8 shows the logic levels on the control pins to access the registers. These control levels must accompany a logic 0 on pin CS\*.

**Table 3.8** Control logic to access registers.

RS0	WR*	RD*	FUNCTION
0	0	1	Write to Transmit Data Register
0	1	0	Read from Receive Data Register
1	0	1	Write to Control Register
1	1	0	Read from Status Register

A software reset must be included at the beginning of the program to initialize the control registers after power-up. The initialization procedure should be executed 100 ms after power-up. The initialization process must follow the sequence below [1]:

	RS0	WR*	RD*	D3	D2	D1	D0
1. Read Status Register	1	1	0	X	X	X	X
2. Write to Control Register	1	0	1	0	0	0	0

3. Write to Control Register	1	0	1	0	0	0	0
4. Write to Control Register	1	0	1	1	0	0	0
5. Write to Control Register	1	0	1	0	0	0	0
6. Read Status Register	1	1	0	X	X	X	X

After this initialization sequence the control registers can be configured according to the users choice of operation. The following sequence was used to configure the control registers of the Controller:

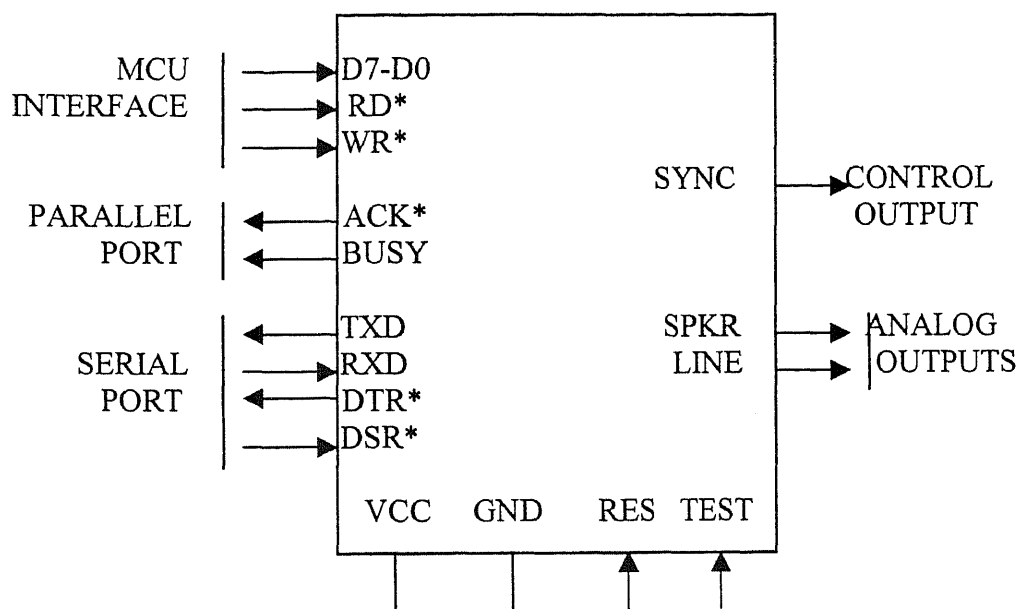
	RS0	WR*	RD*	D3	D2	D1	D0
1. Write to Control Register A (tone out, DTMF, IRQ*, Select CRB)	1	1	0	X	X	X	X
2. Write to Control Register B (burst mode)	1	0	1	0	0	0	0

The tone out, IRQ\* and burst mode were not needed for the Controller since it only operated in the receive mode.

### 3.5 Voice Module

The V8600 Speech Synthesizer, developed by RC Systems was used for the voice prompts and responses of the Controller. It has the capability to convert ASCII text into a high quality male voice. However the module can be programmed for a female voice if the programmer so desires. Figure 3.14 shows the logic symbol of the V8600. It consist

of a  $131,072 \times 8$  ROM which contains the text to speech algorithms. The V8600 supports 524,288 bytes of ROM for storing customization programs such as pre-recorded PCM-encoded speech. An  $8,192 \times 8$  RAM provides storage for the exception dictionary, a 4K FIFO buffer for the DAC (Digital to Analog Converter) and tone generators, and the input text/command buffer. The input buffer and the exception dictionary share approximately 2600 bytes of the RAM.



**Figure 3.14** V8600 logic symbol.

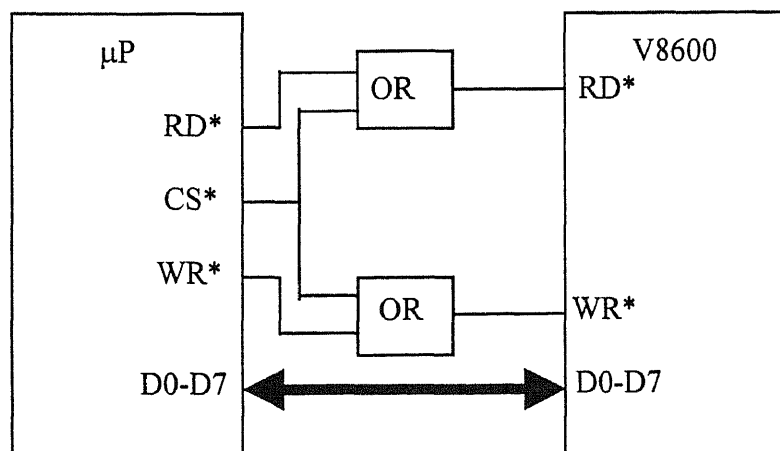
The module supports three types of interfaces namely, microprocessor, parallel and serial. An example of each interface is shown in Figures 3.16, 3.17 and 3.18 respectively. The microprocessor's read and write signals control the data direction between the microprocessor and the V8600. The CS\* signal can be derived from an address decoder. In the microprocessor configuration, the host processor can read the



V8600 status flags. The bit definitions of the status byte are shown in Figure 3.15. When the SYNC bit is set to 1 the synthesizer is either talking or sending out data from the tone generator. It goes to 0 immediately after output ceases. SYNC2 is similar to SYNC but drops to 0 up to 0.5 seconds earlier. The RDY (Ready) is set to 1 when the V8600 is ready to accept data. AF (Almost Full) is set to 1 when less than 300 bytes are available in the text/command input buffer. AE (Almost Empty) is set to 1 when less than 300 bytes are occupied in the text/command input buffer

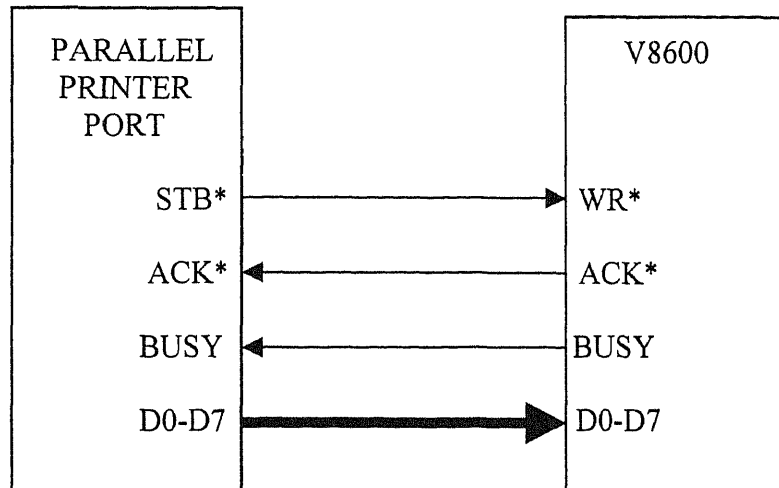
0	SYNC	SYNC2	RDY	AF	AE	0	0
---	------	-------	-----	----	----	---	---

**Figure 3.15** V8600 status flags.

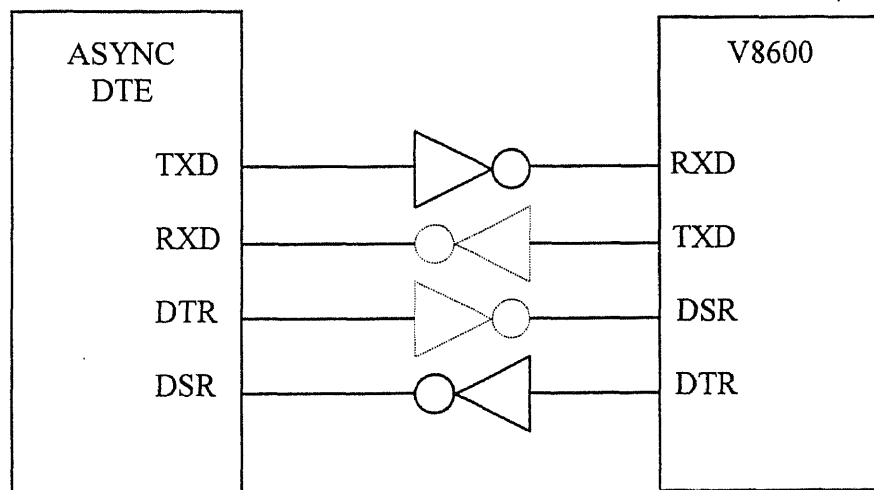


**Figure 3.16** Microprocessor interface example

In the printer port example, the STB\* output from the computer's parallel printer port connects directly to the V8600's WR\* pin. The V8600's ACK\* and busy outputs



**Figure 3.17** Parallel Printer Port interface example.



**Figure 3.18** RS-232 Serial interface example

serve as handshaking signals with the port. In most cases it is not necessary to utilize both. The V8600's asynchronous serial port provides the means to operate with a handshaking signals as they essentially convey the same information. In this configuration, the host computer simply prints the text to be spoken to the V8600 remote computer's

communications port. The port operates with 8 data bits, one or more stop bits and no parity. Baud rate selection is automatic. The V8600 currently does not use the TXD or DSR pins for any purpose, and may be left unconnected. In this configuration, the host computer simply prints the text to be spoken to the V8600. Because the V8600's serial port I/O pins operate at TTL levels, the addition of (at most) two RS-232 line drivers and receivers are necessary

The baud rate of the serial port is determined automatically from the first character received on the RXD pin which is usually CR (0Dh). Since this is done by measuring the duration of the start bit, the first data bit (D0) must be a logic 1 for proper detection of the end of start bit. The first character is then discarded. The baud rate is reset only when the V8600 is reset.

The V8600 has four addressing modes namely Text, Character, Phoneme and PCM. The modes can be changed at any time, even within the same string of text. In the Text mode all text sent to the V8600 is spoken as complete sentences. Punctuation is also taken into consideration by the intonation generation algorithms. The V8600 will not begin translating text until it receives a CR (0Dh) or Null (00h) character -- this ensures that sentence boundaries receive the proper inflection.

The Character mode causes the V8600 to translate input text on a character-by-character basis. In other words text are spelt instead of spoken as words. The V8600 does not wait for a CR/Null in this mode.

The Phoneme mode is useful for creating customized speech, when the normal text-to-speech (Text) mode is inappropriate for producing the desired voice effect. Phoneme mode is used when it is important for the correct stress and emphasis be placed

on certain words in a phrase. This detailed modification is not possible with text mode since changes are only allowed at word boundaries whereas phoneme mode allows changes within words. Table 3.9 lists the phonemes that can be produced by the V8600 and Table 3.10 lists the attribute tokens. For example in text mode a sentence, in C language, would be sent to the V8600 as follows:

```
printf("How dare you speak to me like that way!");
```

In phoneme mode the same sentence could sent as shown below:

```
printf("70H AW -/D>/EH R +<\\YY UW S P\\IY K T UW \\MIY DH AE T -W EY  
.+/");
```

**Table 3.9** Synthesizer Phonemes

Phoneme Symbol	Example Word	Phoneme Symbol	Example Word	Phoneme Symbol	Example Word
AA	father	H	hire	SH	Dish
AE	bat	IH	sit	T	Tip
AH	cut	IX	relative	TH	Thick
AO	lawn	IY	meet	TX	Mistake
AW	cow	JH	jet	UH	Pull
AX	about	K	cute	UW	Tool
AY	kite	KX	ski	V	Give
B	bird	L	long	W	Went
CH	cheese	M	mug	WH	When
D	dare	N	new	YY	You
DH	either	NX	rung	Z	Zero
DX	computer	OW	tone	ZH	Leisure
EH	set	OY	boy	space	variable
ER	were	P	past	,	pause
EY	bake	PX	spot	.	medium
F	fact	R	ring		pause
G	give	S	some		long pause

**Table 3.10** Phoneme Attribute Tokens.

Symbol	Function
nn	Set pitch to 'nn' (0 – 99)
/	Increase pitch 10 steps
\	Decrease pitch 10 steps
+	Increase speed 1 step
-	Decrease speed 1 step
>	Increase volume 1 step
<	Decrease volume 1 step

In PCM mode data sent to the V8600 is written to the digital to analog converter FIFO buffer. The results in a very high data rate, and provides the capability to produce the highest quality speech. This mode also provides sound effects that are not possible in the text mode.

The V8600 interprets a list of commands that are used to change the synthesizer's attributes, such as volume or pitch. A list of these commands is shown in Table 3.11.

The output of the V8600 (SPKR pin) was connected to the DAA so that the audio could travel along the telephone line to the user. A resistor was connected between the V8600 and the DAA as recommended by the manufacturer. A 0.1 $\mu$ F capacitor blocked the 2.5V on the VR pin from reaching the V8600. A logic circuit connected to WR\* and RD\* provides selection of the Speech Synthesizer only when its address was specified on the address bus. This logic was necessary since the V8600 does not have a chip select pin [18].

**Table 3.11** V8600 command summary.

<b>Command</b>	<b>Function</b>	<b>Range n</b>	<b>Default</b>
NB	Punctuation level	0 - 7	6
NC	Character mode/delay	0 - 31	
D	Phoneme mode	-	
E	Enable intonation	-	
NF	Format frequency	0 - 9	
J	Tone generator	-	
L	Load exception dictionary	-	
M	Disable intonation	-	
NP	Pitch	0 - 99	
R	Clear	-	
NS	Speed	0 - 9	
NT	Text mode/delay	0 - 15	
U	Enable exception dictionary	-	
NV	Volume	0 - 9	
NX	Tone	0 - 1	
NY	Timeout delay	0 - 15	
Z	Zap command	-	
@	Reinitialize	-	
n#	PCM mode	0 - 99	

## CHAPTER 4

### SOFTWARE IMPLEMENTATION

This chapter contains all the relevant information pertaining to the development of the software for the Controller. This includes the development software configuration for the banked memory mode which was used to allow the 16-bit address lines of MC68HC11 microcontroller access 128 bytes of memory. The structure of the device application program that allows the proper operation of the Controller in relation to CEBus internal function is described in section 4.2. Also included is the configuration of the CEBench software to allow correct interfacing of software and hardware for communication over the power line.

#### 4.1 Bank Memory Implementation

The file `cstartup.s07` contains the routine to configure the hardware after a power-up or manual reset. The user to comply with the current hardware design can modify this file. One important area of configuration is to allow memory to operate in banked memory mode. As mentioned earlier this allow the MC68HC11 microcontroller which has 16 address lines (with address space of 64Kbytes) to access 128Kbytes of memory. When the banked memory compiler option is selected the designer must assign a memory location to store the bank number. This assignment should be made in both the `cstartup.s07` and `i09.s07` configuration files. The location 1060H was used for this memory location as shown below in the `cstartup.s07` file:

```
IOPORT    EQU    $1060
```

## CLR IOPORT

The foregoing instruction switches the bank to bank 0. The 109.s07 file was modified to include the following assembly code :

```

IOPORT    EQU    $1060
          LDAA   IOPORT
          PSHA
          .
          .
          .
          STAA   IOPORT
          PULA
          STAA   IOPORT

```

The second instruction loads the current bank number into the accumulator. The second stores this number on the stack. After obtaining the new bank number from the X register it is stored in location 1060H (third to last instruction). After the program, in the selected bank, has executed the old bank number is pull of the stack then stored in location 1060H (second to last and last instructions).

The Altera PLD is responsible for putting the bank number onto the memory address lines. Figure 4.1 shows a copy of the bank-switching program used in the Altera. The Altera chip outputs the bank number to the memory chip when the instruction

```
bankreg[].clk = !(bnk & !rw);
```

is executed. The bank number to the memory chip is determined by the following “if” statement in the Altera PLD:



```
IF (addr[14]) THEN
    bank[2..0] = 7;
ELSE IF (!addr[14]) THEN
    bank[2..0] = bankreg[2..0];

TITLE "68HC11 Banked Memory";
Subdesign banked
(
    data[7..0]          :bidir;
    addr[15..], iocs, rw :input;
    bank[2..0]          :output;
)
Variable
    bankreg[2..0]      :dff;
    set, baddr, bnum[7..0] :node;
BEGIN
    baddr = (addr[] == H"1060" & !iocs);
    set = baddr & rw;
    data0 = tri(bnum0, set);
```

**Figure 4.1** Altera bank memory implementation program (part 1).

```
data1 = tri(bnum1, set);  
data2 = tri(bnum2, set);  
data3 = tri(bnum3, set);  
data4 = tri(bnum4, set);  
data5 = tri(bnum5, set);  
data6 = tri(bnum6, set);  
data7 = tri(bnum7, set);  
bankreg[2..0].d data[2..0];  
bankreg[].clk = !(baddr & !rw);  
  
IF (addr[14]) THEN  
    Bank[2..0] = 7;  
ELSEIF (!addr[14]) THEN  
    Bank[2..0] = bankreg[2..0].q;  
END IF;  
  
IF (baddr & rw) THEN  
    bnum[2..0] = bankreg[2..0];  
    bnum[7..3] = GND;  
END IF;
```

**Figure 4.1** Altera bank memory implementation program (part 2).

When address line 14 is low the content of the bank register determines which bank is selected. If the bank register clock is not activated then the accessed location in memory may be the non-banked location (bank 7) or the current banked location. The non-banked location is always selected when address line 14 is high. The address range of each bank is shown in Figure 4.2 and the memory map showing the addresses used for banked and non-banked memory for the MC68HC11 is shown in Figure 4.3. Figure 4.3 also shows

00000H	BANK 0
03FFFH	
04000H	BANK 1
07FFFH	
08000H	BANK 2
0BFFFH	
0C000H	BANK 3
0FFFFH	
10000H	BANK 4
13FFFH	
14000H	BANK 5
17FFFH	
18000H	BANK 6
1BFFFH	
1C000H	NON-BANKED
1FFFFH	

**Figure 4.2** Address range of each bank in memory chip.

0000H	INTERNAL MCU RAM
03FFH	
0400H	EXTERNAL RAM
0FFFH	
1000H	INTERNAL REGISTERS
105FH	
1060H	BANK MEMORY CONTROL REGISTER
1061H	NOT USED
1062H	SPEECH SYNTHESIZER
1063H	COUNTER RESET
1064H	LCD OPERATION REGISTER
1065H	LCD DATA REGISTER
1066H	DTMF DATA REGISTER
1067H	DTMF CONTROL & STATUS REGISTERS
1068H	DAA

**Figure 4.3** Memory map of Controller (part 1).

the total memory map of the Controller. More will be said about this memory map in the next section. Each bank contains 16kbytes. The total number of bytes on the memory

1069H	NOT USED
17FFH	
1800H	EXTERNAL RAM
7FFFH	
8000H	BANKED MEMORY
BFFFH	
C000H	NON-BANKED MEMORY
EDFFH	
EE00H	INTERNAL EEPROM
EFFFH	
F000H	NON-BANKED MEMORY
FFFFH	

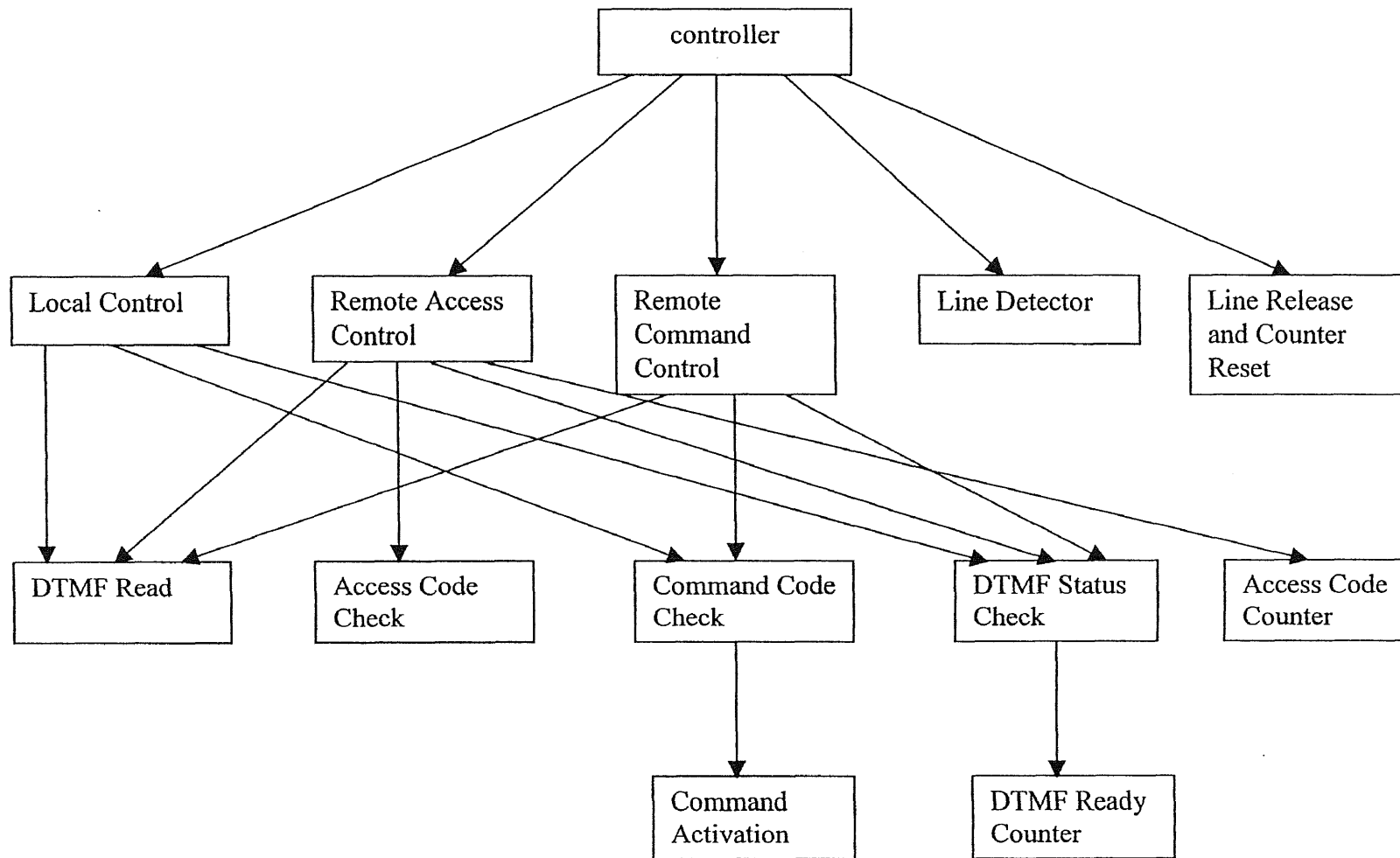
**Figure 4.3** Memory map of Controller (part 2).

chip used for the Controller is 128kbytes. The address range 8000H to BFFFH of the MC68HC11 is used to access the banked portion of memory. Switching to the different banks is done as outlined above. The non-banked area of memory is addressed from C000H to EDFF and F000H to FFFFH of the microcontroller. These addresses are directly mapped to non-banked memory.

## 4.2 Software Development

The software program was designed in a hierarchical arrangement to facilitate the limited timing for the program and for ease of debugging. This hierarchical arrangement is shown in Figure 4.4. Lower level modules are controlled by the upper layer modules. However, data may flow from the lower layers to the upper layers as well as from the upper layers to lower layers. Table 4.1 shows the data transfer between the control and controlled modules. So as not to be confused, the controller in the first column of the Table refers to one of the software module and not the entire unit. (We used a capital “c” when referring to the unit and a lower case “c” for the software module.) This first module is called the controller since it determines when the other modules execute. The column list the module that initiates the execution of other modules. The second column lists the data that is submitted to the called module and indicates the task to be performed. The third column lists the called modules. The last column shows the data that the called module returns to the calling module. This data is the result of the module’s execution.

Figure 4.3 shows the memory map as seen by the MC68HC11 microcontroller. The MC68HC11 only has 16 address lines and therefore can only address 64kbytes (0000H to FFFFH) of memory. The Controller required much more memory than this to perform all its functions. The memory was therefore increased to 128kbytes (00000H to 1FFFF) using banked addressing mode. The microcontroller’s address range 8000H to BFFF was used to address each bank of memory. These addresses provided the offset in each memory bank while the bank selection was done by the Altera PLD, as was outline earlier. The flowcharts for the device-level programs are in Appendix B.



**Figure 4.4** Hierarchical arrangement of software program.

**Table 4.1** Data transfer between modules in hierarchical program

Control Module	Data Sent	Controlled Module	Data Returned
controller	Void	Local Control	0 = Not finished 1 = Finished
controller	Void	Remote Access Control	0 = Not finished 1 = Finished
controller	Void	Remote Command Control	0 = Not finished 1 = Finished
controller	Void	Line Detector	0 = Line not active 1 = Local access request 2 = Remote access request
controller	Void	Line Release and Counter Reset	Void
Local Control	Void	DTMF Read	Digit
Local Control	Command code	Command Code Check	0 = Incorrect 1 = Correct
Local Control	Void	DTMF Status Check	0 = Not ready 1 = Ready 2 = Not-ready counter at max.
Remote Access Control	Void	DTMF Read	Digit
Remote Access Control	Access code	Access Code Check	0 = Incorrect 1 = Correct
Remote Access Control	Void	DTMF Status Check	0 = Not ready 1 = Ready 2 = Not-ready counter at max.
Remote Access Control	0 = Reset counter 1 = Increment counter	Access Code Counter	0 = Not maximum count 1 = Maximum count
Command Code Check	Command code	Command Activation	0 = Code out of range for addressed device 1 = Code accepted
DTMF Status Check	0 = Reset counter 1 = Increment counter	DTMF Ready Counter	0 = Not maximum count 1 = Maximum count



### 4.3 CEBus Message Transfer using CEBench

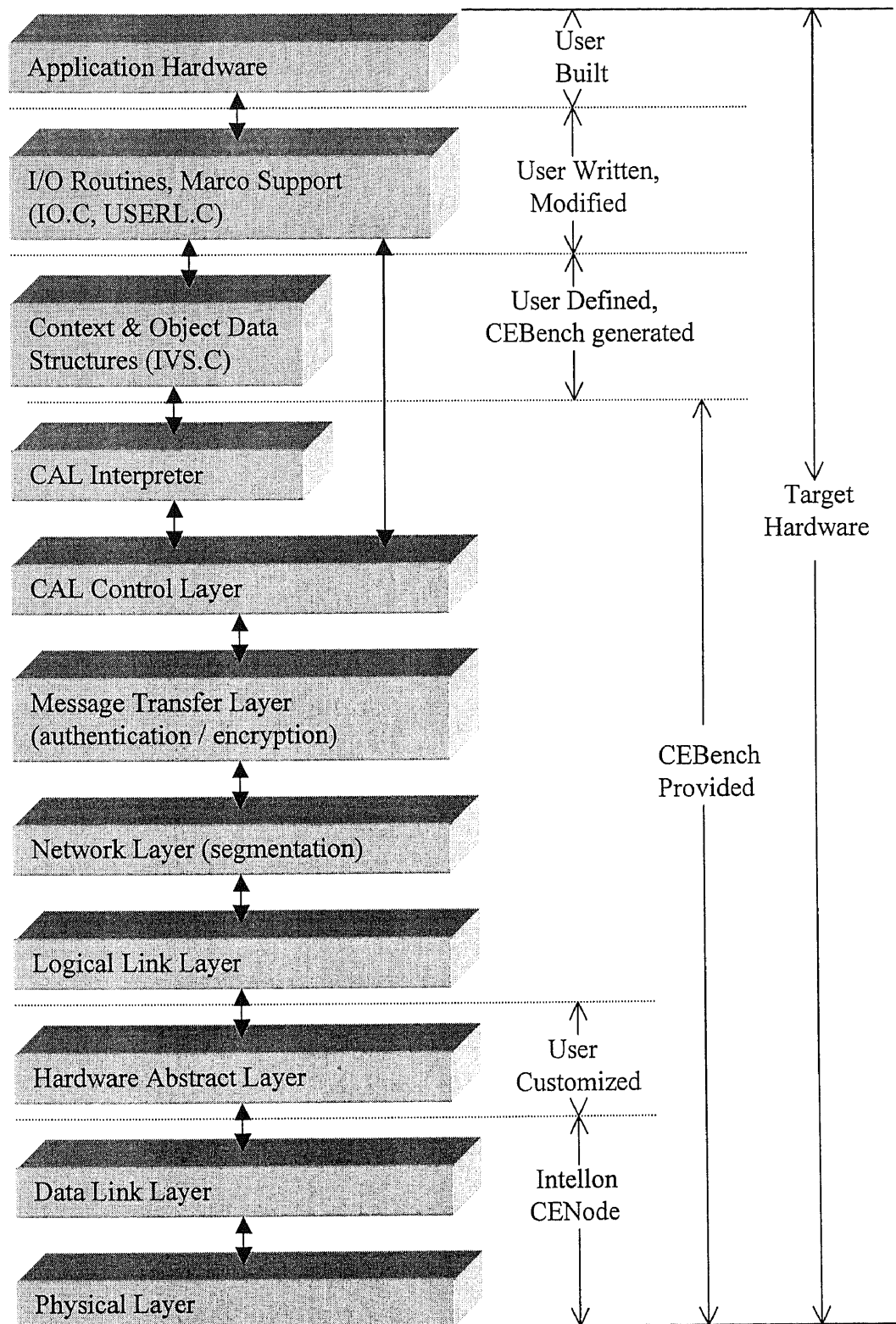
CEBus message transfers were made possible by configuring the CEBench software to suit the Controller's application. The CEBench software consists of Contexts, Objects and Instance Variables (IVs) that may be written to or read from. They provide the interface, in conjunction with the IO.C source file, between the user written program and the power line hardware. The IO.C was modified to transfer data to and from specified IVs so that CEBus messages could be sent at the appropriate times. Another file that was modified was the HC11HAL.C, used to configure the MC68HC11. Figure 4.5 shows the components of CEBench used to construct a CEBus device. It also specifies the files to be modified by the user and the file that must be developed by the user [2].

#### 4.3.1 Layer Definitions

This section defines the layers relevant to the development of a CEBus device from a developer's point of view.

The CAL Control Layer consist of two parts, namely the definition of a data structure that models a products operation and a command syntax that defines the operation of a set of methods on the data structure. The CAL is actually responsible for what products say to each other.

The CAL interpreter is responsible for CEBus message origination and the receiving and parsing of CEBus messages. It is an element of the application layer. It provides services to the application including resource allocation and control functions.



**Figure 4.5** CEBench components of CEBus device

The IVS.C file defines the CEBus Contexts, Objects and IVs, their initialization, and their associations to the IO.C source file. It consists of tables generated by CEBench from user specifications. This file must not be modified since the target libraries are written to interface to this file as it is generated by CEBench. Modifying this file may result in incorrect or non-operation of the target executable file.

The IO.C is a template file to be modified by the user. It provides the connection between the Contexts, Objects IVs and the associated target system hardware. This association is implemented by using a series of case statements in which each case number corresponds to a number placed in the I/O column of the Context. When the CEBus execution cycle reaches this I/O number the of the Context the corresponding series of code will execute.

The USERL.C file allows the user to implement customized CEBus functionality. It provides user access to CAL control indications of CEBus functions such as macros, and result messages and also allows user access to the message transfer layer for sending CAL messages [2].

Another important file that is not included in Figure 4.5 is MAIN.C. This file implements the overall target program flow. It is responsible for calling the subroutines to initialize the CON Control, Message Transfer, Network, Logical Link and the Hardware Abstract Layers. After the initialization of these layers it then calls the CEBus executive entry point – CEBus\_Proc(). This in turn initiates the execution of the CON Control, Message Transfer and Network Layers. Since this entry point is in a while loop these layers are repeatedly executed.

The user developed program is also called from within the while loop in MAIN.C. However the user program must not execute for more than 1msec. Violating this time constraint will result in the breakdown of the communication with the CENode. Using interrupts in the program may also cause problems. Since the Intellon CENode is single buffered and subject to the stringent timing requirements of the CEBus Data Link Layer, the Logical Link Layer interface software disables interrupts during CENode handshaking. This may result in the user interrupts being delayed for up to 2msec or more depending upon CEBus traffic to the device.

#### **4.3.2 CEBench Modifications**

To allow the Controller to operate in a CEBus environment we had to select the appropriate Context, Objects and IVs. After making these selections we then modified them to allow the Controller to function as designed.

Figure 4.6 shows the Contexts, Objects and IVs used in the Controller. Three Contexts are used to perform the function for a light switch and a heater. The Controller address is set to 0003:0001 as shown in IV 'h' and 'a' in the Universal Context. To perform the operation for a light switch, we used the Light Context and three Binary Sensor Objects. Object 06 was set up for automatic data transfer. The automatic data transfer is initiated by changing IV 'C'. This condition for automatic transfer is set by the 'R' IV (43 ed 00) which tells the Controller to send the data whenever the 'C' IV changes. The destination of the data is set by the IVs 'H' and 'A'. 'A' contains the system address (0001) and the MAC address (0002) of the destination device. The IV 'H' contains the Context ID (21), Object (06) and IV (43 the ASCII for C) in the destination

CEBus Remote Access Controller										
#	CONTEXT									ID
A0	Universal () : univ.cxl									00
#	OBJECT									CLASS
01	Node Control (Device Control) : nodectrl.cob									01
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
s	serial_#	ROM	R	String	21	3	3	GA1198		
n	manuf_name	ROM	R	String	21	3	3	GA CEBus Solutions		
m	manuf_model	ROM	R	String	21	3	3	CON997		
c	product_class	ROM	R	String	21	3	3	UNLISTED		
h	system_addr	NVM	R/W	Data	1x2	3	3	00 03		
a	mac_addr	NVM	R/W	Data	1x2	3	3	00 01		
o	context_list	ROM	R	Data	4x2	3	3	a0 00 a0 21 a0 74 a1 74		
f	configured	RAM	R/W	Boolean	1	3	3	True		
i	setup	RAM	R/W	Integer	2	3	3	0		
u	user_feedback	RAM	R/W	Integer	2	3	3	0		
02	Context Control (Context Control) : cntxctrl.cob									02
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
o	object_list	ROM	R	Data	3x2	3	3	01 01 02 02 16 03		
03	Data Memory (Event Manager) : datamem.cob									16
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
C	current_index	RAM	R/W	Integer	2	3	3	0		
I	memory_block	RAM	R/W	Data	1x25	3	3			
CEBus Remote Access Controller										
#	CONTEXT									ID
A0	Light (Switches) : light.cxl									21
#	OBJECT									CLASS
01	Context Control (Context Control) : cntxctrl.cob									02
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
o	object_list	ROM	R	Data	4x2	3	3	02 01 06 06 06 07 06 08		
06	Binary Sensor (Light Switch) : bsensor.cob									06
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
C	current_state	RAM	R	Boolean	1	3	3	False		
R	report_condition	NVM	R/W	Data	1x4	3	3	43 ed 00		
H	report_header	NVM	R/W	Data	1x6	3	3	21 06 45 43 f5		
A	dest_address	NVM	R/W	Data	1x4	3	3	00 02 00 01		
P	previous_value	RAM	R	Boolean	1	3	3	False		
07	Binary Sensor (Light Intermediate Switch) : bsensor.cob									I/O # (2) 06
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
C	current_state	RAM	R	Boolean	1	3	3	False		
08	Binary Sensor (Ack) : bsensor.cob									I/O # (3) 06
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
C	current_state	RAM	R/W	Boolean	1	3	3	False		

Figure 4.6 Contexts, Objects and IVs used for the Controller (part 1).

CEBus Remote Access Controller										
#	CONTEXT									ID
A1	Refrigerator/Freezer (Heater) : refrigfr.cxl									74
#	OBJECT									CLASS
01	Context Control (Context Control) : cntxctrl.cob									02
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
o	object_list	ROM	R	Data	7x2	3	3	02 01,07 02,05 05,06 07,08 08,06 09,08 0a		
02	Analog Control (Temperature Setting) : analctrl.cob									07
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
N	min_value	ROM	R	Integer	2	3	3	65		
M	max_value	ROM	R	Integer	2	3	3	85		
D	default_value	ROM	R	Integer	2	3	3	72		
C	current_value	RAM	R/W	Integer	2	3	3	0		
R	report_condition	NVM	R/W	Data	1x4	3	3	43 ed 31		
H	report_header	NVM	R/W	Data	1x6	3	3	a1 74 02 45 43 f5		
A	dest_address	NVM	R/W	Data	1x4	3	3	00 02 00 01		
P	previous_value	RAM	R	Integer	2	3	3	0		
05	Binary Switch (Heater Switch) : bswitch.cob									05
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
C	current_position	RAM	R/W	Boolean	1	3	3	False		
R	report_condition	NVM	R/W	Data	1x4	3	3	43 ed 00		
H	report_header	NVM	R/W	Data	1x6	3	3	a1 74 05 45 43 f5		
A	dest_address	NVM	R/W	Data	1x4	3	3	00 02 00 01		
P	previous_value	RAM	R	Boolean	1	3	3	False		
07	Binary Sensor (Heater Intermediate Switch) : bsensor.cob									I/O # (4) 06
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
C	current_state	RAM	R	Boolean	1	3	3	False		
08	Analog Sensor (Temperature Storage) : analsens.cob									I/O # (6) 08
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
C	current_value	RAM	R	Integer	2	3	3	0		
09	Binary Sensor (Switch Ack) : bsensor.cob									I/O # (5) 06
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
C	current_state	RAM	R/W	Boolean	1	3	3	False		
0A	Analog Sensor (Temperature Ack) : analsens.cob									I/O # (7) 08
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
C	current_value	RAM	R/W	Integer	2	3	3	0		

Figure 4.6 Context, Objects and IVs used for the Controller (part 2).

device to modify. (If there were more than one Contexts with the same ID in the destination device then 'H' would have to include the Context sequence number. In the Controller the sequence number for the Light Context is A0. A second Light Context would have sequence number A1.) The destination device would read this IV and perform the functions on its hardware (for example turn off/on the light) accordingly. When the Controller changes the 'C' IV to true it indicates to the controlled device that the light must be turned on. A false indicates to turn the light off.

The Object with number 07 is used for temporary storage of the state that will eventually be transferred to the 'C' IV in Object 06. The reason for this is that we did not want the user written device application program to affect the timing of data transfer. We preferred that the data be transferred when the CEBus\_Proc() executes.

The Object number 08 is used for acknowledgement. When the Controller send commands to a device it expects an acknowledgement from the device. This acknowledgement alters the 'C' IV of Object 08. The Controller reads this IV and determines whether it has changed from the last time it was read. If it has changed the Controller will send a response to the user indicating that the task is complete. If it has not changed the response to the user will indicate that the device has not responded. At this point the user may try again. If the response is still negative then the controlled device may be disconnected or defective.

The Context with ID 74 and sequence number A1 implements a device with both binary and analog adjustments. For this Controller it represents a heater control. The Objects 05, 07 and 09 operates just like the Objects 06, 07 and 08 respectively in the Light Context. This implement turning the heater on and off. The other Object allows

analog adjustments of the heater. Object 02 is set up to allow automatic transfer of data whenever the IV 'C' changes by 1. This report condition is set in the 'R' IV. The number 43 is the ASCII code for 'C', ed means change by and 31 is the ASCII code for 1. What the report condition says is transmit data when the 'C' IV changes by 1. Note that in the report header IV Context sequence number (a1) is used since another Context with ID 74 also exist in the Controller. (See Appendix D for all Contexts used.) The '74' is the destination device Context ID, '05' the destination Object, 45 the method (means 'to set'), 43 ASCII for 'C' and f5 is the delimiter. The same destination address is specified in IV 'A' as for the Light Context. Actually a different address should be specified. The same address was used for test purposes only. In testing the Controller the computer was used for all controlled devices hence, the same address. The acknowledgement for the analog adjustment is received by object 0A that will be set to the temperature value just sent to the controlled device. In other words, the temperature value sent by the Controller will be the same value returned by the controlled device as an acknowledgement. The Object 08 contains the temporary storage for the temperature value that will be read from within CEBus\_Proc() so that it will be sent to the controlled device at the appropriate time.



## CHAPTER 5

### USER INTERFACE

This chapter presents the details of how the user would use the Controller to control a CEBus device. The Controller operates in two modes namely dial-up and extension (or remote and local). We will first look at the dial-up mode and then describe the procedure that would not be included in the extension mode.

The Controller must be installed at the location where CEBus devices are connected to the power line. To install the Controller just plug the RJ11 plug into a telephone jack. Plug the power line cord into the jack on the Controller and the other end of the cord into the power line receptacle. This is all that is required for installation. There are no settings to make. The Controller is now ready to go.

The first step in operating the Controller (in the dial-up) is that the user lifts the handset and dial the location where a device needs to be accessed and where the Controller is installed. When the Central Office sets up the connection it will send the ringing signal to the Controller. The Controller is able to detect this ringing signal and activate the appropriate circuitry. There is a switch on the Controller that allows the user to be able to adjust the number of rings the Controller will allow before activating the circuitry. It may be set to 2 or 4 rings. The Controller however, will not produce an audible ring. It only needs to detect the ringing signal.

When the set number of rings have occurred the Controller will send a voice prompt to the user. The prompt will be:

“PLEASE ENTER ACCESS CODE.”

The user will then respond by entering a four-digit access code. If the access code is incorrect the Controller will respond with:

“ENTER ACCESS CODE AGAIN.”

Only two errors can be made. On the third error the Controller will respond with:

“NO MORE TRIES.”

After this response the Controller will hang-up.

If the user enters the correct access code the Controller will then give another prompt. This prompt asks the user to enter the command code. The prompts will be as follows:

“PLEASE ENTER COMMAND CODE.”

The command code is also a four-digit code. The first two digits is the device address and the last two digits is the actual command for the device. For example, if the user enters the command code 0270 from the telephone keypad, 02 would be the device address. This device would then be set to 70. For a heating device this would mean to set the device to 70°F. However, before this device can be set it must first be turned on. The command to turn it on would be 02#X. The “#” symbol means to turn on the device. The ‘X’ is just a filler and can be any digit. To turn off the device the command would be 02\*X. The ‘\*’ symbol means that the device is to be turned off. The ‘X’ is a filler as in the ‘on’ command.

There are a number of voice responses that can be generated after issuing the command code. If the command code is invalid the response would be:

“INVALID COMMAND CODE.”

A list of the invalid conditions for a command code is shown in Table 5.1. The Table also states when a command is valid.

If the user enters the command to turn on a device when it is already on the Controller will respond with:

“DEVICE IS ALREADY ON.”

If the user enters the command to turn off a device when the device is already off the response would be:

“DEVICE IS ALREADY OFF.”

**Table 5.1** Valid and Invalid conditions for a command code.

ADDRESS PART	COMMAND PART	COMMAND STATE
There are no devices with this address.	Valid or invalid command.	Invalid
A device does have this address.	The command does not apply to this device.	Invalid
A device does have this address.	The command value is out of range for this device.	Invalid
A device does have this address.	The command does not apply to any device.	Invalid
A device does have this address.	Valid command for device at this address.	Valid

Say that the user wants to turn on a device but the device is not plugged in. The Controller will also respond to this condition. The response to the user will be:

“NO RESPONSE FROM DEVICE.”

The Controller is designed to receive acknowledgements from the devices with which it communicates. This acknowledgement is different from the one received by the Data Link Layer (DLL). These acknowledgements are set up in the Contexts, Objects and IV's. The controlled device must be set to automatically generate this acknowledgment. When the Controller receives a correct command code and the device to be controlled is plugged into the power line the user will hear the following response after receiving the acknowledgement:

“TASK COMPLETED.”

After the user hears this command the user hangs up. Figure 5.1 shows a flowchart that summarizes the interaction between the Controller and the User.

In the extension mode the Controller does not ask for an access code. The only prompt is for the user to enter the command code. The rest of the operation follows exactly as the dial up mode.

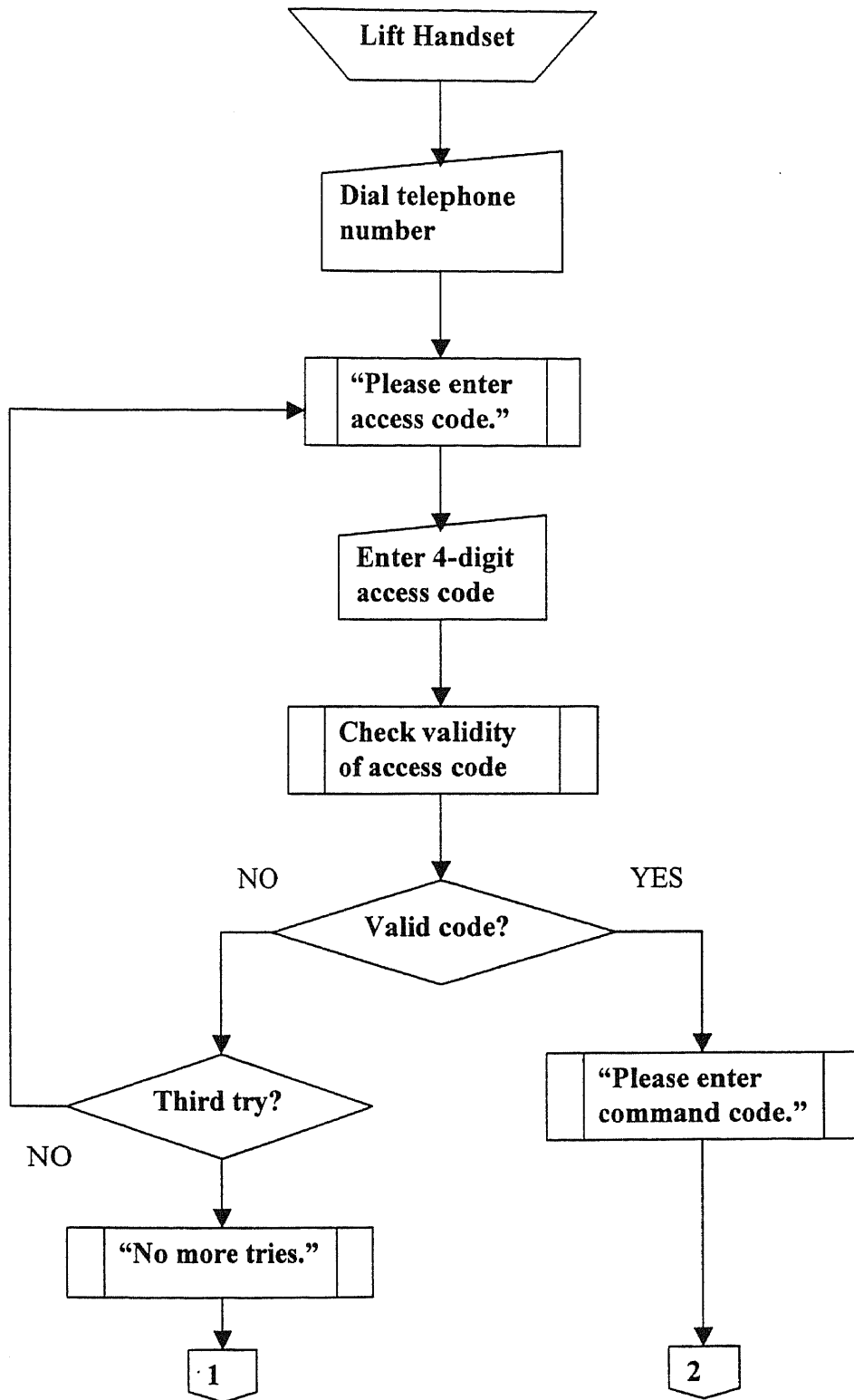


Figure 5.1 Operation flowchart of Controller (part 1).

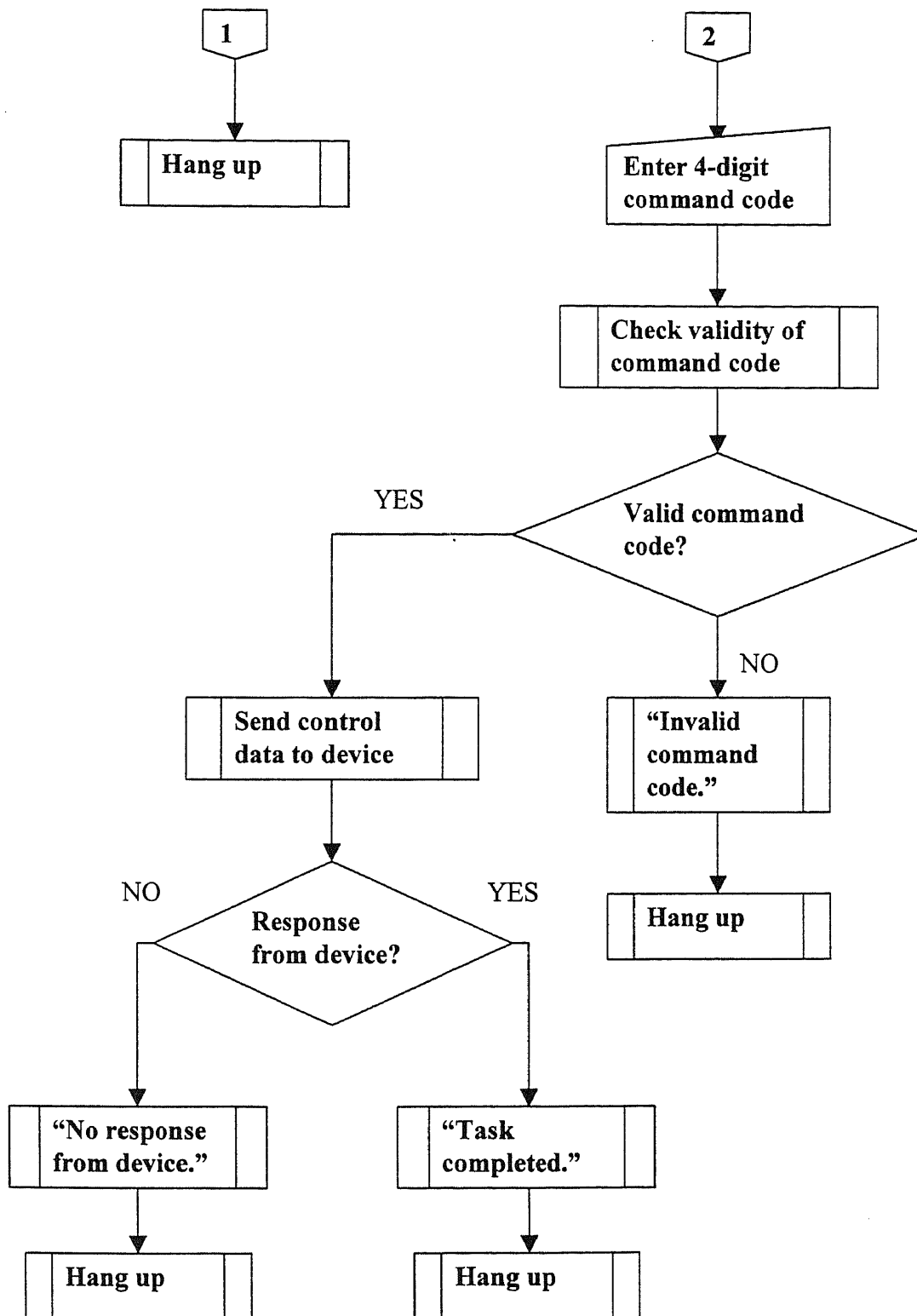


Figure 5.1 Operation flowchart of Controller (part 2).

## CHAPTER 6

### CONCLUSION AND FUTURE IMPROVEMENTS

The design and implementation of the controller has been completed. The five stages were first using a high-level block diagram proceeding to the component level design and implementation. Also the software proceeded from any high-level flowcharts to the writing of many lines of code. It is able to accept request from either an extension telephone or a remote telephone in the dial-up mode. In the dial-up mode the user must enter a four-digit access code followed by a four-digit command code. In the extension mode no access code is required. Since the first two digits of the command code make up the device address, the controller can actually handle 100 devices (00 to 99).

The controller analyzes the command code to determine the task or message to send to the controlled device. When the task is completed the user will receive an intelligible synthesized voice acknowledgment from the controller. The voice response also provides the user with the necessary feedback while the user interacts with the Controller.

There are several changes that can be done to the controller to improve its capability. The following paragraphs outline some of these improvements.

The Controller could be allowed to automatically acquire all relevant information about a new CEBus device connected to the power line. After acquiring this information it should then put the information in a table so that it will know how to communicate with such device. When a new device is recognized a user address should be assigned and

displayed to the user. The user knowing this address will be able to use it when the device needs to be accessed via the Controller.

Currently the Controller allows a user three chances to enter the correct access code. For the commands, only one try is allowed. That is, whether the command is correct or incorrect the user cannot issue another command. If the command is correct the Controller will perform the respective task. If the command is incorrect the Controller will let the user know that the command is incorrect but will not allow the user to issue another command. Therefore the user will have to call again. It will require some modification to the software to allow multiple commands in a single Controller access. In implementing this change a command could be used to tell the Controller that no more commands will follow; or a timer could be used to hang-up the Controller when no commands are received after a certain time elapses.

Another future improve would be to allow the user to change the access code at any time. This will improve the security of the system since the possibility exists for unwanted users to access the Controller and operate a CEBus device in the system.

The Controller could be improved so that it can respond to emergence situations. In this case the Controller would receive packets from devices such a smoke detector or a burglar alarm. The Controller would then dial the user and issue a voice response that indicates the emergence situation. In implementing this feature the user should also be allowed to remotely change the telephone number so that the emergence situation can still be receive when the user changes location.

Currently, if someone dials the location where the Controller is installed, the Controller will respond, requesting the access code. This is unacceptable when the

purpose of the call is to have a conversation with someone in the house. To get rid of this annoyance, the Controller could be improved so that it does not respond unless an initial code is entered. This could be the access code. Therefore the Controller would not respond with a prompt for the access code but that the code will be entered first then, if the code is correct, prompt for the command code.



## APPENDIX A

### CONTROLLER APPLICATION SOURCE CODE

#### appl1.c

```
/*
Written by Gerald Aska
*/
```

#### Purpose :

To monitor and determine the subroutines to be executed. These subroutines are for local and remote control, line monitoring, initialization, and reset/hang-up. Called from main.c.

```
*/
```

```
//The Controller#include "appl1.h"#include
"appl2.h"#include "appl3.h"#include "appl4.h"
#include "voice.h"
#include "lcd.h"
#include <stdio.h>
```

```
void cntl(void)
{
static long int line_count;
static int select;
static int vcom;
int line_d, lactive;

switch(select)
{
case 0:
line_d = l_detect();
if(line_d == 0)
select = 0;
else if(line_d == 1)
select = 1;
else
select = 2;
break;

case 1:
if(local_control())
select = 4;
else
```

```
        select = 1;
        break;

case 2:
    if(remote_acc_control())
    {
        dtmf_init();
        select = 3;
    }
    else
    {
        dtmf_init();
        select = 2;
    }
    break;

case 3:
    if(remote_com_control())
    {
        select = 4;
        dtmf_init();
        vcom = 0;
    }
    else
    {
        dtmf_init();
        select = 3;
    }
    break;

case 4:
    line_count++;
    if(line_count == 2000)
    {
        select = 0;
        line_count = 0;
        l_init_nc();
    }
    else
        select = 4;
    break;
}
}
```

**appl1.h**

```
//The Controller  
void cntl(void);
```

**appl2.c**

```

/*****
Written by Gerald Aska

```

**Purpose:**

Initialize Controller, monitor telephone line for local or remote request, control local and remote operation.

```

*****/

```

```

#include "voice.h"#include "lcd.h"#include
"appl2.h"#include "appl3.h"#include "appl4.h"#include
<stdio.h>#pragma language=extended#include
<io6811f1.h>static int rvalid;static int rvlc;void
l_init(void)

```

```

{
    LIC = 0; //LIC disable
    COUNTER = 1; //reset counter
}

```

```

void l_init_nc(void)

```

```

{
    LIC = 0; //LIC disable
    COUNTER = 1; //reset counter
    rvlc = 1;
}

```

```

int l_detect(void)

```

```

{
char rdetect, xdetect;
static long int xcount;
int select;

```

```

    xdetect = PORTD;
    xdetect &= 0x08;
    rdetect = PORTD;
    rdetect &= 0x10;

```

```

    if(xdetect != 0x00 && rdetect == 0x10)
    {
        xcount = 0;
        select = 2;
    }

```

```

        LIC = 1; //line enable
        v_acc_code();
    }
else if (xdetect == 0x00 && rdetect != 0x10)
{
    if(rvlc)
    {
        rvlc = 0;
        select = 0;
        COUNTER = 1; //reset counter
    }
    else
    {
        xcount++;
        if(xcount == 4000)
        {
            select = 1;
            xcount = 0;
            COUNTER =1;
            LIC = 1;
            v_com_code();
        }
        else
            select = 0;
    }
}
else
{
    select = 0;
    xcount = 0;
}

return select;
}

```

```

int l_detect_nv(void)
{
char rdetect, xdetect;
int select;

    xdetect = PORTD;
    xdetect &= 0x08;
    rdetect = PORTD;
    rdetéct &= 0x10;

    if(xdetect != 0x00 && rdetect == 0x10)

```

```

        select = 2;
    else if (xdetect == 0x00 && rdetect != 0x10)
        select = 1;
    else
        select = 0;

    return select;
}

```

```

int local_control(void)
{
    static int ldtmf_code[4];
    static int lcount = 0;
    static int lselect;
    static int lreading;
    static int lbusy;
    int lready;
    int lfin;
    static int lvalid;

    switch(lselect)
    {
        case 0:
            lready = dtmf_ready();
            if(lready == 0)
            {
                lselect = 0;
                lfin = 0;
            }
            else if(lready == 1)
            {
                lselect = 1; //read code from dtmf
                lfin = 0;
            }
            else
            {
                not_valid_counter
                not_valid_count(0); //reset

                lselect = 0;
                lready = 0;
                lreading = 0;
                lbusy = 0;
            }
        }
    }

```

```

        lcount = 0;
        lfin = 1;
    }
break;

case 1:
    ldtmf_code[lcount] = dtmf_read();
    if(lcount == 3)
    {
        lselect = 2;
        lreading = 0;
        lcount = 0;
        lfin = 0;
    }
    else
    {
        lcount++;
        lselect = 4; //check for dtmf
read
        lreading = 1;
        lfin = 0;
    }
break;

case 2:
    lvalid = c_code_remote(ldtmf_code);
    lselect = 3;
    lfin = 0;
break;

case 3:
    if(lvalid)
    {
        lselect = 0;
        lreading = 0;
        lfin = 1;    //finished
    }
    else
    {
        v_com_invalid();
        lselect = 0; //reset all variables
        lbusy = 0;
        lreading = 0;
        lfin = 1;
    }
break;

```

```

        case 4:
            lready = dtmf_ready();
            if(lready == 0)
            {
                lselect = 0;
                lfin = 0;
            }
            else if(lready == 1)
            {
                lselect = 4;
                lfin = 0;
            }
            else
            {
                not_valid_counter
                not_valid_count(0); //reset

                lselect = 0;
                lready = 0;
                lreading = 0;
                lbusy = 0;
                lcount = 0;
                lfin = 1;
            }
            break;
        }

    return lfin;
}

```

```

int remote_acc_control(void)
{
    static int rdtmf_code[4];
    static int rcount = 0;
    static int rselect = 5;
    static int rreading;
    static int rbusy;
    static int rrbusy;
    static int rrepeat;
    int rready;
    int resp;
    int rfin;

    switch(rselect)
    {

```



```

case 0:
    rready = dtmf_ready();
    if(rready == 0)
        {
            rselect = 0;
            rfin = 0;
        }
    else if(rready == 1)
        {
            rselect = 1; //read code from dtmf
            rfin = 0;
        }
    else
        {
            not_valid_count(0); //reset
            not
            //valid counter

            rselect = 5;
            rready = 0;
            rbusy = 0;
            rcount = 0;
            rfin = 1;
        }
    break;

case 1:
    rdtmf_code[rcount] = dtmf_read();
    if(rcount == 3)
        {
            rselect = 2;
            rreading = 0;
            rcount = 0;
            rfin = 0;
        }
    else
        {
            rcount++;
            rselect = 5; //check for dtmf
            read

            rfin = 0;
        }
    break;

case 2:
    rvalid = a_code_remote(rdtmf_code);
    rselect = 3;
    rfin = 0;

```

```

break;

case 3:
    if(rvalid)
    {
        rselect = 5;
        not_valid_count(0); //reset
        v_com_code();
        rfin = 1;    //finished
    }
    else
    {
        if(not_valid_count(1))    //no
            more tries
                {
                    v_trial_end();
                    rcount = 0;
                    rselect = 5; //reset all

//variables
                    rbusy = 0;
                    rrbusy = 0;
                    rreading = 0;
                    rrepeat = 0;
                    rfin = 1;
                }
            else
                {
                    rselect = 5; //try again
                    rcount = 0;
                    rbusy = 1;
                    rreading = 0;
                    v_repeat();
                    rfin = 0;
                }
    }

break;
case 5:
    rready = dtmf_ready();
    if(rready == 0)
    {
        rselect = 0;
        rfin = 0;
    }
    else if(rready == 1)

```

```

        {
            rselect = 5;
            rfin = 0;
        }
    else
    {
        not_valid_count(0); //reset
not
//valid counter
        rselect = 5;
        rready = 0;
        rreading = 0;
        rbusy = 0;
        rcount = 0;
        rfin = 1;
    }
    break;
}
return rfin;
}

```

```

int remote_com_control(void)
{
    static int rdtmf_code[4];
    static int rcount = 0;
    static int rrepeat;
    static int rselect = 5;
    static int rrbusy;
    static int rreading;
    int rready;
    static int rbusy;
    int resp;
    int rfin;
    static int cvalid;
    int buf;

    if(!rvalid)
    {
        rfin = 1;
        return rfin;
    }

    switch(rselect)
    {
        case 0:

```

```

rready = dtmf_ready();
if(rready == 0)
{
    rselect = 0;
    rfin = 0;
}
else if(rready == 1)
{
    rselect = 1; //read code from dtmf
    rfin = 0;
}
else
{
    not_valid_count(0); //reset
not
//valid counter

    rselect = 5;
    rrepeat = 0;
    rcount = 0;
    rreading = 0;
    rready = 0;
    rrbusy = 0;
    rbusy = 0;
    rfin = 1;
}
break;

case 1:
rdtmf_code[rcount] = dtmf_read();
if(rcount == 3)
{
    rselect = 2;
    rreading = 0;
    rcount = 0;
    rfin = 0;
}
else
{
    rcount++;
    rselect = 5; //check for dtmf read
    rreading = 1;
    rfin = 0;
}
break;

case 2:
cvalid = c_code_remote(rdtmf_code);

```

```

        rselect = 3;
        rfin = 0;
break;

case 3:
    if(cvalid)
    {
        rselect = 5;
        rbusy = 0;
            rrbusy = 0;
            rcount = 0;
            rreading = 0;
        rrepeat = 0;
        rreading = 0;
        rfin = 1;    //finished
    }
    else
    {
        v_com_invalid();
        if(not_valid_count(1))    //no
more tries
            {
                v_trial_end();
                rselect = 5; //reset all

//variables
                rbusy = 0;
                rrbusy = 0;
                    rcount = 0;
                    rreading = 0;
                rrepeat = 0;
                rfin = 1;
            }
        else
        {
            rselect = 5; //try again
            rbusy = 1;
            rreading = 0;
            v_repeat();
            rfin = 0;
        }
    }
}

```

```
break;

case 5:
    rready = dtmf_ready();
    if(rready == 0)
        {
            rselect = 0;
            rfin = 0;
        }
    else if(rready == 1)
        {
            rselect = 5;
            rfin = 0;
        }
    else
        {
            not_valid_count(0); //reset
not
//valid counter

            rselect = 5;
            rready = 0;
            rreading = 0;
            rbusy = 0;
            rcount = 0;
            rfin = 1;
        }
        break;
    }
return rfin;
}
```

**appl2.h**

```
//define
#define LIC (*(unsigned char *) (0x1068)) //Line Interface
                                     // Circuit address
#define COUNTER (*(unsigned char *) (0x1063)) //address of
                                     // counter cct

//functions
void l_init(void); //off hook
void l_init_nc(void); // off hook without display clear
void l_hangup(void); //off hook
int l_detect(void); //on hook if line is active
int l_detect_nv(void); //detects active line without voice
                                     // and line activate
int local_control(void); // local operation
int remote_acc_control(void); //remote access operation
int remote_com_control(void); //remote command operation
```

**appl3.c**

```

/*****
Written by Gerald Aska

```

## Purpose:

```

    Read DTMF check validity of access and command codes,
    check the status of DTMF and limits the number of time
    an error in access code can be made by the user.

```

```

*****/

```

```

#include "appl2.h"
#include "appl3.h"
#include "appl4.h"
#include "lcd.h"
#include <stdio.h>

```

```

#define rdy 0
#define maskhi 0x0f
#define pound 0x0b
#define star 0x0c

```

```

int com_a = 0;
int com_b = 0;
int com_c = 0;
int com_d = 0;
int valid = 0;

```

```

int dtmf_ready(void)
{
    int status, ready, max = 0;

    status = dtmfc & maskhi;

    if(status == rdy)
        ready = 1;
    else
        ready = 0;

    if(!ready)
    {
        max = not_ready_count(1);
    }
}

```



```
        else
        {
            not_ready_count(0);
        }

        if (max)
            ready = 2;

    return(ready);
}

int dtmf_read(void)
{
    int resp;

    resp = dtmfd & maskhi;

    return(resp);
}

void dtmf_init(void)
{
    char status;

    status = dtmfc;
    dtmfc = 0;
    dtmfc = 0;
    dtmfc = 0x08;
    dtmfc = 0;
    status = dtmfc;
    dtmfc = 0x0d;
    dtmfc = 0;
}

void dtmfs_init(void)
{
    char status;

    status = dtmfc;
    dtmfc = 0;
    dtmfc = 0;
}
```

```
dtmfc = 0x08;
dtmfc = 0;
status = dtmfc;
dtmfc = 0x0d;
dtmfc = 0x0c;
}
```

```
int a_code_remote(int ra_code[])
{
static int code[4] = {0x02,0x07,0x05,0x02};

    if(code[0]==ra_code[0] && code[1]==ra_code[1] &&
code[2]==ra_code[2] && code[3]==ra_code[3])
        valid = 1;
    else
        valid = 0;

return valid;
}
```

```
int c_code_remote(int rc_code[])
{
int c_code[3];

    if(rc_code[0] == 10)
        rc_code[0] = 0;
    if(rc_code[1] == 10)
        rc_code[1] = 0;
    if(rc_code[2] == 10)
        rc_code[2] = 0;
    if(rc_code[3] == 10)
        rc_code[3] = 0;

    if(rc_code[0] <= 9)
        com_a = 1;
    if(rc_code[1] <= 9)
        com_b = 1;
    if(rc_code[2] <= 8 || rc_code[2] == 0x0b ||
rc_code[2] == 0x0c)
```

```
        com_c = 1;
    if(rc_code[3] <= 9)
        com_d = 1;

    if (com_a == 1 && com_b == 1 && com_c == 1 && com_d
== 1)
    {
        c_code[0] = (rc_code[0] * 10) + rc_code[1];
        c_code[1] = rc_code[2];
        c_code[2] = rc_code[3];
        valid = do_command(c_code);
    }
    else
        valid = 0;

    return valid;

}
```

```
int not_valid_count(int set)
{
    static int not_valid;
    int max;

    if(set == 1)
        not_valid++;
    else
        not_valid = 0;

    if(not_valid == 3)
        max = 1;
    else
        max = 0;

    return max;
}
```

**appl3.h**

```
//define
#define dtmfd (*(unsigned char *) (0x1066)) //DTMF data
//address
#define dtmfc (*(unsigned char *) (0x1067)) //DTMF control
//address

//functions
int dtmf_ready(void);
int dtmf_read(void);
int c_code_local(int []);
int c_code_remote(int []);
int a_code_remote(int []);
int not_valid_count(int);
void dtmf_init(void);
void dtmfs_init(void);
```

**appl4.c**

```

/*****
Written by Gerald Aska

```

Purpose:

Indicates to io.c the IVs to change according to the commands entered by the user and sets the time for a DTMF ready state to occur.

```

*****/

```

```

#include "voice.h"
#include "io.h"
#include "appl4.h"
#include "lcd.h"
#include <stdio.h>

```

```

int do_command(int do_com[])
{
int IV_temp, light, heat, valid = 1;

switch(do_com[0])
{
case 0: //light
switch(do_com[1])
{
case 0x0b: //turn off
peek_IV_bool(0xa0,0x21,0x07,"C",&light);
if(light)
poke_IV_bool(0xa0,0x21,0x07,"C",0);
else
v_alreadyoff();
break;
case 0x0c: //turn on
peek_IV_bool(0xa0,0x21,0x07,"C",&light);
if(!light)
poke_IV_bool(0xa0,0x21,0x07,"C",1);
else
v_alreadyon();
break;
default:
valid = 0;

```

```

        break;
    }
    break;

    case 1:                //AC
        switch(do_com[1])
        {
            case 0x0b:     //turn off

poke_IV_bool(0xa0,0x74,0x07,"C",0);
            break;
            case 0x0c:     //turn on

poke_IV_bool(0xa0,0x74,0x07,"C",1);
            break;
            default:       //set temp
                do_com[1] = do_com[1] * 10;
                IV_temp = do_com[1] + do_com[2];

poke_IV_int(0xa0,0x74,0x08,"C",IV_temp);
            break;
        }
    break;

    case 2:
        switch(do_com[1])
        {
            case 0x0b:     //turn off

poke_IV_bool(0xa1,0x74,0x07,"C",&heat);
                if(heat)

poke_IV_bool(0xa1,0x74,0x07,"C",0);
                else
                    v_alreadyoff();
            break;
            case 0x0c:     //turn on

poke_IV_bool(0xa1,0x74,0x07,"C",&heat);
                if(!heat)

poke_IV_bool(0xa1,0x74,0x07,"C",1);
                else
                    v_alreadyon();
            break;
            default:       //set temp
                do_com[1] = do_com[1] * 10;

```

```
        IV_temp = do_com[1] + do_com[2];
poke_IV_int(0xa1,0x74,0x08,"C",IV_temp);
        break;
    }
    break;

    default:
        valid = 0;
    break;
}

return valid;
}
```

```
int not_ready_count(int set)
{
static long int not_ready;
int max;

    if(set == 1)
        not_ready++;
    else
        not_ready = 0;

    if(not_ready == 5000)
        max = 1;
    else
        max = 0;

return max;
}
```

**app14.h**

```
//functions  
int do_command(int []);  
int not_ready_count(int);
```



**voice.c**

```

/*****
Written by Gerald Aska

```

## Purpose:

To activate the Speech Synthesizer with the prompts and responses for the user interface.

```

*****/

```

```

//Voice response functions
//funtions

```

```

#include "lcd.h"
#include "voice.h"
#include <stdio.h>

```

```

int sentlen;

```

```

void v_com_code(void)
{
    char a[] = "Please enter command code.";
    sentlen = 26;

    disp_gotoxy(12,0);
    printf("V");
    voice_iom(a, sentlen);
}

```

```

void v_ack(void)
{
    char ack[] = "Task completed.";
    sentlen = 15;

    voice_iom(ack, sentlen);
}

```

```

void v_nack(void)
{
    char nack[] = "No response from device.";
    sentlen = 24;

    voice_iom(nack, sentlen);
}

```

```
void v_trial_end(void)
{
    char b[] = "No more tries.";

    sentlen = 14;
    voice_iom(b, sentlen);
}

void v_repeat(void)
{
    char c[] = "Enter access code again.";

    sentlen = 24;
    voice_iom(c, sentlen);
}

void v_acc_code(void)
{
    char d[] = "Please enter access code.";
    int buff;
    sentlen = 25;
    buff = SPEECH;
    disp_gotoxy(12,1);
    printf("%d",buff);

    voice_iom(d, sentlen);
}

void v_com_invalid(void)
{
    char g[] = "In valid command code.";

    sentlen = 22;
    voice_iom(g, sentlen);
}

void v_alreadyon(void)
{
    char h[] = "Device is already on.";

    sentlen = 21;
    voice_iom(h, sentlen);
}

void v_alreadyoff(void)
{
    char i[] = "Device is already off.";

    sentlen = 22;
```

```
    voice_iom(i, sentlen);
    }

void v_qnext_com(void)
    {
    char j[] = "Do you want to enter another command?";

    sentlen = 37;
    voice_iom(j, sentlen);
    }

void voice_iom(char speak[], int num)
    {
    int n = 0;

    SPEECH = 0x01;
    SPEECH = 1;
    SPEECH = 'Y';

    while (n <= num)
        {
        SPEECH = speak[n];
        n++;
        }

    SPEECH = 0x0D;

    }
```

**voice.h**

```
#define SPEECH (*(unsigned char*) (0x1062)) //voice cct
                                                //address

#define vmaskhi 0x10

//header function for voice module

//funtions
void voice_io(char[]);
void voice_iom(char[], int);
void v_com_code(void);
void v_trial_end(void);
void v_repeat(void);
void v_acc_code(void);
void v_com_invalid(void);
void v_alreadyon(void);
void v_alreadyoff(void);
void v_qnext_com(void);
void v_nconnect(void);
void v_ack(void);
void v_nack(void);
```

**main.c**

```

/*****
*****

```

Modified by Gerald Aska

Purpose:

To call initialization subroutines and start the execution of the CEBus communication layers

```

*****
*****/

```

```
#include "includes.h"
```

```

void CEBus_Init(void);
void CEBus_Proc(void);

```

```
non_banked void main()
```

```

{
int i = 0, j = 0 ;

```

```

    //init display, dtmf, cenode, controller and enable interrupts
    _opc(0x0E);
    disp_init();
    dtmf_init();
    l_init();
    CEBus_Init();

```

```

    //main loop
    while(1)
    {
        CEBus_Proc();
        cntl();
    }

```

```

}

```

**include.h**

```
//Include file for main.c
//
//version 1.0

#include <stdio.h>
#include "io6811f1.h"
#include "hal.h"
#include "lcd.h"
#include "appl1.h"
#include "appl2.h"
#include "appl3.h"
#include "appl4.h"
```

**io.c**

```

/*****
Modified by Gerald Aska

```

**Purpose:**

To alter the IVs according to user request and initiate automatic data transfer to controlled device.

```

*****/

```

```

#define IO_GLOBALS_OWNER

```

```

#include <string.h>
#include "cebtypes.h"
#include "io.h"
#include "loglink.h"
#include "network.h"
#include "msgxfer.h"
#include "user.h"

```

```

#include "lcd.h"
#include "voice.h"

```

```

#pragma language=extended
#define HC_REG_OFFSET 0x1000 /* set base i/o control register address */
#include <stdio.h>
#include <io68111.h>
#include <int6811.h>

```

```

#define TRUE 1
#define FALSE 0

```

```

static void init(void);
//static void digital_in0(void);
//static void analog_in0(void);

```

```

/*-----
** io_routine()
**
** Called by the CON layer to execute object specific user I/O routines.
**
** Inputs:
**   int index
**       The number of the I/O routine to be executed.
**       Some special values exist:
**       0: user I/O initialization

```

```

**-----
*/
void io_routine(int index)
    {
char value[2] ;
int light, l_ack, h_ack, hs_ack;
int heat_sw, heat_temp;
static int lack_count, hack_count, hs_ack_count;
static int send, sendh, hs_send;

    switch (index)
        {

        case 0:
            init();
            disp_clear();
            printf("Initialization..");
            peek_IV_data(0xa0,0,1,"h",value,2) ;
            disp_gotoxy(0,1);
            printf("Sys: %d%d",(int)value[0],(int)value[1]) ;
            peek_IV_data(0xa0,0,1,"a",value,2) ;
            disp_gotoxy(8,1);
            printf("Mac: %d%d",(int)value[0],(int)value[1]) ;
            return;

        case 2:
            if(get_IV_bool("C", &light) == 2)
            {
                if(light)
                {
                    poke_IV_bool(0xa0, 0x21, 0x06, "C", 1);
                    send = 1;
                }
                else
                {
                    poke_IV_bool(0xa0, 0x21, 0x06, "C", 0);
                    send = 1;
                }
            }
            return;

        case 3:
            if(send)
            {
                if(get_IV_bool("C", &l_ack) == 2)
                {

```



```

        v_ack();
        send = 0;
        lack_count = 0;
    }
    else
    {
        lack_count++;
        if(lack_count == 10)
        {
            v_nack();
            lack_count = 0;
            send = 0;
        }
    }
}
return;

case 4:
if(get_IV_bool("C", &heat_sw) == 2)
{
    if(heat_sw)
    {
        poke_IV_bool(0xa1, 0x74, 0x05, "C", 1);
        sendh = 1;
    }
    else
    {
        poke_IV_bool(0xa1, 0x74, 0x05, "C", 0);
        sendh = 1;
    }
}
return;

case 5:
if(sendh)
{
    if(get_IV_bool("C", &h_ack) == 2)
    {
        v_ack();
        sendh = 0;
        hack_count = 0;
    }
    else
    {

```

```
        hack_count++;
        if(hack_count == 10)
        {
            v_nack();
            hack_count = 0;
            sendh = 0;
        }
    }
}
return;

    case 6:
    if(get_IV_int("C", &heat_temp) == 2)
    {
        poke_IV_int(0xa1, 0x74, 0x02, "C", heat_temp);
        hs_send = 1;
    }

return;

    case 7:
    if(hs_send)
    {
        if(get_IV_int("C", &hs_ack) == 2)
        {
            v_ack();
            hs_send = 0;
            hs_ack_count = 0;
        }
        else
        {
            hs_ack_count++;
            if(hs_ack_count == 10)
            {
                v_nack();
                hs_ack_count = 0;
                hs_send = 0;
            }
        }
    }
}
return;
```

```

        default:
        return;
    }

}

/*
 * Function:  init()
 *
 * Purpose:
 *   Initializes the analog input conversion and timer output compare
 *   subsystems. This routine is called once at system power-up while
 *   CAL is initializing.
 *
 * Returns:
 *   void
 *
 * Input:
 *   none
 *
 * Output:
 *   none
 */
static void init(void)
{
    int    j;

    OPTION &= 0x3f;          // make sure CSEL clear
    OPTION |= 0x80;         // enable analog input

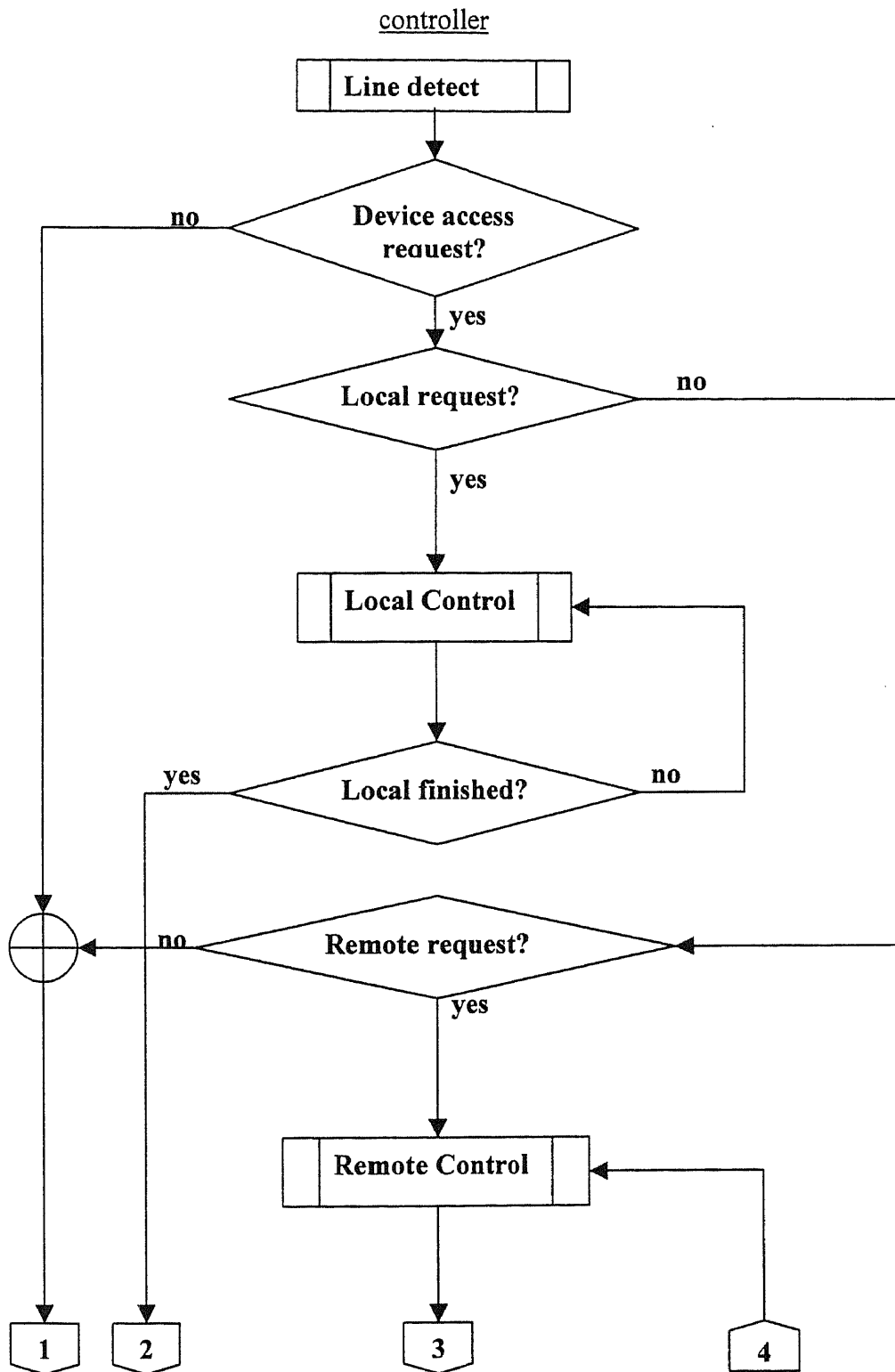
    for (j = 0; j < 100; ++j)
        /*
         * delay at least 100 microseconds so analog
         * converter stable
         */
        ;

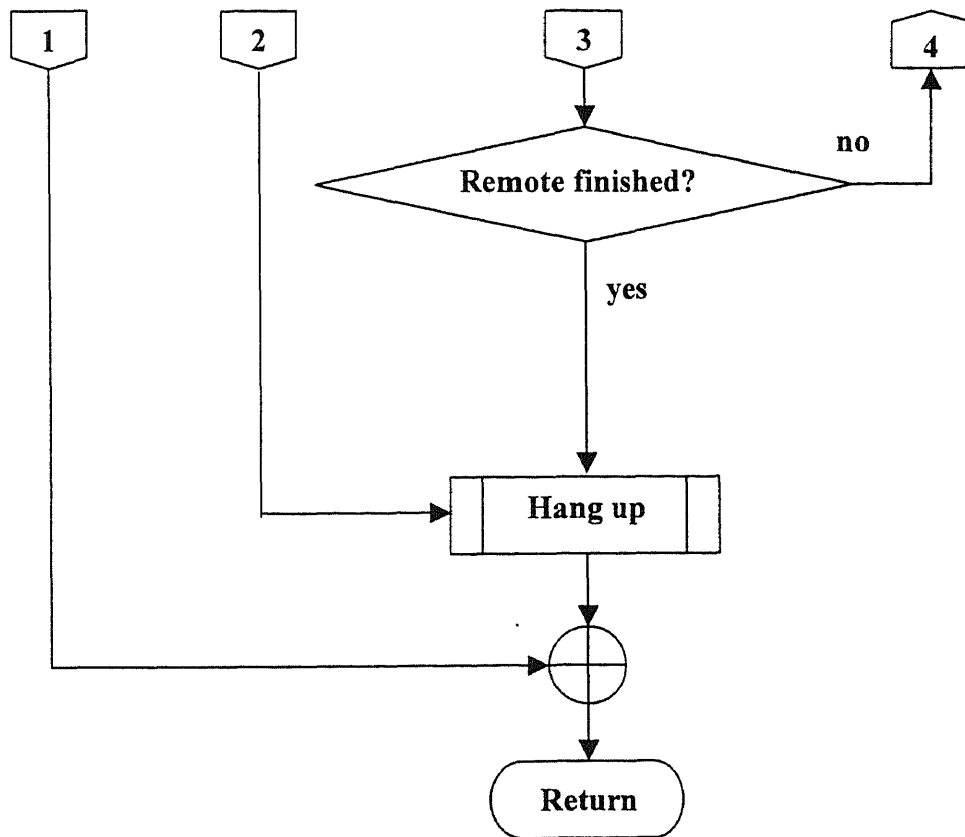
    DDRD = 0;  // make all port D pins input (default from reset)
}

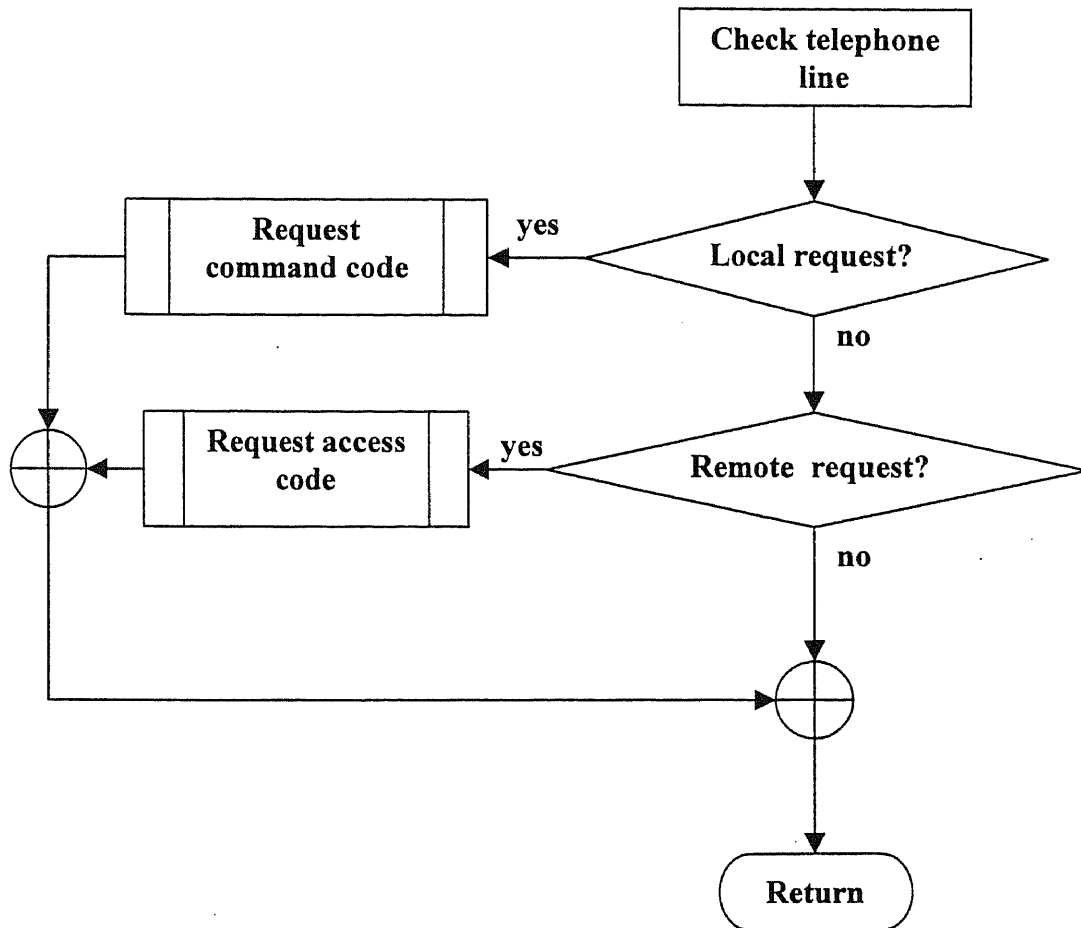
```

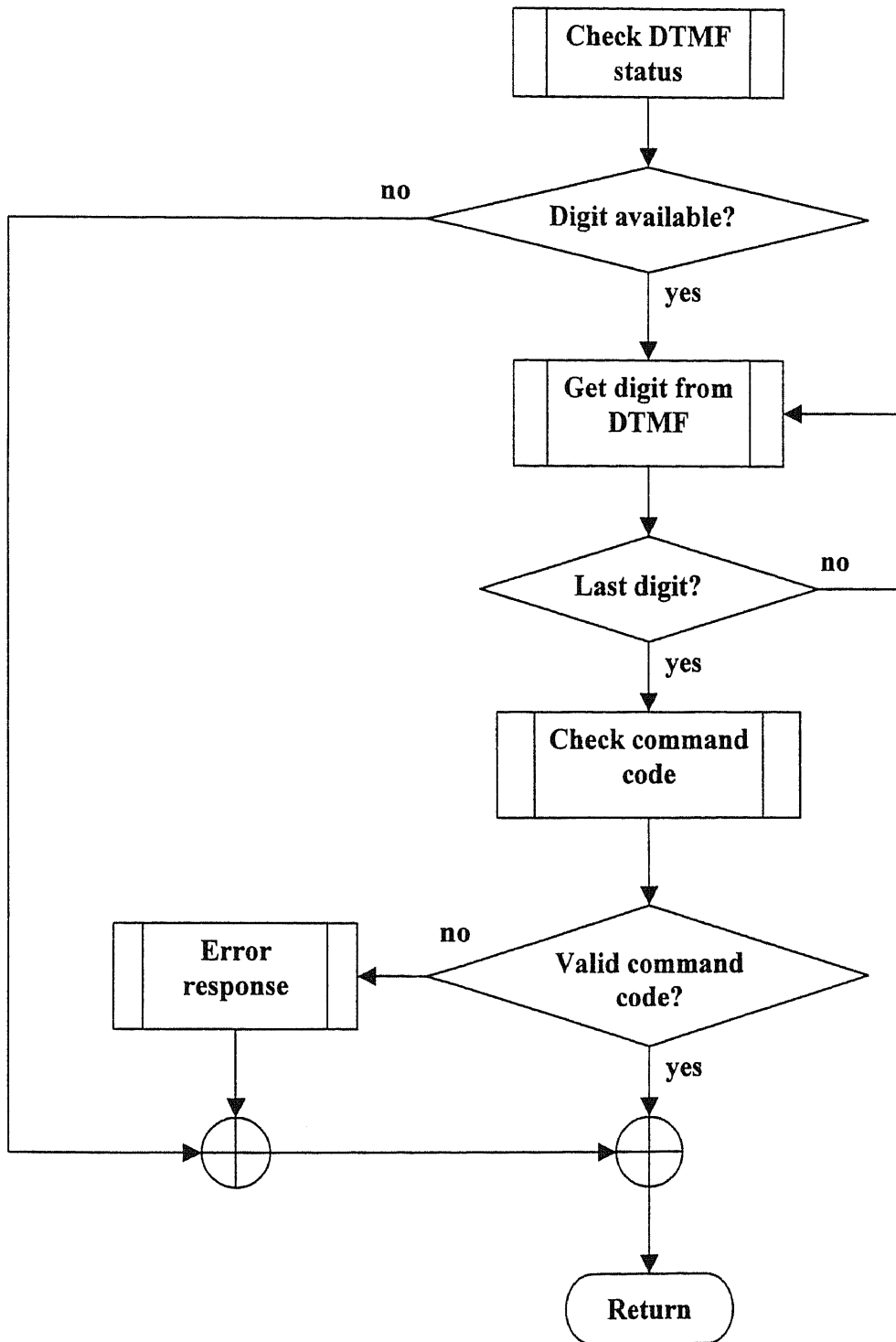
APPENDIX B

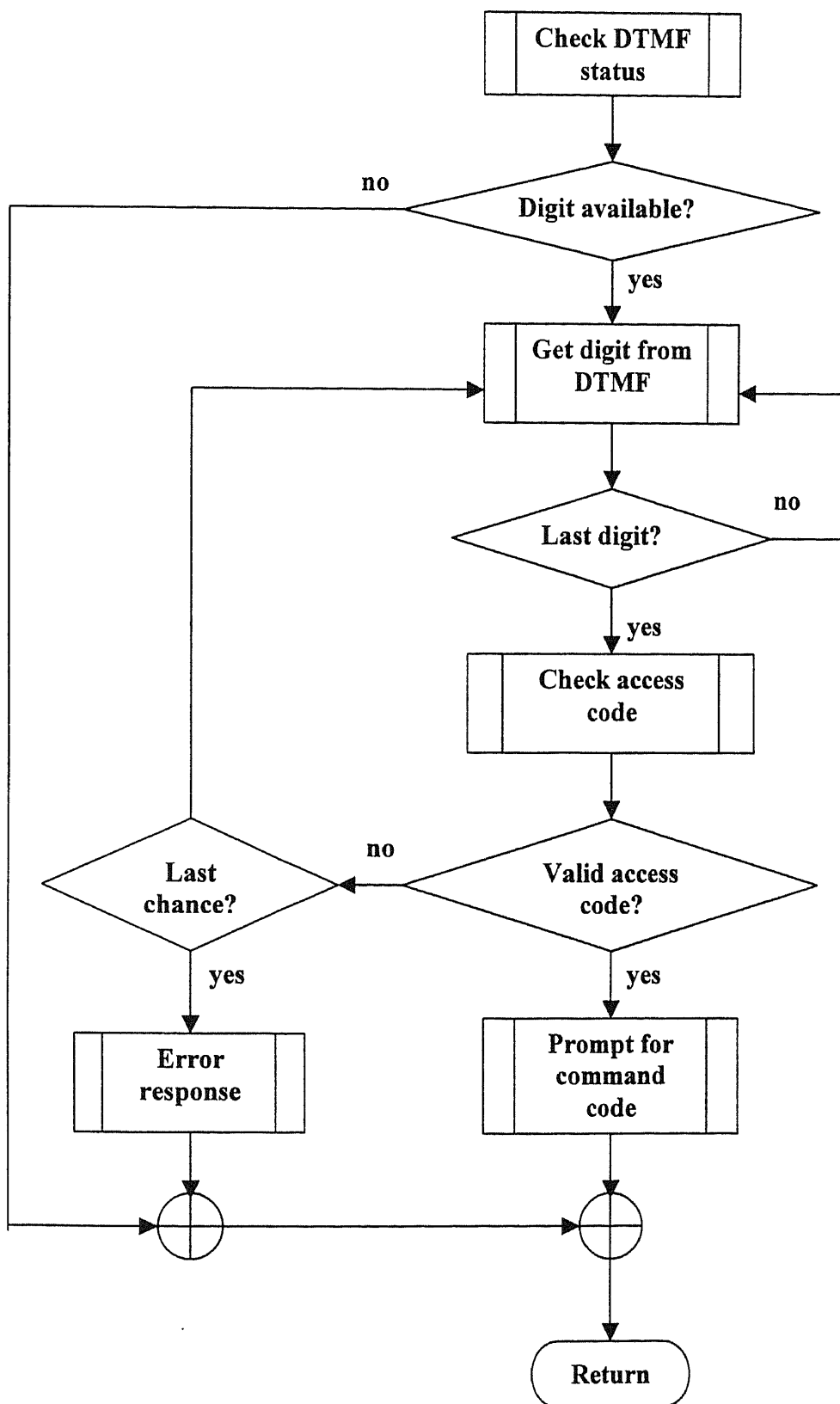
CONTROLLER APPLICATION FLOWCHARTS



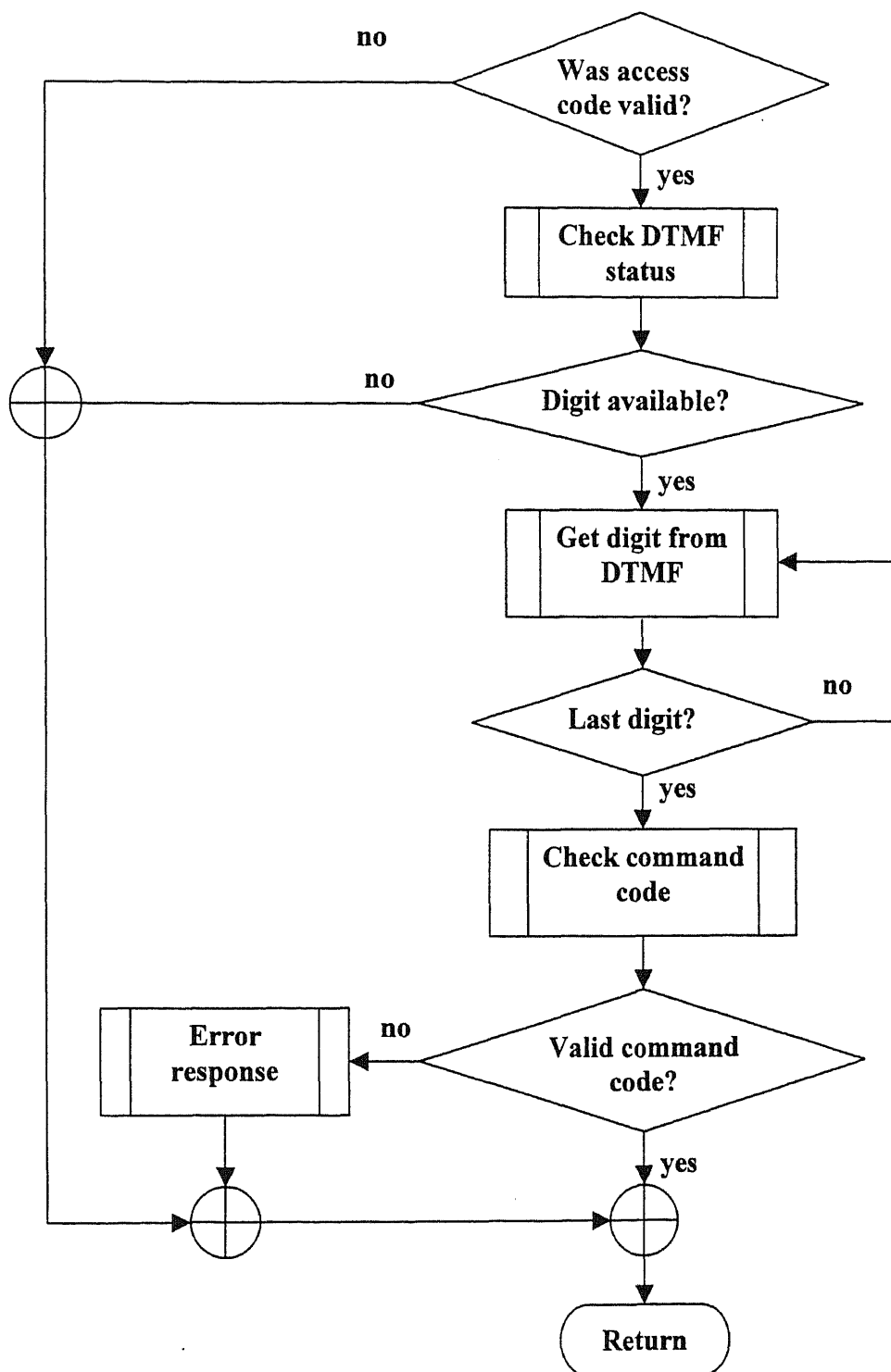
Controller (cont'd)

Line Detect

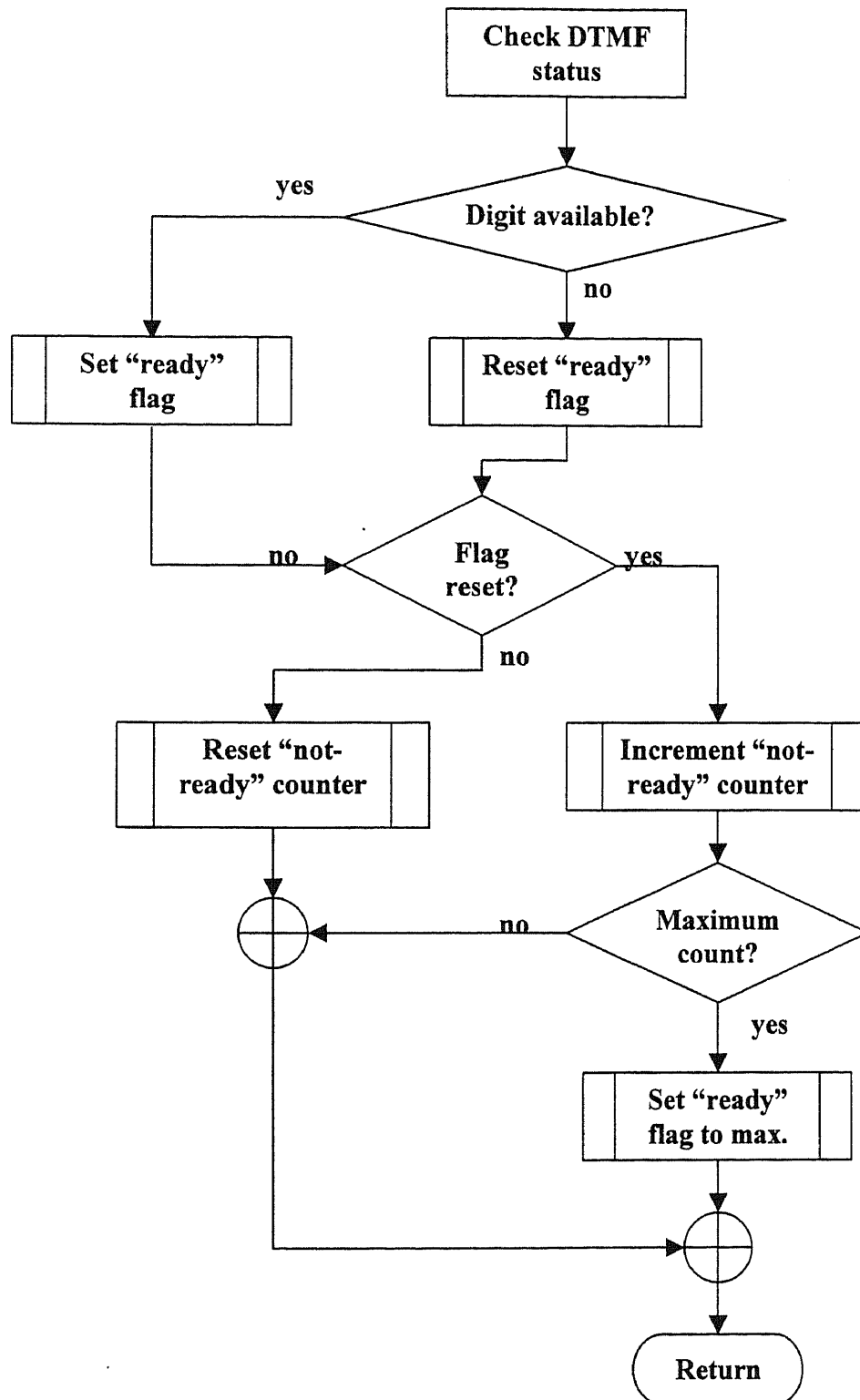
Local Control

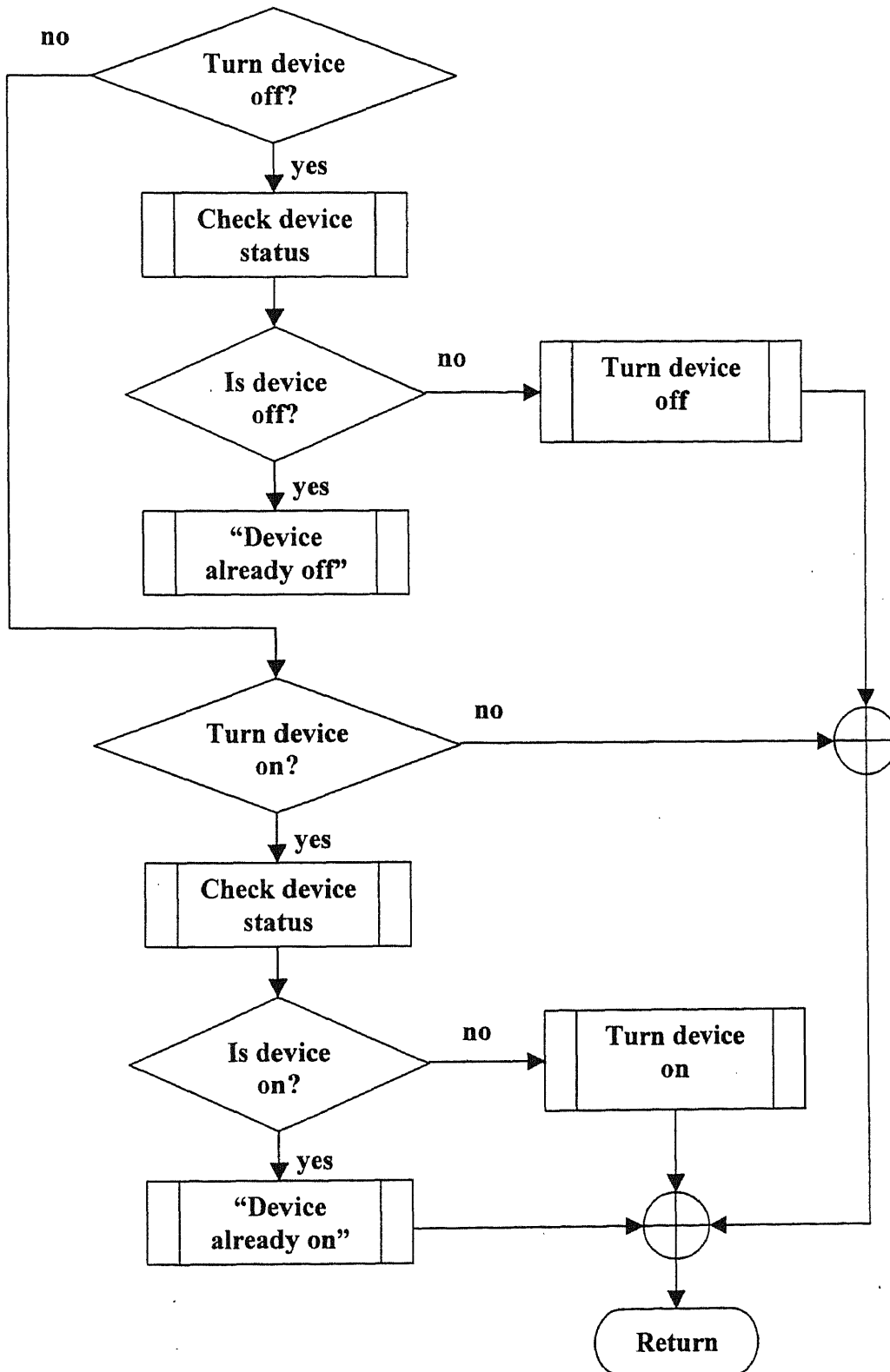
Remote Access Control



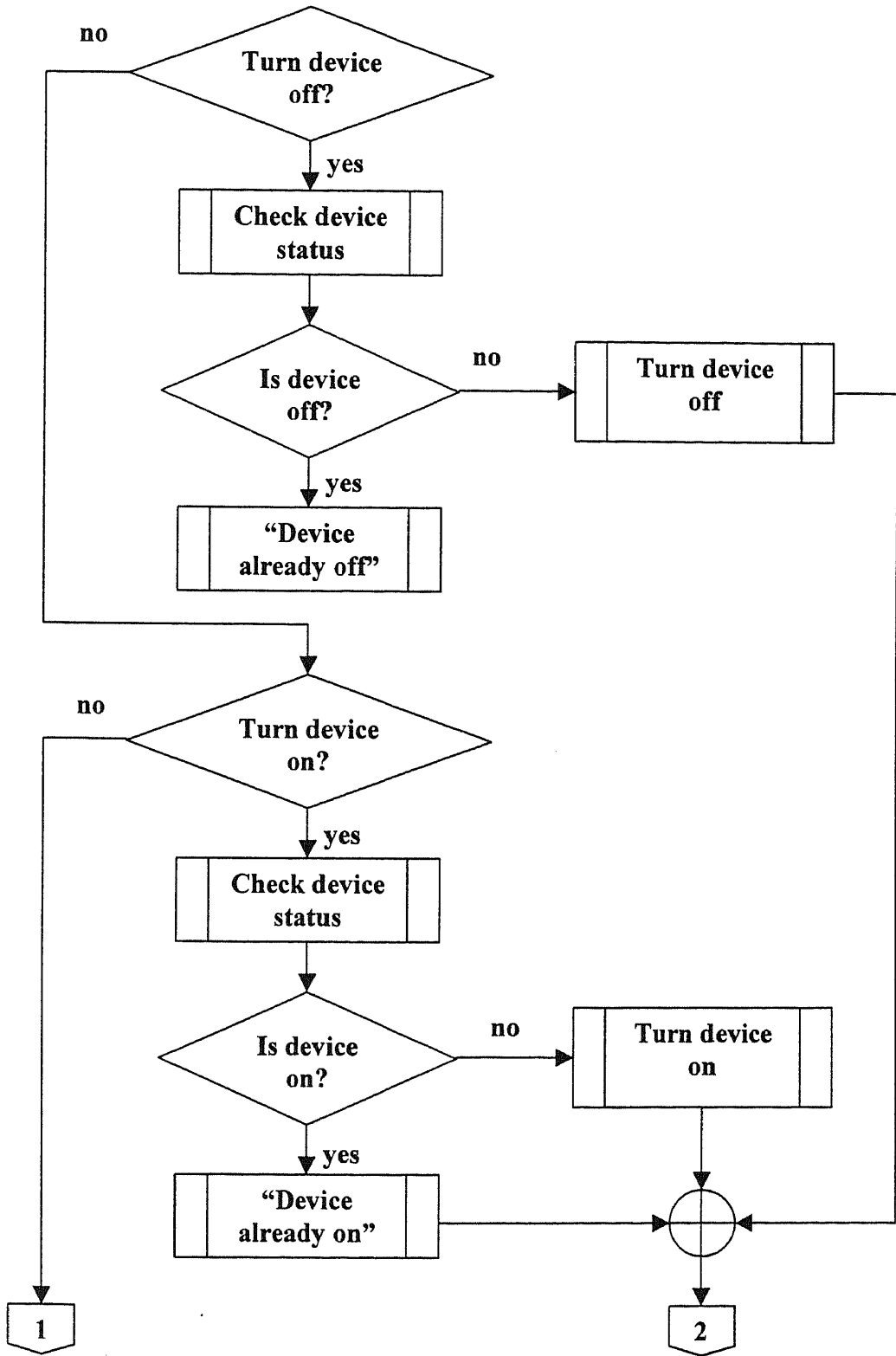
Remote Command Control

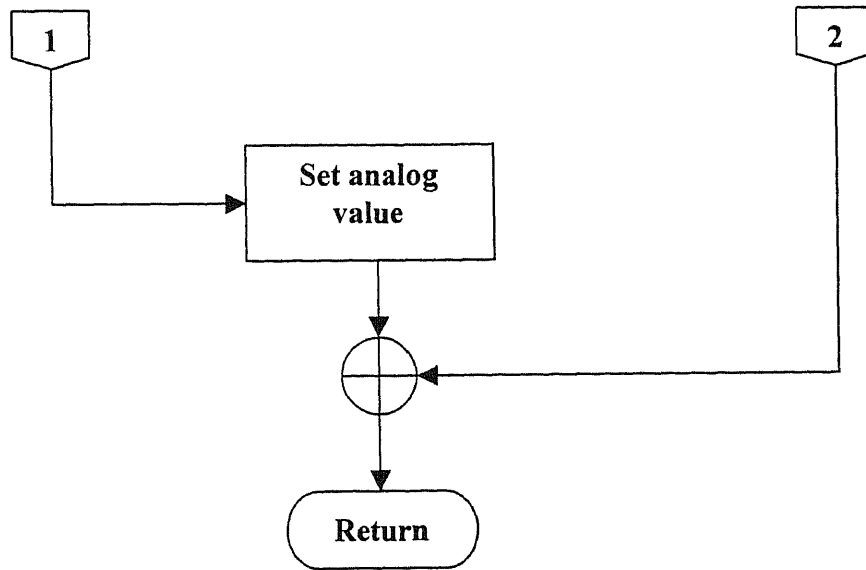
## DTMF Status Check



Command Activation (without analog adjustment)

Command Activation (with analog adjustment)



Command Activation (with analog adjustment) – cont'd

## APPENDIX C

### COMPILE BATCH FILE

#### compile.bat

```
@echo off
cls
ICC6811 -mb -e -g -L -q -K -P -RRCODE main
pause
ICC6811 -mb -e -g -L -q -K -P ivs
pause
ICC6811 -mb -e -g -L -q -K -P io
pause
icc6811 -mb -e -g -L -q -K -P user1
pause
icc6811 -mb -e -g -L -q -K -P -RRCODE hc11hal
pause
icc6811 -mb -e -g -L -q -K -P -RRCODE lcd
pause
icc6811 -mb -e -g -L -q -K -P -RRCODE appl4
pause
icc6811 -mb -e -g -L -q -K -P -RRCODE appl3
pause
icc6811 -mb -e -g -L -q -K -P -RRCODE appl2
pause
icc6811 -mb -e -g -L -q -K -P -RRCODE appl1
pause
icc6811 -mb -e -g -L -q -K -P -RRCODE voice
pause

xlink -f hc11 main ivs user1 io hc11hal lcd appl4 appl3 appl2 appl1 voice cb11libb
msgx11b netw11b -o done -l done.map
:xlink -f hc11 main -o done -l done.map
pause
sconv done.a07 code.a07
keystack "2code.a07" 13 "M" 13 "3" 13 "A"
eepl
```

## APPENDIX D

### CONTEXTS, OBJECTS AND IVS

The following Contexts, Objects and IVs are used for the Controller:

<b>CEBus Remote Access Controller</b>										
#	CONTEXT								ID	
A0	Universal () : univ.cxl								00	
	#	OBJECT								CLASS
	01	Node Control (Device Control) : nodectrl.cob								01
		IV	NAME	MEM	RW	TYPE	SIZE	RL	WL	VALUE
		s	serial_#	ROM	R	String	21	3	3	GA1198
		n	manuf_name	ROM	R	String	21	3	3	GA CEBus Solutions
		m	manuf_model	ROM	R	String	21	3	3	CON997
		c	product_class	ROM	R	String	21	3	3	UNLISTED
		h	system_addr	NVM	R/W	Data	1x2	3	3	00 03
		a	mac_addr	NVM	R/W	Data	1x2	3	3	00 01
		o	context_list	ROM	R	Data	4x2	3	3	a0 00 a0 21 a0 74 a1 74
		f	configured	RAM	R/W	Boolean	1	3	3	True
	i	setup	RAM	R/W	Integer	2	3	3	0	
	u	user_feedback	RAM	R/W	Integer	2	3	3	0	
	02	Context Control (Context Control) : cntxctrl.cob								02
IV		NAME	MEM	RW	TYPE	SIZE	RL	WL	VALUE	
	o	object_list	ROM	R	Data	3x2	3	3	01 01 02 02 16 03	
03	Data Memory (Event Manager) : datamem.cob								16	
	IV	NAME	MEM	RW	TYPE	SIZE	RL	WL	VALUE	
	C	current_index	RAM	R/W	Integer	2	3	3	0	
	I	memory_block	RAM	R/W	Data	1x25	3	3		

CEBus Remote Access Controller												
#	CONTEXT											ID
A0	Light (Switches) : light.cxl											21
	#	OBJECT										CLASS
	01	Context Control (Context Control) : cntxctrl.cob										02
	IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE			
	o	object_list	ROM	R	Data	4x2	3	3	02 01 06 06 06 07 06 08			
	06	Binary Sensor (Light Switch) : bsensor.cob										06
	IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE			
	C	current_state	RAM	R	Boolean	1	3	3	False			
	R	report_condition	NVM	R/W	Data	1x4	3	3	43 ed 00			
	H	report_header	NVM	R/W	Data	1x6	3	3	21 06 45 43 f5			
	A	dest_address	NVM	R/W	Data	1x4	3	3	00 02 00 01			
	P	previous_value	RAM	R	Boolean	1	3	3	False			
	07	Binary Sensor (Light Intermediate Switch) : bsensor.cob										I/O # (2) 06
	IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE			
	C	current_state	RAM	R	Boolean	1	3	3	False			
	08	Binary Sensor (Ack) : bsensor.cob										I/O # (3) 06
	IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE			
	C	current_state	RAM	R/W	Boolean	1	3	3	False			



CEBus Remote Access Controller										
#	CONTEXT									ID
A1	Refrigerator/Freezer (Heater) : refrigfr.cxl									74
#	OBJECT									CLASS
01	Context Control (Context Control) : cntxctrl.cob									02
	IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE	
	o	object_list	ROM	R	Data	7x2	3	3	02 01, 07 02, 05 05, 06 07, 08 08, 06 09, 08 0a	
02	Analog Control (Temperature Setting) : analctrl.cob									07
	IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE	
	N	min_value	ROM	R	Integer	2	3	3	65	
	M	max_value	ROM	R	Integer	2	3	3	85	
	D	default_value	ROM	R	Integer	2	3	3	72	
	C	current_value	RAM	R/W	Integer	2	3	3	0	
	R	report_condition	NVM	R/W	Data	1x4	3	3	43 ed 31	
	H	report_header	NVM	R/W	Data	1x6	3	3	a1 74 02 45 43 f5	
	A	dest_address	NVM	R/W	Data	1x4	3	3	00 02 00 01	
	P	previous_value	RAM	R	Integer	2	3	3	0	
05	Binary Switch (Heater Switch) : bswitch.cob									05
	IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE	
	C	current_position	RAM	R/W	Boolean	1	3	3	False	
	R	report_condition	NVM	R/W	Data	1x4	3	3	43 ed 00	
	H	report_header	NVM	R/W	Data	1x6	3	3	a1 74 05 45 43 f5	
	A	dest_address	NVM	R/W	Data	1x4	3	3	00 02 00 01	
	P	previous_value	RAM	R	Boolean	1	3	3	False	
07	Binary Sensor (Heater Intermediate Switch) : bsensor.cob									I/O # (4)   06
	IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE	
	C	current_state	RAM	R	Boolean	1	3	3	False	
08	Analog Sensor (Temperature Storage) : analsens.cob									I/O # (6)   08
	IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE	
	C	current_value	RAM	R	Integer	2	3	3	0	
09	Binary Sensor (Switch Ack) : bsensor.cob									I/O # (5)   06
	IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE	
	C	current_state	RAM	R/W	Boolean	1	3	3	False	
0A	Analog Sensor (Temperature Ack) : analsens.cob									I/O # (7)   08
	IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE	
	C	current_value	RAM	R/W	Integer	2	3	3	0	

CEBus Remote Access Controller										
#	CONTEXT									ID
A0	Refrigerator/Freezer (AC) : refrigfr.cxl									74
#	OBJECT									CLASS
01	Context Control (Context Control) : cntxctrl.cob									02
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
o	object_list	ROM	R	Data	7x2	3	3	02 01,07 02,05 05,06 07,08 08,06 09,08 0a		
02	Analog Control (Temperature Setting) : analctrl.cob									07
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
N	min_value	ROM	R	Integer	2	3	3	0		
M	max_value	ROM	R	Integer	2	3	3	40		
D	default_value	ROM	R	Integer	2	3	3	32		
C	current_value	RAM	R/W	Integer	2	3	3	0		
R	report_condition	NVM	R/W	Data	1x4	3	3	43 ed 31		
H	report_header	NVM	R/W	Data	1x6	3	3	a0 74 02 45 43 f5		
A	dest_address	NVM	R/W	Data	1x4	3	3	00 02 00 01		
P	previous_value	RAM	R	Integer	2	3	3	0		
05	Binary Switch (AC Switch) : bswitch.cob									05
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
C	current_position	RAM	R/W	Boolean	1	3	3	False		
07	Binary Sensor (AC Intermediate Switch) : bsensor.cob									06
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
C	current_state	RAM	R	Boolean	1	3	3	False		
08	Analog Sensor (Temperature Storage) : analsens.cob									08
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
C	current_value	RAM	R	Integer	2	3	3	0		
09	Binary Sensor (Switch Ack) : bsensor.cob									06
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
C	current_state	RAM	R/W	Boolean	1	3	3	False		
0A	Analog Sensor (Temperature Ack) : analsens.cob									08
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
C	current_value	RAM	R	Integer	2	3	3	0		

The following Contexts, Objects and IVs were used on the computer to simulate a CEBus device:

CEBus Device Simulator										
#	CONTEXT									ID
A0	Universal () : univ.cxl									00
#	OBJECT									CLASS
01	Node Control (Device Control) : nodectrl.cob									01
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
n	manuf_name	ROM	R	String	21	3	3	GA CEBus Solutions		
h	system_addr	NVM	R/W	Data	1x2	3	3	00 01		
a	mac_addr	NVM	R/W	Data	1x2	3	3	00 02		
o	context_list	ROM	R	Data	4x2	3	3	a0 00 a0 21 a0 74 a1 74		
f	configured	RAM	R/W	Boolean	1	3	3	False		
i	setup	RAM	R/W	Integer	2	3	3	0		
u	user_feedback	RAM	R/W	Integer	2	3	3	0		
02	Context Control (Context Control) : cntxctrl.cob									02
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
o	object_list	ROM	R	Data	3x2	3	3	01 01 02 02 16 03		
03	Data Memory (Event Manager) : datamem.cob									16
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
C	current_index	RAM	R/W	Integer	2	3	3	0		
I	memory_block	RAM	R/W	Data	1x25	3	3			

CEBus Device Simulator										
#	CONTEXT									ID
A0	Light (Switch) : light.cxl									21
#	OBJECT									CLASS
01	Context Control (Context Control) : cntxctrl.cob									02
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
o	object_list	ROM	R	Data	3x2	3	3	02 01 05 06 06 07		
06	Binary Switch (Light Switch) : bswitch.cob									05
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
C	current_position	RAM	R/W	Boolean	1	3	3	True		
R	report_condition	NVM	R/W	Data	1x4	3	3	43 ed 00		
H	report_header	NVM	R/W	Data	1x6	3	3	21 08 45 43 f5		
A	dest_address	NVM	R/W	Data	1x4	3	3	00 01 00 03		
P	previous_value	RAM	R	Boolean	1	3	3	True		
07	Binary Sensor (Intermediate Switch) : bsensor.cob									06
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE		
C	current_state	RAM	R	Boolean	1	3	3	False		

CEBus Device Simulator												
#	CONTEXT											ID
A1	Refrigerator/Freezer (Heater) : refrigfr.cxl											74
#	OBJECT											CLASS
01	Context Control (Context Control) : cntxctrl.cob											02
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE				
o	object_list	ROM	R	Data	3x2	3	3	02 01 07 02 05 05				
02	Analog Control (Temperature Setting) : analctrl.cob											07
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE				
C	current_value	RAM	R/W	Integer	2	3	3	0				
R	report_condition	NVM	R/W	Data	1x4	3	3	43 ed 31				
H	report_header	NVM	R/W	Data	1x6	3	3	a1 74 0a 45 43 f5				
A	dest_address	NVM	R/W	Data	1x4	3	3	00 01 00 03				
P	previous_value	RAM	R/W	Integer	2	3	3	0				
05	Binary Switch (Switch) : bswitch.cob											05
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE				
C	current_position	RAM	R/W	Boolean	1	3	3	False				
R	report_condition	NVM	R/W	Data	1x4	3	3	43 ed 00				
H	report_header	NVM	R/W	Data	1x6	3	3	a1 74 09 45 43 f5				
A	dest_address	NVM	R/W	Data	1x4	3	3	00 01 00 03				
P	previous_value	RAM	R	Boolean	1	3	3	False				

CEBus Device Simulator												
#	CONTEXT											ID
A0	Refrigerator/Freezer (AC) : refrigfr.cxl											74
#	OBJECT											CLASS
01	Context Control (Context Control) : cntxctrl.cob											02
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE				
o	object_list	ROM	R	Data	3x2	3	3	02 01 07 02 05 05				
02	Analog Control (Temperature Setting) : analctrl.cob											07
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE				
N	min_value	ROM	R	Integer	2	3	3	00				
M	max_value	ROM	R	Integer	2	3	3	40				
D	default_value	ROM	R	Integer	2	3	3	32				
C	current_value	NVM	R/W	Integer	2	3	3	72				
R	report_condition	NVM	R/W	Data	1x4	3	3	43 ed 31				
H	report_header	NVM	R/W	Data	1x6	3	3	21 09 45 43 f5				
A	dest_address	NVM	R/W	Data	1x4	3	3	00 01 00 03				
P	previous_value	RAM	R	Integer	2	3	3	72				
05	Binary Switch (AC Switch) : bswitch.cob											05
IV	NAME	MEM	R/W	TYPE	SIZE	RL	WL	VALUE				
C	current_position	RAM	R/W	Boolean	1	3	3	False				

## APPENDIX E

### ALTERA SOURCE CODE

```
-- MAX+plus II Version 6.0 11/22/95
-- Modified: 12/01/97

TITLE "68HC11 Banked Memory Control with LCD support";

Subdesign phone
(
    data[7..0]                : bidir ;
    addr[15..0], iocs, rw     : Input;
    bank[2..0], dispena, ctload, voice : Output;
    dtmfcs, lc                : Output;
)

Variable
    bankreg[2..0], daa       : dff ;
    s, bnk, sdat[7..0], dispa, dispb : node ;
    dtmfa, dtmfb            :node;
    line                    :node;

BEGIN
    bnk = (addr[2]==H"1060" & !iocs) ;
    dispa = (addr[2]==H"1064" & !iocs) ;
    dispb = (addr[2]==H"1065" & !iocs) ;
    ctload = (addr[2]==H"1063" & !iocs);
    voice = (addr[2]==H"1062" & !iocs);
    dtmfa = (addr[2]==H"1066" & !iocs);
    dtmfb = (addr[2]==H"1067" & !iocs);
    line = (addr[2]==H"1068" & !iocs);

    s = bnk & rw;
    data0 = tri(sdat0,s) ;
    data1 = tri(sdat1,s) ;
    data2 = tri(sdat2,s) ;
    data3 = tri(sdat3,s) ;
    data4 = tri(sdat4,s) ;
    data5 = tri(sdat5,s) ;
    data6 = tri(sdat6,s) ;
    data7 = tri(sdat7,s) ;
```

```

    bankreg[2..0].d = data[2..0] ;
% ***** %
    if (addr[14]) THEN
        bank[2..0] = 7 ;
    ELSIF (!addr[14]) THEN
        bank[2..0] = bankreg[2..0].q ;
    end if;
%*****%
    if (bnk & rw) then
        sdat[2..0] = bankreg[2..0] ;
        sdat[7..3] = GND ;
    end if ;
%*****%
bankreg[].clk = !(bnk & !rw) ;
%*****%

% LCD Code ***** %
dispena = (dispa # dispb) ;

% Enable dtmf %
dtmfcs = (dtmfa # dtmfb);

% Enable or disable access to phone line %
daa.clk = !(line & !rw);
daa.d = data0;
lc = daa.q;

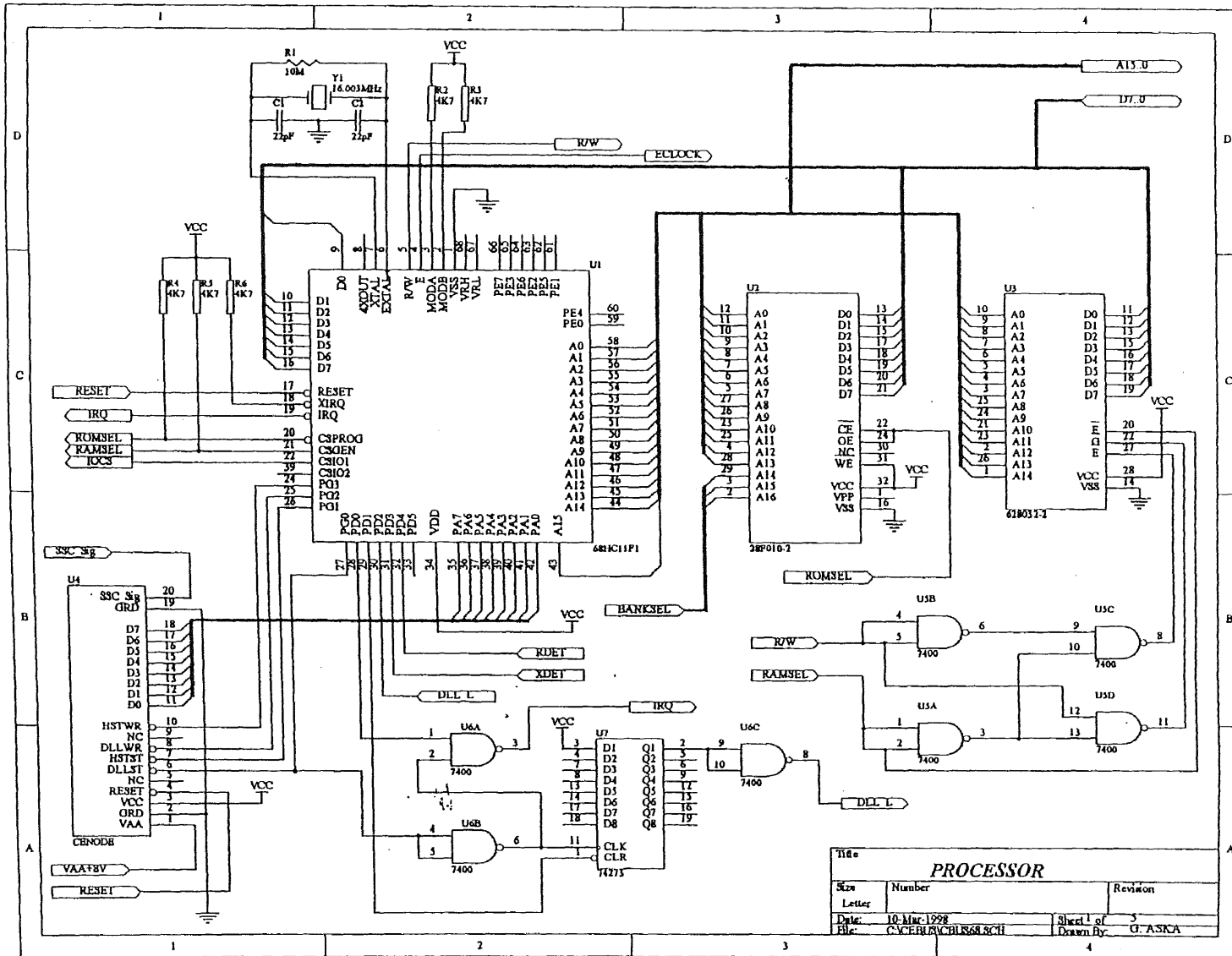
END;

```

## **APPENDIX F**

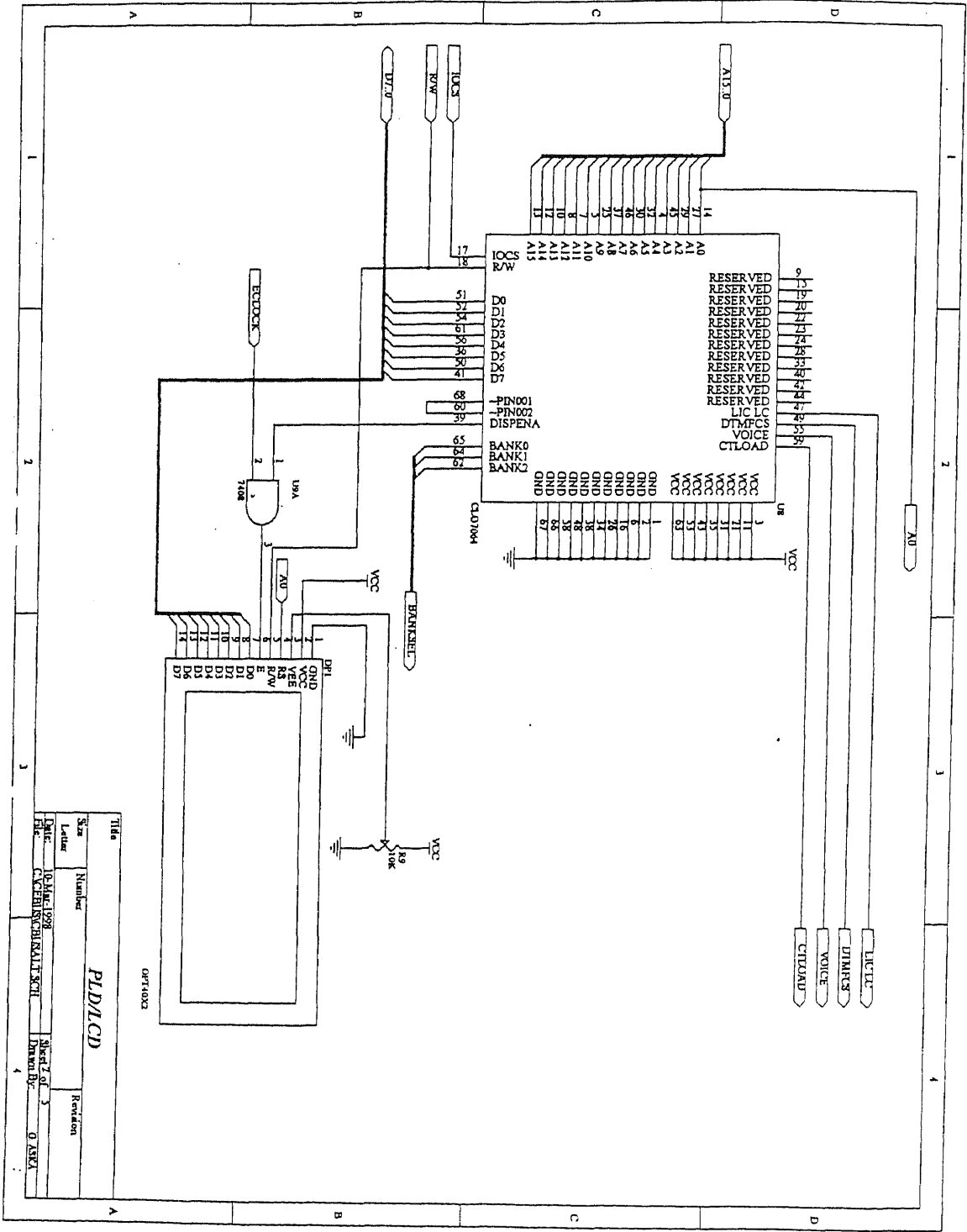
### **SCHEMATIC DIAGRAMS**

This Appendix contains all the schematic diagrams for the Controller. The Power Line Interface is on sheet 1 along with the MC68HC11 microcontroller, ROM and RAM chips. Sheet 2 shows the Altera PLD and the LCD display. The display was used extensively in the troubleshooting phase of the Controller's development. It was not necessary for it to be used when the Controller was fully functional. The Power Supply and Reset circuits are on sheet 3. Sheet 4 contains the Tone Detector circuit and Voice Module. The Voice Module (V8600) and associated logic are at the top of the sheet. The Tone Detector with its gain circuitry and logic devices are at the bottom of the sheet. The final sheet (sheet 5) shows the Telephone Line Interface in the upper half and the Ring Detector with associated logic in the lower half.

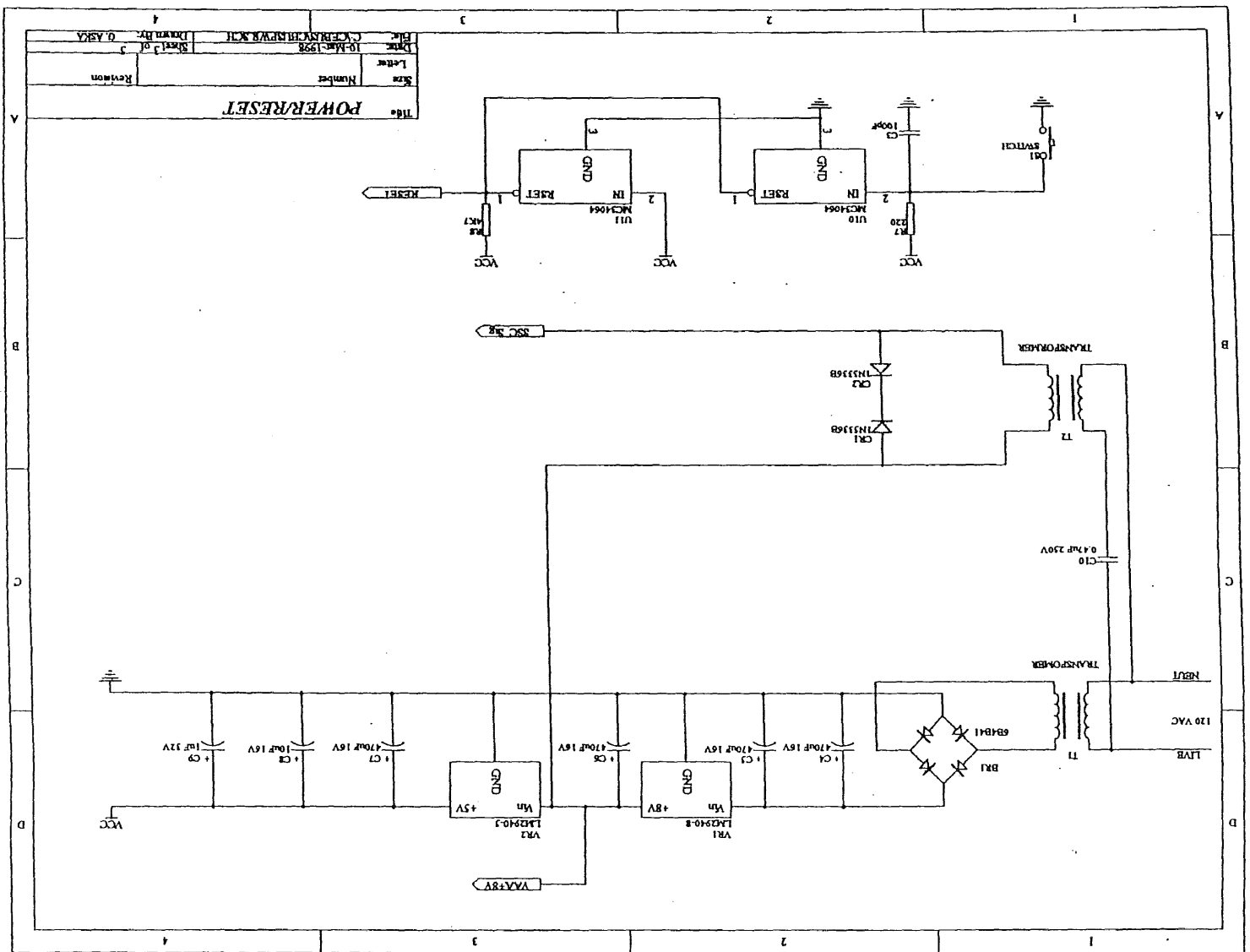


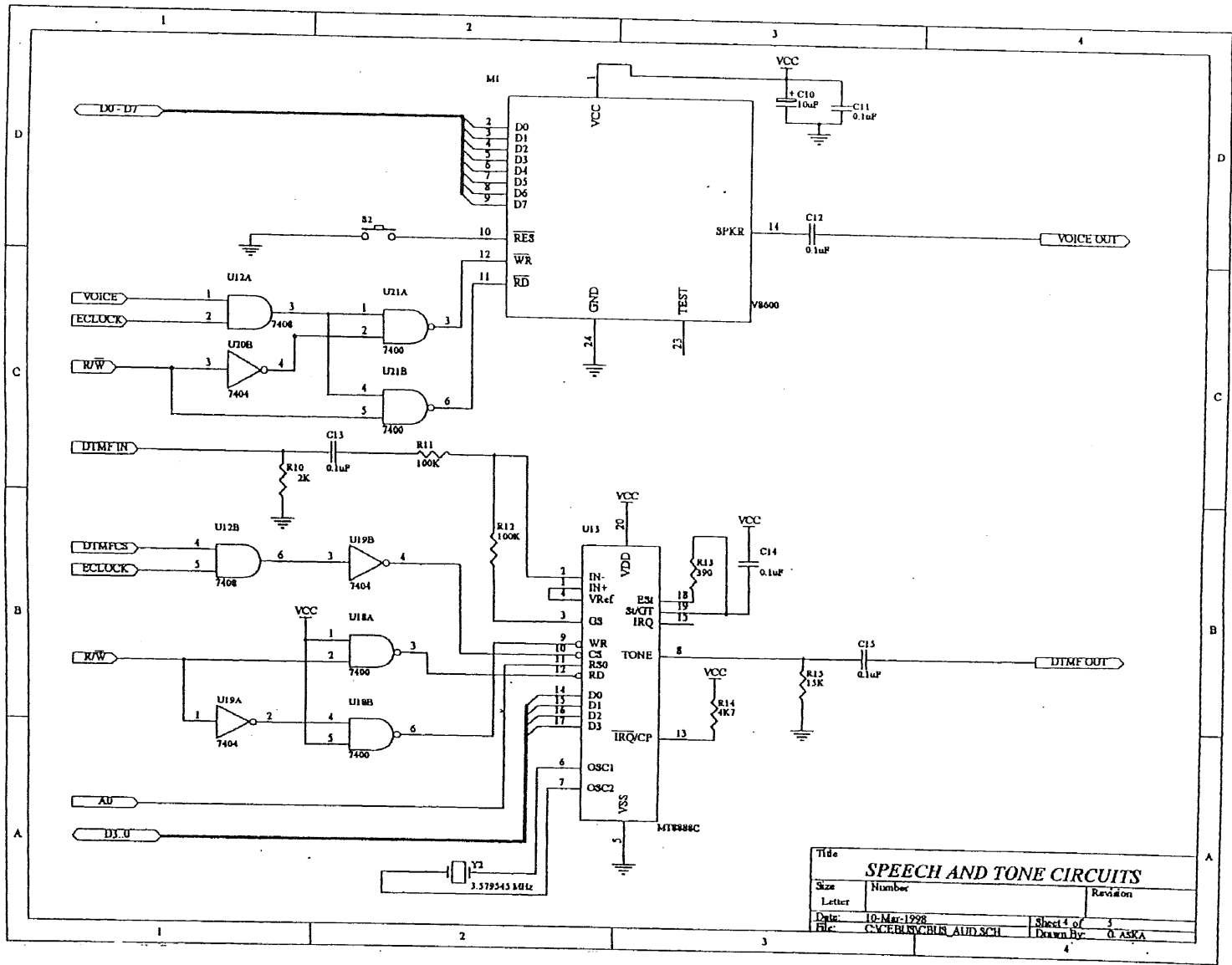
Title		
PROCESSOR		
Size	Number	Revision
Letter		
Date:	10-Mar-1998	Sheet 1 of 3
File:	C:\CEBU\CBUR68SCH	Drawn By: G. ASKA



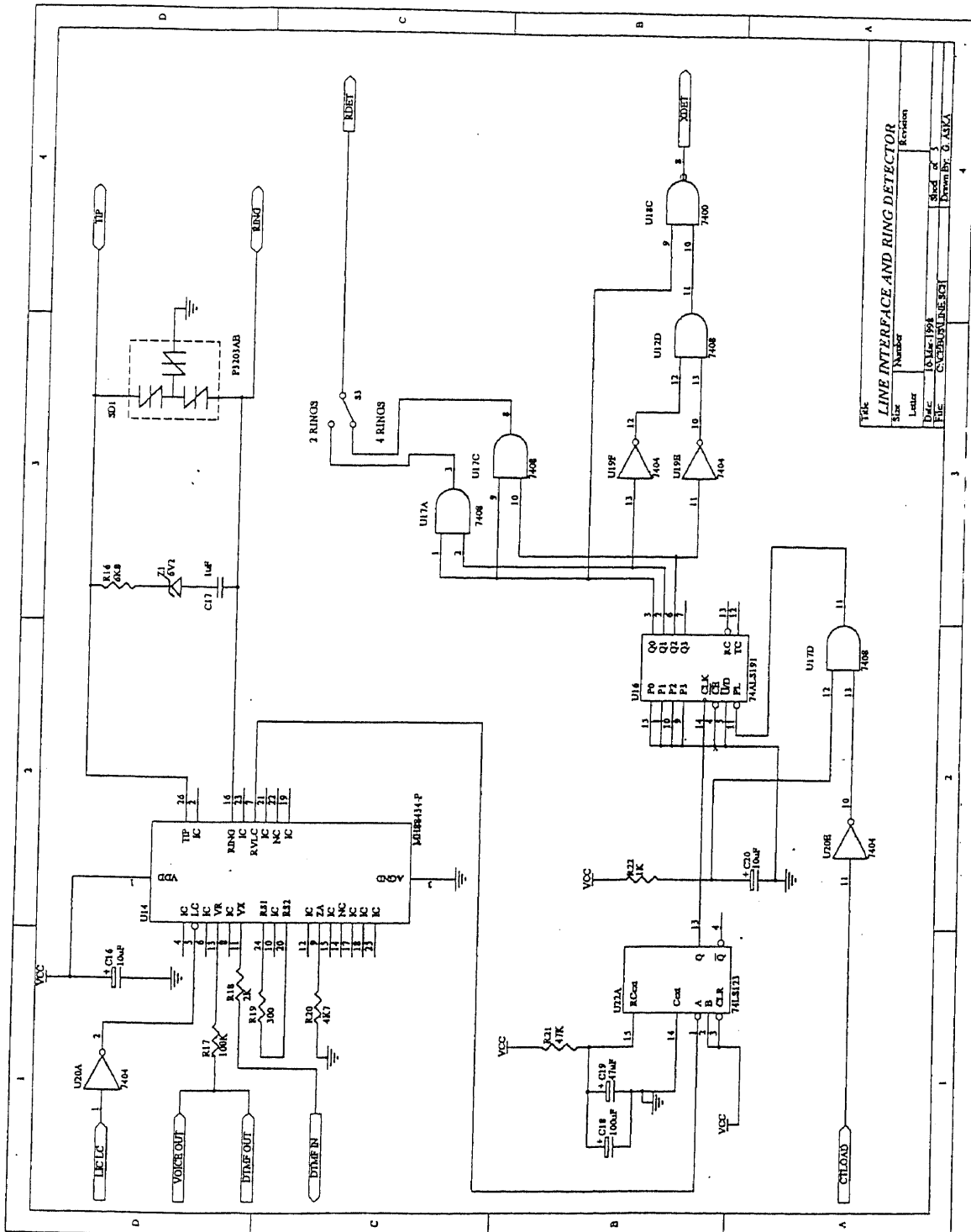


Title		PLD/CD	
Size	Number	Revision	
Label			
Date	10/16/1998	Sheet 2 of 3	
File	C:\EHEIN\CH\RA113\CH1	Drawn By	G. TAKI





Title		
<b>SPEECH AND TONE CIRCUITS</b>		
Size	Number	Revision
Letter		
Date:	10-Mar-1998	Sheet 4 of 5
File:	C:\CEBUS\CIBUS_AUD\SCH	Drawn By: G. ASKA



## APPENDIX G

### PARTS LIST

SHEET 1 (PROCESSOR)		
PART ID	PART NUMBER	DESCRIPTION
C1	22pF	Capacitor
C2	22pF	Capacitor
R1	10M	Resistor
R2	4k7	Resistor
R3	4k7	Resistor
R4	4k7	Resistor
R5	4k7	Resistor
R6	4k7	Resistor
U1	68HC11F1	Microcontroller
U2	28F010-2	128 Kbytes Flash Memory
U3	628032-2	32 Kbytes RAM
U4	SSC CIN PL01S-02	CEBus Power Line Network Interface
U5	7400	Quadruple 2-input NAND gates
U6	7400	Quadruple 2-input NAND gates
U7	74273	Octal D-type flip flop with clear
Y1	16.003MHz	Crystal

SHEET 2 (PLD/LCD)		
PART ID	PART NUMBER	DESCRIPTION
DP1	OPT40X2	LCD Display
R9	10K	Potentiometer
U8	CLO7064	Erasable Programmable Logic Device
U9	7408	Quadruple 2-input AND gate

SHEET 3(POWER/RESET)		
PART ID	PART NUMBER	DESCRIPTION
BR1	6B4B41	Bridge rectifier
T1	120VAC/12.6VDC	Transformer
T2	18:18 turns	Transformer
CR1	1N5336B	Diode
CR2	1N5336B	Diode
C4	470 $\mu$ F 16V	Capacitor
C5	470 $\mu$ F 16v	Capacitor
C6	470 $\mu$ F 16V	Capacitor
C7	470 $\mu$ F 16V	Capacitor
C8	10 $\mu$ F 16V	Capacitor
C9	1 $\mu$ F 35V	Capacitor
VR1	LM2940-8	8V 1amp low dropout voltage regulator
VR2	LM2940-5	5V 1amp low dropout voltage regulator
U10	MC34064	Microprocessor supervisor
U11	MC34064	Microprocessor supervisor
R7	220	Resistor
R8	4K7	Resistor
C3	100pF	Capacitor
S1	PBS100	Push button switch

SHEET 4 (SPEECH AND TONE CIRCUITS)		
PART ID	PART NUMBER	DESCRIPTION
C10	10 $\mu$ F	Capacitor
C11	0.1 $\mu$ F	Capacitor
C12	0.1 $\mu$ F	Capacitor
C13	0.1 $\mu$ F	Capacitor
C14	0.1 $\mu$ F	Capacitor
C15	0.1 $\mu$ F	Capacitor
M1	V8600	Speech Synthesizer
R10	2K	Resistor
R11	100K	Resistor
R12	100K	Resistor
R13	390	Resistor
R14	4K7	Resistor
R15	15K	Resistor
S1	PBS100	Push button switch
U12	7408	Quadruple 2-input AND gate
U13	MT8888CE	DTMF transceiver
U18	7400	Quadruple 2-input NAND gates
U19	7404	Hex inverter
U20	7404	Hex inverter
U21	7400	Quadruple 2-input NAND gates
Y2	3.579545 MHz	Crystal

SHEET 5 (LINE INTERFACE AND RING DETECTOR)		
PART ID	PART NUMBER	DESCRIPTION
U20	7404	Hex inverter
U17	7408	Quadruple 2-input AND gate
U19	7404	Hex inverter
U18	7400	Quadruple 2-input NAND gates
U14	MH88434-P	Data Access Arrangement
U22	74LS123	Dual monostable multivibrator with clear
R17	100K	Resistor
R18	2K	Resistor
R19	300	Resistor
R20	4K7	Resistor
R16	6K8	Resistor
R21	47K	Resistor
R22	1K	Resistor
C16	10 $\mu$ F	Capacitor
C17	1 $\mu$ F	Capacitor
C18	100 $\mu$ F	Capacitor
C19	47 $\mu$ F	Capacitor
C20	10 $\mu$ F	Capacitor
S3	SW153	SPDT switch



## REFERENCES

1. *Analog/Digital Telecom Components*, Mitel Semiconductor, Kanata, Canada, 1997.
2. *CEBench User's Manual*, Intellon Corp., Ocala, FL, 1996.
3. *CEBus Power Line Encoding and Signaling*, Intellon, Corp., Ocala, FL, 1997.
4. *CEBus Power Line Network Interface Technical Data Sheet*, Intellon, Corp., Ocala, FL, 1996.
5. G. Evans, *CEBus Standard User's Guide*. Tualatin, OR: Training Department, 1996.
6. G. Held, *Data Communications Networking Devices*, New York: Wiley, 1994.
7. H. M. Deitel and P.J. Deitel, *C, How to Program*, Englewood Cliffs, NJ: Prentice Hall, 1992
8. *M68HC11 Reference Manual*, Motorola, Schaumburg, IL, 1991.
9. *M68HC11 Technical Data*, Motorola, Schaumburg, IL, 1997.
10. *MAX+PLUS II Programmable Logic Development System*, Altera Corp., San Jose, CA, 1994.
11. R. L. Freeman, *Telecommunication System Engineering*, New York: Wiley, 1996.
12. *68HC11 Assembler, Linker, and Librarian Programming, User Guide*, IAR Systems, San Francisco, CA, 1995.
13. *68HC11 C Compiler Programming Guide*, IAR Systems, San Francisco, CA, 1995.
14. *68HC11 Command Line Interface Guide*, IAR Systems, San Francisco, CA, 1995.
15. *68HC11 UI User Interface*, IAR Systems, San Francisco, CA, 1994.
16. S. J. Bigelow, *Understanding Telephone Electronics*, Indianapolis, Indiana: SAMS Publishing, 1994.
17. *TTL Logic Data Book*, Texas Instrument, Dallas, TX, 1988.
18. *V8600/1 Speech Synthesizers Data Book*, RC Systems, Bothell, WA, 1991.