Summer 1998

Telephone-accessed controller using CEBus for device control over power line

Gerald Aska
New Jersey Institute of Technology

Follow this and additional works at: https://digitalcommons.njit.edu/theses
Part of the Electrical and Electronics Commons

Recommended Citation
https://digitalcommons.njit.edu/theses/898

This Thesis is brought to you for free and open access by the Theses and Dissertations at Digital Commons @ NJIT. It has been accepted for inclusion in Theses by an authorized administrator of Digital Commons @ NJIT. For more information, please contact digitalcommons@njit.edu.
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-ACCESSSED 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 responses 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-ACCESSSED 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
TELEPHONE-ACCESSSED CONTROLLER USING CEBUS FOR DEVICE CONTROL OVER POWER LINE

by

Gerald Aska

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

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

Dr. Yun-Qing Shi, Committee Member
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

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 INTRODUCTION</td>
<td>..........................................................1</td>
</tr>
<tr>
<td>1.1 Objective</td>
<td>..........................................................1</td>
</tr>
<tr>
<td>1.2 Background Information</td>
<td>.........................................................2</td>
</tr>
<tr>
<td>2 SPREAD SPECTRUM</td>
<td>.......................................................4</td>
</tr>
<tr>
<td>3 HARDWARE DESCRIPTION</td>
<td>.....................................................10</td>
</tr>
<tr>
<td>3.1 Power Line Interface</td>
<td>....................................................12</td>
</tr>
<tr>
<td>3.2 Telephone Line Interface</td>
<td>..................................................18</td>
</tr>
<tr>
<td>3.3 Ring Detector</td>
<td>........................................................23</td>
</tr>
<tr>
<td>3.4 Tone Detector</td>
<td>..........................................................29</td>
</tr>
<tr>
<td>3.5 Voice Module</td>
<td>..........................................................34</td>
</tr>
<tr>
<td>4 SOFTWARE IMPLEMENTATION</td>
<td>.....................................................42</td>
</tr>
<tr>
<td>4.1 Bank Memory Implementation</td>
<td>.................................................42</td>
</tr>
<tr>
<td>4.2 Software Development</td>
<td>........................................................49</td>
</tr>
<tr>
<td>4.3 CEBus Message Transfer using CEBench</td>
<td>........................................52</td>
</tr>
<tr>
<td>4.3.1 Layers Definitions</td>
<td>........................................................52</td>
</tr>
<tr>
<td>4.3.2 CEBench Modifications</td>
<td>..................................................55</td>
</tr>
<tr>
<td>5 USER INTERFACE</td>
<td>..........................................................60</td>
</tr>
<tr>
<td>6 CONCLUSION AND FUTURE IMPROVEMENTS</td>
<td>.......................................65</td>
</tr>
<tr>
<td>APPENDIX A CONTROLLER APPLICATION SOURCE CODE</td>
<td>....................................68</td>
</tr>
<tr>
<td>APPENDIX B CONTROLLER APPLICATION FLOWCHARTS</td>
<td>................................103</td>
</tr>
<tr>
<td>Chapter</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>APPENDIX C COMPIL DE BATCH FILE</td>
<td>113</td>
</tr>
<tr>
<td>APPENDIX D CONTEXTS, OBJECTS AND IVS</td>
<td>114</td>
</tr>
<tr>
<td>APPENDIX E ALTERA SOURCE CODE</td>
<td>120</td>
</tr>
<tr>
<td>APPENDIX F SCHEMATIC DIAGRAMS</td>
<td>122</td>
</tr>
<tr>
<td>APPENDIX G PARTS LIST</td>
<td>128</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>132</td>
</tr>
</tbody>
</table>
## LIST OF TABLES

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

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Spread spectrum signal for a 1 bit</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Representation of two consecutive 1 bits</td>
<td>7</td>
</tr>
<tr>
<td>2.3</td>
<td>Bit representation for the preamble</td>
<td>8</td>
</tr>
<tr>
<td>2.4</td>
<td>Preamble and preamble EOF signaling</td>
<td>8</td>
</tr>
<tr>
<td>2.5</td>
<td>Beginning of data using φ1 and φ2 for symbols</td>
<td>9</td>
</tr>
<tr>
<td>3.1</td>
<td>High-level block diagram of Controller</td>
<td>10</td>
</tr>
<tr>
<td>3.2</td>
<td>Second-level block diagram of Controller</td>
<td>11</td>
</tr>
<tr>
<td>3.3</td>
<td>CENode PL block diagram</td>
<td>13</td>
</tr>
<tr>
<td>3.4</td>
<td>Timing diagram of Attention Sequence</td>
<td>17</td>
</tr>
<tr>
<td>3.5</td>
<td>Timing diagram of read Command Sequence</td>
<td>17</td>
</tr>
<tr>
<td>3.6</td>
<td>Timing diagram of write Command Sequence</td>
<td>18</td>
</tr>
<tr>
<td>3.7</td>
<td>Ringing cadence at Tip and Ring.</td>
<td>21</td>
</tr>
<tr>
<td>3.8</td>
<td>Ringing cadence at output of DAA.</td>
<td>21</td>
</tr>
<tr>
<td>3.9</td>
<td>Telephone Line Interface Circuit</td>
<td>22</td>
</tr>
<tr>
<td>3.10</td>
<td>Multivibrator used in Ring Detector Circuit</td>
<td>24</td>
</tr>
<tr>
<td>3.11</td>
<td>Counter circuit used in Ring Detector Circuit</td>
<td>26</td>
</tr>
<tr>
<td>3.12</td>
<td>Timing relation of Multivibrator and ring cadence of DAA</td>
<td>27</td>
</tr>
<tr>
<td>3.13</td>
<td>Tone Detector Circuit</td>
<td>30</td>
</tr>
<tr>
<td>3.14</td>
<td>V8600 logic symbol</td>
<td>35</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.15</td>
<td>V8600 status flags</td>
<td>36</td>
</tr>
<tr>
<td>3.16</td>
<td>Microprocessor interface example</td>
<td>36</td>
</tr>
<tr>
<td>3.17</td>
<td>Parallel Printer Port interface example</td>
<td>37</td>
</tr>
<tr>
<td>3.19</td>
<td>RS-232 Serial interface example</td>
<td>37</td>
</tr>
<tr>
<td>4.1</td>
<td>Altera bank memory implementation program (part 1)</td>
<td>44</td>
</tr>
<tr>
<td>4.1</td>
<td>Altera bank memory implementation program (part 2)</td>
<td>45</td>
</tr>
<tr>
<td>4.2</td>
<td>Address range of each bank in memory chip</td>
<td>46</td>
</tr>
<tr>
<td>4.3</td>
<td>Memory map of Controller (part 1)</td>
<td>47</td>
</tr>
<tr>
<td>4.3</td>
<td>Memory map of Controller (part 2)</td>
<td>48</td>
</tr>
<tr>
<td>4.4</td>
<td>Hierarchical arrangement of software program</td>
<td>50</td>
</tr>
<tr>
<td>4.5</td>
<td>CEBench components of CEBus device</td>
<td>53</td>
</tr>
<tr>
<td>4.6</td>
<td>Contexts, Objects and IVs used for the Controller (part 1)</td>
<td>56</td>
</tr>
<tr>
<td>4.6</td>
<td>Contexts, Objects and IVs used for the Controller (part 2)</td>
<td>57</td>
</tr>
<tr>
<td>5.1</td>
<td>Operation flowchart of Controller (part 1)</td>
<td>63</td>
</tr>
<tr>
<td>5.1</td>
<td>Operation flowchart of Controller (part 2)</td>
<td>64</td>
</tr>
</tbody>
</table>
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 develop 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 100μsec. 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 must back off. This detection method is called carrier sense multiple access (CSMA). If, after all the eight bits of the preamble have 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

Figure 2.1 Spread spectrum signal for a 1 bit.
received the whole packet. However, the packet would eventually be discard 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 of Frame (EOF), and End of Packet (EOP) for different parts of a packet. A 100μs 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 100μs. This inverted symbol is called a INFERIOR or SUPERIOR φ2 (phase 2) state and the non-inverted SUPERIOR state is called the SUPERIOR φ1 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 100μs 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 200μs. 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.

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PREMBLE</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UST</td>
<td>Timing</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>114μs</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>228μs</td>
</tr>
<tr>
<td>EOF</td>
<td>8</td>
<td>800μs</td>
</tr>
<tr>
<td>EOP</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
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 100μs 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 φ1 and φ2 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 responses 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.

![CENode PL block diagram.](image)

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>
<thead>
<tr>
<th>CETHIXN FUNCTIONS</th>
<th>TRANSMITTING MODE</th>
<th>RECEIVING MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate preamble</td>
<td>Separate packet fragments</td>
<td></td>
</tr>
<tr>
<td>Compress data using leading zero suppression</td>
<td>Detect packet errors</td>
<td></td>
</tr>
<tr>
<td>Generate FCS for error checking</td>
<td>Compare received and generated FCS</td>
<td></td>
</tr>
<tr>
<td>Generate acknowledgement packet</td>
<td>Recognize System and MAC addresses</td>
<td></td>
</tr>
<tr>
<td>Access the power line medium using the CSMA/CDCR protocol</td>
<td>Reject duplicate packets</td>
<td></td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Command Value</th>
<th>Command Code</th>
<th>Command Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00H</td>
<td>FIE</td>
<td>Force Interface Error</td>
<td>Used by Factory Testing ONLY.</td>
</tr>
<tr>
<td>01H</td>
<td>RST</td>
<td>Reset</td>
<td>Host interface reset of CENode.</td>
</tr>
<tr>
<td>02H</td>
<td>LR</td>
<td>Layer Management read</td>
<td>Layer management read of configuration information.</td>
</tr>
<tr>
<td>03H</td>
<td>LW</td>
<td>Layer Management write</td>
<td>Layer management write of configuration information.</td>
</tr>
<tr>
<td>04H</td>
<td>IR</td>
<td>Interface Read</td>
<td>Host read of interface flags.</td>
</tr>
<tr>
<td>05H</td>
<td>CW</td>
<td>Control Wrote</td>
<td>Host write to data link control parameters.</td>
</tr>
<tr>
<td>06H</td>
<td>SR</td>
<td>Status Read</td>
<td>Host read of transmit, receive and statistics status information.</td>
</tr>
<tr>
<td>07H</td>
<td>--</td>
<td>Reserved</td>
<td>Reserved.</td>
</tr>
<tr>
<td>08H</td>
<td>PR</td>
<td>Packet Receive</td>
<td>Host read of received CENode packet.</td>
</tr>
<tr>
<td>09H</td>
<td>PT</td>
<td>Packet Transmit</td>
<td>Host write of CENode packet.</td>
</tr>
<tr>
<td>0AH</td>
<td>RRI</td>
<td>Read Receive Information</td>
<td>Host read of buffered received packet NDPU field from CENode.</td>
</tr>
<tr>
<td>0BH</td>
<td>WTI</td>
<td>Write Transmit Invoke</td>
<td>Host write of transmit NDPU field to CENode.</td>
</tr>
<tr>
<td>0CH</td>
<td>RRH</td>
<td>Read Receive Header</td>
<td>Host read of data link NPDU header information from received.</td>
</tr>
<tr>
<td>0DH</td>
<td>WTH</td>
<td>Write Transmit Header</td>
<td>Host write of the NDPU header information for subsequent transmit packets.</td>
</tr>
<tr>
<td>0EH-45H</td>
<td>--</td>
<td>Reserved</td>
<td>Reserved.</td>
</tr>
<tr>
<td>46H</td>
<td>WR46</td>
<td>Write Register 4</td>
<td>Host write of data link communications access control parameters.</td>
</tr>
<tr>
<td>47H-83H</td>
<td>--</td>
<td>Reserved</td>
<td>Reserved.</td>
</tr>
<tr>
<td>84H</td>
<td>RR4</td>
<td>Read Register 4</td>
<td>Host read of receive status flags.</td>
</tr>
<tr>
<td>85H-FFH</td>
<td>--</td>
<td>Reserved</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0-D7</td>
<td>Data pins</td>
<td>Bi-directional data lines. In high impedance state when not in use for transfer.</td>
</tr>
<tr>
<td>HSTST*</td>
<td>Host Strobe</td>
<td>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.</td>
</tr>
<tr>
<td>HSTWR*</td>
<td>Host Write</td>
<td>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.</td>
</tr>
<tr>
<td>DLLST*</td>
<td>CENode Strobe</td>
<td>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.</td>
</tr>
<tr>
<td>DLLWR*</td>
<td>CENode Write</td>
<td>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.</td>
</tr>
<tr>
<td>Reset</td>
<td>CENode Reset</td>
<td>Reset signal line for CENode. The host may assert this line low (open collector drive) to provide a hardware reset to the CENode.</td>
</tr>
</tbody>
</table>
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 6μsec 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 15μsec 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 15μsec, 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 15μsec, 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 5µsec 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].

![Timing diagram of Attention Sequence.](image)

**Figure 3.4** Timing diagram of Attention Sequence.

![Timing diagram of read Command Sequence.](image)

**Figure 3.5** Timing diagram of read 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

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

*dBSpl = dB sound pressure level.

**dBrnC = 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 \( Z_{\text{ext}} \):

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

Where \( Z_{\text{int}} \) is the internal impedance of the DAA and is equal to 1.3k\( \Omega \). \( Z_{\text{in}} \) (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 20Vrms. The ring signal from the central office is typically 90Vrms. 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 R3 and R4 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 2kΩ 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 47kΩ. A 100kΩ 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 400Ω 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:

\[ t_w = K \times R1 \times C1 \]  

(3.2)

When C1 is greater than or equal to 1μF, the output pulse duration is calculated from:

\[ t_w = 0.33 \times R1 \times C1 \]  

(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 KΩ

C1 is in pF

tw 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 late. 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.

<table>
<thead>
<tr>
<th>INPUTS</th>
<th>OUTPUTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR</td>
<td>A B</td>
</tr>
<tr>
<td>L</td>
<td>X X</td>
</tr>
<tr>
<td>X</td>
<td>H X</td>
</tr>
<tr>
<td>X</td>
<td>X L</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>H</td>
<td></td>
</tr>
<tr>
<td></td>
<td>L H</td>
</tr>
</tbody>
</table>
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

![Counter Circuit Diagram]

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.

![Diagram](image)

**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.

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

### Table 3.7  Functional Encode/Decode Table.

<table>
<thead>
<tr>
<th>LOW FREQ.</th>
<th>HIGH FREQ.</th>
<th>DIGIT</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>697</td>
<td>1209</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>697</td>
<td>1336</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>697</td>
<td>1477</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>770</td>
<td>1209</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>770</td>
<td>1336</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>770</td>
<td>1477</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>852</td>
<td>1209</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>852</td>
<td>1336</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>852</td>
<td>1477</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>941</td>
<td>1336</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>941</td>
<td>1209</td>
<td>*</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>941</td>
<td>1477</td>
<td>#</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>697</td>
<td>1633</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>770</td>
<td>1633</td>
<td>B</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>852</td>
<td>1633</td>
<td>C</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>941</td>
<td>1633</td>
<td>D</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
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 ms±1 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>
<thead>
<tr>
<th>RS0</th>
<th>WR*</th>
<th>RD*</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Write to Transmit Data Register</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Read from Receive Data Register</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Write to Control Register</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Read from Status Register</td>
</tr>
</tbody>
</table>

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]:

1. Read Status Register
   - RS0 = 1
   - WR* = 1
   - RD* = 0
   - D3 = X
   - D2 = X
   - D1 = X
   - D0 = X

2. Write to Control Register
   - RS0 = 1
   - WR* = 0
   - RD* = 1
   - D3 = 0
   - D2 = 0
   - D1 = 0
   - D0 = 0
3. Write to Control Register  1  0  1  0  0  0  0  0
4. Write to Control Register  1  0  1  1  0  0  0  0
5. Write to Control Register  1  0  1  0  0  0  0  0
6. Read Status Register      1  1  0  X  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:

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

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 x 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 x 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.

![V8600 logic symbol](image)

**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.

<table>
<thead>
<tr>
<th></th>
<th>SYNC</th>
<th>SYNC2</th>
<th>RDY</th>
<th>AF</th>
<th>AE</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>

**Figure 3.15** V8600 status flags.

![Microprocessor interface example diagram](image)

**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...
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|Y K T UW \M IY DH AE T -\W EY .+/");

<table>
<thead>
<tr>
<th>Phoneme Symbol</th>
<th>Example Word</th>
<th>Phoneme Symbol</th>
<th>Example Word</th>
<th>Phoneme Symbol</th>
<th>Example Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>father</td>
<td>H</td>
<td>hire</td>
<td>SH</td>
<td>Dish</td>
</tr>
<tr>
<td>AE</td>
<td>bat</td>
<td>IH</td>
<td>sit</td>
<td>T</td>
<td>Tip</td>
</tr>
<tr>
<td>AH</td>
<td>cut</td>
<td>IX</td>
<td>relative</td>
<td>TH</td>
<td>Thick</td>
</tr>
<tr>
<td>AO</td>
<td>lawn</td>
<td>IY</td>
<td>meet</td>
<td>TX</td>
<td>Mistake</td>
</tr>
<tr>
<td>AW</td>
<td>cow</td>
<td>JH</td>
<td>jet</td>
<td>UH</td>
<td>Pull</td>
</tr>
<tr>
<td>AX</td>
<td>about</td>
<td>K</td>
<td>cute</td>
<td>UW</td>
<td>Tool</td>
</tr>
<tr>
<td>AY</td>
<td>kite</td>
<td>KX</td>
<td>ski</td>
<td>V</td>
<td>Give</td>
</tr>
<tr>
<td>B</td>
<td>bird</td>
<td>L</td>
<td>long</td>
<td>W</td>
<td>Went</td>
</tr>
<tr>
<td>CH</td>
<td>cheese</td>
<td>M</td>
<td>mug</td>
<td>WH</td>
<td>When</td>
</tr>
<tr>
<td>D</td>
<td>dare</td>
<td>N</td>
<td>new</td>
<td>YY</td>
<td>You</td>
</tr>
<tr>
<td>DH</td>
<td>either</td>
<td>NX</td>
<td>rung</td>
<td>Z</td>
<td>Zero</td>
</tr>
<tr>
<td>DX</td>
<td>computer</td>
<td>OW</td>
<td>tone</td>
<td>ZH</td>
<td>Leisure</td>
</tr>
<tr>
<td>EH</td>
<td>set</td>
<td>OY</td>
<td>boy</td>
<td>space</td>
<td>variable</td>
</tr>
<tr>
<td>ER</td>
<td>were</td>
<td>P</td>
<td>past</td>
<td>pause</td>
<td>pause</td>
</tr>
<tr>
<td>EY</td>
<td>bake</td>
<td>PX</td>
<td>spot</td>
<td>medium</td>
<td>pause</td>
</tr>
<tr>
<td>F</td>
<td>fact</td>
<td>R</td>
<td>ring</td>
<td>long pause</td>
<td>pause</td>
</tr>
<tr>
<td>G</td>
<td>give</td>
<td>S</td>
<td>some</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 0.1μF 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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
<th>Range n</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>NB</td>
<td>Punctuation level</td>
<td>0 - 7</td>
<td>6</td>
</tr>
<tr>
<td>NC</td>
<td>Character mode/delay</td>
<td>0 - 31</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>Phoneme mode</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>Enable intonation</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>NF</td>
<td>Format frequency</td>
<td>0 - 9</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>Tone generator</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>Load exception dictionary</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>Disable intonation</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>NP</td>
<td>Pitch</td>
<td>0 - 99</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>Clear</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>NS</td>
<td>Speed</td>
<td>0 - 9</td>
<td></td>
</tr>
<tr>
<td>NT</td>
<td>Text mode/delay</td>
<td>0 - 15</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>Enable exception dictionary</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>NV</td>
<td>Volume</td>
<td>0 - 9</td>
<td></td>
</tr>
<tr>
<td>NX</td>
<td>Tone</td>
<td>0 - 1</td>
<td></td>
</tr>
<tr>
<td>NY</td>
<td>Timeout delay</td>
<td>0 - 15</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>Zap command</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>@</td>
<td>Reinitialize</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>n#</td>
<td>PCM mode</td>
<td>0 - 99</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 4
SOFTWARE IMPLEMENTAION

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 109.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

<table>
<thead>
<tr>
<th>Address</th>
<th>Bank 0</th>
<th>Bank 1</th>
<th>Bank 2</th>
<th>Bank 3</th>
<th>Bank 4</th>
<th>Bank 5</th>
<th>Bank 6</th>
<th>Non-Banked</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03FFFFH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>04000H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07FFFFH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08000H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0BFFFFH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0C000H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0FFFFH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13FFFFH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14000H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17FFFFH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18000H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1BFFFFH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1C000H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1FFFFH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4.2** Address range of each bank in memory chip.
<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000H</td>
<td>INTERNAL MCU RAM</td>
</tr>
<tr>
<td>03FFH</td>
<td></td>
</tr>
<tr>
<td>0400H</td>
<td></td>
</tr>
<tr>
<td>0FFFFH</td>
<td>EXTERNAL RAM</td>
</tr>
<tr>
<td>1000H</td>
<td>INTERNAL REGISTERS</td>
</tr>
<tr>
<td>105FH</td>
<td>BANK MEMORY CONTROL REGISTER</td>
</tr>
<tr>
<td>1061H</td>
<td>NOT USED</td>
</tr>
<tr>
<td>1062H</td>
<td>SPEECH SYNTHESIZER</td>
</tr>
<tr>
<td>1063H</td>
<td>COUNTER RESET</td>
</tr>
<tr>
<td>1064H</td>
<td>LCD OPERATION REGISTER</td>
</tr>
<tr>
<td>1065H</td>
<td>LCD DATA REGISTER</td>
</tr>
<tr>
<td>1066H</td>
<td>DTMF DATA REGISTER</td>
</tr>
<tr>
<td>1067H</td>
<td>DTMF CONTROL &amp; STATUS REGISTERS</td>
</tr>
<tr>
<td>1068H</td>
<td>DAA</td>
</tr>
</tbody>
</table>

**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 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.

**Figure 4.3** Memory map of Controller (part 2).
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 1FFFFF) 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 outlined 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

<table>
<thead>
<tr>
<th>Control Module</th>
<th>Data Sent</th>
<th>Controlled Module</th>
<th>Data Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>controller</td>
<td>Void</td>
<td>Local Control</td>
<td>0 = Not finished 1 = Finished</td>
</tr>
<tr>
<td>controller</td>
<td>Void</td>
<td>Remote Access Control</td>
<td>0 = Not finished 1 = Finished</td>
</tr>
<tr>
<td>controller</td>
<td>Void</td>
<td>Remote Command Control</td>
<td>0 = Not finished 1 = Finished</td>
</tr>
<tr>
<td>controller</td>
<td>Void</td>
<td>Line Detector</td>
<td>0 = Line not active 1 = Local access request 2 = Remote access request</td>
</tr>
<tr>
<td>controller</td>
<td>Void</td>
<td>Line Release and Counter Reset</td>
<td>Void</td>
</tr>
<tr>
<td>Local Control</td>
<td>Void</td>
<td>DTMF Read</td>
<td>Digit</td>
</tr>
<tr>
<td>Local Control</td>
<td>Command code</td>
<td>Command Code Check</td>
<td>0 = Incorrect 1 = Correct</td>
</tr>
<tr>
<td>Local Control</td>
<td>Void</td>
<td>DTMF Status Check</td>
<td>0 = Not ready 1 = Ready 2 = Not-ready counter at max.</td>
</tr>
<tr>
<td>Remote Access Control</td>
<td>Void</td>
<td>DTMF Read</td>
<td>Digit</td>
</tr>
<tr>
<td>Remote Access Control</td>
<td>Access code</td>
<td>Access Code Check</td>
<td>0 = Incorrect 1 = Correct</td>
</tr>
<tr>
<td>Remote Access Control</td>
<td>Void</td>
<td>DTMF Status Check</td>
<td>0 = Not ready 1 = Ready 2 = Not-ready counter at max.</td>
</tr>
<tr>
<td>Remote Access Control</td>
<td>0 = Reset counter 1 = Increment counter</td>
<td>Access Code Counter</td>
<td>0 = Not maximum count 1 = Maximum count</td>
</tr>
<tr>
<td>Command Code Check</td>
<td>Command code</td>
<td>Command Activation</td>
<td>0 = Code out or range for addressed device 1 = Code accepted</td>
</tr>
<tr>
<td>DTMF Status Check</td>
<td>0 = Reset counter 1 = Increment counter</td>
<td>DTMF Ready Counter</td>
<td>0 = Not maximum count 1 = Maximum count</td>
</tr>
</tbody>
</table>
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
<table>
<thead>
<tr>
<th>#</th>
<th>CONTEXT</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>Universal (0) : univ.cxl</td>
<td>00</td>
</tr>
<tr>
<td>01</td>
<td>Node Control (Device Control) : nodectrl.cob</td>
<td>01</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>serial_#</td>
<td>ROM</td>
<td>R</td>
<td>String</td>
<td>21</td>
<td>3</td>
<td>3</td>
<td>GA1198</td>
</tr>
<tr>
<td>n</td>
<td>manuf_name</td>
<td>ROM</td>
<td>R</td>
<td>String</td>
<td>21</td>
<td>3</td>
<td>3</td>
<td>GA CEBus Solutions</td>
</tr>
<tr>
<td>m</td>
<td>manuf_model</td>
<td>ROM</td>
<td>R</td>
<td>String</td>
<td>21</td>
<td>3</td>
<td>3</td>
<td>CON997</td>
</tr>
<tr>
<td>c</td>
<td>product_class</td>
<td>ROM</td>
<td>R</td>
<td>String</td>
<td>21</td>
<td>3</td>
<td>3</td>
<td>UNLISTED</td>
</tr>
<tr>
<td>h</td>
<td>system_addr</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x2</td>
<td>3</td>
<td>3</td>
<td>00 03</td>
</tr>
<tr>
<td>a</td>
<td>mac_addr</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x2</td>
<td>3</td>
<td>3</td>
<td>00 01</td>
</tr>
</tbody>
</table>

| 02 | Context Control (Context Control) : cntxctrl.cob | 02 |

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>o</td>
<td>object_list</td>
<td>ROM</td>
<td>R</td>
<td>Data</td>
<td>4x2</td>
<td>3</td>
<td>3</td>
<td>a0 00 a0 21 a0 74 a1 74</td>
</tr>
</tbody>
</table>

| 03 | Data Memory (Event Manager) : datamem.cob | 16 |

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>o</td>
<td>object_list</td>
<td>ROM</td>
<td>R</td>
<td>Data</td>
<td>3x2</td>
<td>3</td>
<td>3</td>
<td>01 01 02 02 16 03</td>
</tr>
</tbody>
</table>

| 06 | Binary Sensor (Light Switch) : bsensor.cob | 06 |

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_state</td>
<td>RAM</td>
<td>R/W</td>
<td>Boolean</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>False</td>
</tr>
<tr>
<td>R</td>
<td>report_condition</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x4</td>
<td>3</td>
<td>3</td>
<td>43 00</td>
</tr>
<tr>
<td>H</td>
<td>report_header</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x6</td>
<td>3</td>
<td>3</td>
<td>21 06 45 43 55</td>
</tr>
<tr>
<td>A</td>
<td>dest_address</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x4</td>
<td>3</td>
<td>3</td>
<td>00 02 00 01</td>
</tr>
</tbody>
</table>

| 07 | Binary Sensor (Light Intermediate Switch) : bsensor.cob | I/O # (2) | 06 |

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_state</td>
<td>RAM</td>
<td>R</td>
<td>Boolean</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>False</td>
</tr>
</tbody>
</table>

| 08 | Binary Sensor (Ack) : bsensor.cob | I/O # (3) | 06 |

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_state</td>
<td>RAM</td>
<td>R/W</td>
<td>Boolean</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>False</td>
</tr>
</tbody>
</table>

---

**Figure 4.6** Contexts, Objects and IVs used for the Controller (part 1).
### CEBus Remote Access Controller

<table>
<thead>
<tr>
<th>ID</th>
<th>CONTEXT</th>
<th>OBJECT</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>74</td>
<td>Refrigerator/Freezer (Heater) : refrigfr.cxl</td>
<td>01</td>
<td>Context Control (Context Control) : cntxctrl.cob</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>object_list</td>
<td>ROM</td>
<td>R</td>
<td>Data</td>
<td>7x2</td>
<td>1</td>
<td>1</td>
<td>02 01 07 02 05 05 06 07 08 08 06 09 08 04</td>
</tr>
</tbody>
</table>

| 07 | Analog Control (Temperature Setting) : analctrl.cob |

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>min_value</td>
<td>ROM</td>
<td>R</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>65</td>
</tr>
<tr>
<td>M</td>
<td>max_value</td>
<td>ROM</td>
<td>R</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>85</td>
</tr>
<tr>
<td>D</td>
<td>default_value</td>
<td>ROM</td>
<td>R</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>72</td>
</tr>
<tr>
<td>C</td>
<td>current_value</td>
<td>RAM</td>
<td>R/W</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>R</td>
<td>report_condition</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x4</td>
<td>3</td>
<td>3</td>
<td>43 31</td>
</tr>
<tr>
<td>H</td>
<td>report_header</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x6</td>
<td>3</td>
<td>3</td>
<td>74 02 45 43 55</td>
</tr>
<tr>
<td>A</td>
<td>dest_address</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x4</td>
<td>3</td>
<td>3</td>
<td>00 02 00 01</td>
</tr>
<tr>
<td>P</td>
<td>previous_value</td>
<td>RAM</td>
<td>R</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

| 05 | Binary Switch (Heater Switch) : bswitch.cob |

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_position</td>
<td>RAM</td>
<td>R/W</td>
<td>Boolean</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>False</td>
</tr>
<tr>
<td>R</td>
<td>report_condition</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x4</td>
<td>3</td>
<td>3</td>
<td>43 05</td>
</tr>
<tr>
<td>H</td>
<td>report_header</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x6</td>
<td>3</td>
<td>3</td>
<td>74 05 45 43 25</td>
</tr>
<tr>
<td>A</td>
<td>dest_address</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x4</td>
<td>3</td>
<td>3</td>
<td>00 02 00 01</td>
</tr>
<tr>
<td>P</td>
<td>previous_value</td>
<td>RAM</td>
<td>R</td>
<td>Boolean</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>False</td>
</tr>
</tbody>
</table>

| 07 | Binary Sensor (Heater Intermediate Switch) : bsensor.cob |

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_state</td>
<td>RAM</td>
<td>R</td>
<td>Boolean</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>False</td>
</tr>
</tbody>
</table>

| 08 | Analog Sensor (Temperature Storage) : analsen.cob |

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_value</td>
<td>RAM</td>
<td>R</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

| 09 | Binary Sensor (Switch Ack) : bsensor.cob |

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_state</td>
<td>RAM</td>
<td>R/W</td>
<td>Boolean</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>False</td>
</tr>
</tbody>
</table>

| 0A | Analog Sensor (Temperature Ack) : analsen.cob |

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_value</td>
<td>RAM</td>
<td>R/W</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

---

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.

<table>
<thead>
<tr>
<th>ADDRESS PART</th>
<th>COMMAND PART</th>
<th>COMMAND STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>There are no devices with this address.</td>
<td>Valid or invalid command.</td>
<td>Invalid</td>
</tr>
<tr>
<td>A device does have this address.</td>
<td>The command does not apply to this device.</td>
<td>Invalid</td>
</tr>
<tr>
<td>A device does have this address.</td>
<td>The command value is out of range for this device.</td>
<td>Invalid</td>
</tr>
<tr>
<td>A device does have this address.</td>
<td>The command does not apply to any device.</td>
<td>Invalid</td>
</tr>
<tr>
<td>A device does have this address.</td>
<td>Valid command for device at this address.</td>
<td>Valid</td>
</tr>
</tbody>
</table>

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).
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_control())
      select = 3;
    else
      select = 2;
    break;

  case 3:
    if(initialize())
      select = 5;
    else
      select = 4;
    break;

  case 4:
    if(hangup())
      select = 6;
    else
      select = 5;
    break;

  default:
    select = -1;
    break;
  }
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 <i06811f1.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;
    rdetect &= 0x10;

    if(xdetect != 0x00 && rdetect == 0x10)
```c
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_count(0); //reset not valid counter
                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_count(0); //reset not valid counter
        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
    }
//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_trail_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
    //valid counter
    rselect = 5; 
    rready = 0; 
    rreading = 0; 
    rbasy = 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 rrbus;
    static int rreading;
    int rready;
    static int rbasy;
    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
    //valid counter
    rselect = 5;
    rrepeat = 0;
    rcount = 0;
    rreading = 0;
    rready = 0;
    rrbusy = 0;
    rbusty = 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
            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 times
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;
in_t com_b = 0;
in_t com_c = 0;
in_t com_d = 0;
in_t 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 = dtmf & maskhi;
    return(resp);

}

void dtmf_init(void)
{
    char status;

    status = dtmf;
    dtmf = 0;
    dtmf = 0;
    dtmf = 0x08;
    dtmf = 0;
    status = dtmf;
    dtmf = 0x0d;
    dtmf = 0;
}

void dtmfs_init(void)
{
    char status;

    status = dtmf;
    dtmf = 0;
    dtmf = 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};

        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
            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
            peek_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
            peek_IV_bool(0xa1, 0x74, 0x07, "C", &heat);
            if(!heat)
                poke_IV_bool(0xa1, 0x74, 0x07, "C", 1);
            else
                v_alreadyon();
            break;
        default:      // set temp
        }
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;
}
apl4.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[] = "Invalid 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

//functions
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, cnode, 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 date transfer to controlled device.
/*---------------------------------------------------------------------------------------------------------------*/

#define IO_GLOBALS_OWNER

#include <string.h>
#include "cebtypes.h"
#include "io.h"
#include "loglinkl.h"
#include "networkl.h"
#include "msgxferl.h"
#include "userl.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 <io6811f1.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)
    {
        ;
    }

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

CONTROLLER APPLICATION FLOWCHARTS

controller

Line detect

Device access request?

yes

Local request?

no

Local Control

Local finished?

no

Remote request?

yes

Remote Control

1 2 3 4

103
Line Detect

1. Check telephone line
   - Yes: Local request?
   - No: Remote request?
     - Yes: Request access code
     - No: Return

2. Request command code
   - Yes: Local request?
     - Yes: Request access code
     - No: Remote request?
       - Yes: Request access code
       - No: Return

Return to Start
Local Control

- Check DTMF status
  - no
    - Digit available?
      - yes
        - Get digit from DTMF
          - Last digit?
            - no
              - Error response
            - yes
              - Check command code
                - Valid command code?
                  - no
                    - Error response
                  - yes
                    - Return
Remote Access Control

1. Check DTMF status
2. Digit available?
   - yes: Get digit from DTMF
   - no: Last digit?
     - yes: Check access code
     - no: Last chance?
       - yes: Error response
       - no: Prompt for command code
3. Valid access code?
   - yes: Return
   - no: Continue flowchart
Remote Command Control

- Was access code valid?
  - yes: Check DTMF status
  - no: Digit available?
    - yes: Get digit from DTMF
      - no: Last digit?
        - yes: Check command code
          - no: Error response
          - yes: Valid command code?
            - yes: Return
            - no: Error response
DTMF Status Check

Check DTMF status

Digit available?

yes
Set "ready" flag

no
Reset "ready" flag

Flag reset?

no
Reset "not-ready" counter

yes
Increment "not-ready" counter

Maximum count?

no
Return

yes
Set "ready" flag to max.
Command Activation (without analog adjustment)

- **Turn device off?**
  - **yes**
  - Check device status
  - **Is device off?**
    - **no**
    - Turn device off
    - **yes**
    - "Device already off"
  - **no**
  - Turn device on?
    - **yes**
    - Check device status
    - **Is device on?**
      - **no**
      - Turn device on
      - **yes**
      - "Device already on"
    - **no**
      - Return

Command Activation (with analog adjustment)

- Turn device off?
  - yes: Check device status
  - no: Turn device off

- Is device off?
  - yes: "Device already off"
  - no: Turn device off

- Turn device on?
  - yes: Check device status
    - no: Turn device on
    - yes: "Device already on"
  - no: Turn device on
Command Activation (with analog adjustment) – cont’d

1

Set analog value

2

Return
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 userl
pause
icc6811 -mb -e -g -L -q -K -P -RRCODE hcllhal
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 hcll main ivs userl io hcllhal lcd appl4 appl3 appl2 appl1 voice cb11libb
msgx11b netw11b -o done -l done.map
:xlink -f hcll main -o done -l done.map
pause
sconv done.a07 code.a07
keystack "2code.a07" 13 "M" 13 "3" 13 "A"
eepl
The following Contexts, Objects and IVs are used for the Controller:

### CEBus Remote Access Controller

<table>
<thead>
<tr>
<th>Context</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0 Universal Q: univ.cxl</td>
<td>00</td>
</tr>
<tr>
<td><strong>OBJECT</strong></td>
<td><strong>CLASS</strong></td>
</tr>
<tr>
<td>Node Control (Device Control): nodectrl.cob</td>
<td>01</td>
</tr>
<tr>
<td>serial #: ROM R String 21 3 3 GA1198</td>
<td></td>
</tr>
<tr>
<td>manuf_name: ROM R String 21 3 3 GA CEBus Solutions</td>
<td></td>
</tr>
<tr>
<td>manuf_model: ROM R String 21 3 3 CON997</td>
<td></td>
</tr>
<tr>
<td>product_class: ROM R String 21 3 3 UNLISTED</td>
<td></td>
</tr>
<tr>
<td>system_addr: NVM R/W Data 1x2 3 1 00 03</td>
<td></td>
</tr>
<tr>
<td>mac_addr: NVM R/W Data 1x2 3 1 00 01</td>
<td></td>
</tr>
<tr>
<td>context_list: ROM R Data 4x2 3 3 a0 00 a0 21 a0 74 a1 74</td>
<td></td>
</tr>
<tr>
<td>configured: RAM R/W Boolean 1 3 3 True</td>
<td></td>
</tr>
<tr>
<td>setup: RAM R/W Integer 2 3 3 0</td>
<td></td>
</tr>
<tr>
<td>user_feedback: RAM R/W Integer 2 3 3 0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Context</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>02 Context Control (Context Control): cntxctrl.cob</td>
<td>02</td>
</tr>
<tr>
<td>object_list: ROM R Data 3x2 3 3 01 01 02 02 16 03</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Context</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>03 Data Memory (Event Manager): datamem.cob</td>
<td>16</td>
</tr>
<tr>
<td>current_index: RAM R/W Integer 2 3 3 0</td>
<td></td>
</tr>
<tr>
<td>memory_block: RAM R/W Data 1x2 3 3</td>
<td></td>
</tr>
</tbody>
</table>
# CEBus Remote Access Controller

## Light (Switches): light.cxl

<table>
<thead>
<tr>
<th>#</th>
<th>OBJECT</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Context Control (Context Control)</td>
<td>cntxctrl.cob</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>object_list</td>
<td>ROM</td>
<td>R</td>
<td>Data</td>
<td>4x2</td>
<td>3</td>
<td>3</td>
<td>02 01 06 06 06 07 06 08</td>
</tr>
</tbody>
</table>

## Binary Sensor (Light Switch): bsensor.cob

<table>
<thead>
<tr>
<th>#</th>
<th>OBJECT</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>06</td>
<td>Binary Sensor (Light Switch)</td>
<td>bsensor.cob</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_state</td>
<td>RAM</td>
<td>R</td>
<td>Boolean</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>False</td>
</tr>
<tr>
<td>R</td>
<td>report_condition</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x4</td>
<td>3</td>
<td>3</td>
<td>43 ed 00</td>
</tr>
<tr>
<td>H</td>
<td>report_header</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x6</td>
<td>3</td>
<td>3</td>
<td>21 06 45 43 f5</td>
</tr>
<tr>
<td>A</td>
<td>dest_address</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x4</td>
<td>3</td>
<td>3</td>
<td>00 02 00 01</td>
</tr>
<tr>
<td>P</td>
<td>previous_value</td>
<td>RAM</td>
<td>R</td>
<td>Boolean</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>False</td>
</tr>
</tbody>
</table>

## Binary Sensor (Light Intermediate Switch): bsensor.cob

<table>
<thead>
<tr>
<th>#</th>
<th>OBJECT</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>07</td>
<td>Binary Sensor (Light Intermediate Switch)</td>
<td>bsensor.cob</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_state</td>
<td>RAM</td>
<td>R</td>
<td>Boolean</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>False</td>
</tr>
</tbody>
</table>

## Binary Sensor (Ack): bsensor.cob

<table>
<thead>
<tr>
<th>#</th>
<th>OBJECT</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td>Binary Sensor (Ack)</td>
<td>bsensor.cob</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_state</td>
<td>RAM</td>
<td>R/W</td>
<td>Boolean</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>False</td>
</tr>
</tbody>
</table>
# CEBus Remote Access Controller

## Context Control (Context Control) : ctxctrl.cob

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>object_list</td>
<td>ROM</td>
<td>R</td>
<td>Data</td>
<td>7x2</td>
<td>3</td>
<td>2</td>
<td>02, 01, 07, 02, 05, 05, 06, 07, 08, 08, 06, 09, 08, 0a</td>
</tr>
</tbody>
</table>

## Analog Control (Temperature Setting) : analctrl.cob

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>min_value</td>
<td>ROM</td>
<td>R</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>65</td>
</tr>
<tr>
<td>M</td>
<td>max_value</td>
<td>ROM</td>
<td>R</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>85</td>
</tr>
<tr>
<td>D</td>
<td>default_value</td>
<td>ROM</td>
<td>R</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>72</td>
</tr>
<tr>
<td>C</td>
<td>current_value</td>
<td>RAM</td>
<td>R/W</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>R</td>
<td>report_condition</td>
<td>NVMM</td>
<td>R/W</td>
<td>Data</td>
<td>1x4</td>
<td>3</td>
<td>3</td>
<td>43 ed 31</td>
</tr>
<tr>
<td>H</td>
<td>report_header</td>
<td>NVMM</td>
<td>R/W</td>
<td>Data</td>
<td>1x6</td>
<td>3</td>
<td>3</td>
<td>17 74 02 45 43 f6</td>
</tr>
<tr>
<td>A</td>
<td>dest_address</td>
<td>NVMM</td>
<td>R/W</td>
<td>Data</td>
<td>1x4</td>
<td>3</td>
<td>3</td>
<td>00 02 00 01</td>
</tr>
<tr>
<td>P</td>
<td>previous_value</td>
<td>RAM</td>
<td>R</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

## Binary Switch (Heater Switch) : bswitch.cob

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_position</td>
<td>RAM</td>
<td>R/W</td>
<td>Boolean</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>False</td>
</tr>
<tr>
<td>R</td>
<td>report_condition</td>
<td>NVMM</td>
<td>R/W</td>
<td>Data</td>
<td>1x4</td>
<td>3</td>
<td>3</td>
<td>43 ed 00</td>
</tr>
<tr>
<td>H</td>
<td>report_header</td>
<td>NVMM</td>
<td>R/W</td>
<td>Data</td>
<td>1x6</td>
<td>3</td>
<td>3</td>
<td>17 74 05 45 43 f5</td>
</tr>
<tr>
<td>A</td>
<td>dest_address</td>
<td>NVMM</td>
<td>R/W</td>
<td>Data</td>
<td>1x4</td>
<td>3</td>
<td>3</td>
<td>00 02 00 01</td>
</tr>
<tr>
<td>P</td>
<td>previous_value</td>
<td>RAM</td>
<td>R</td>
<td>Boolean</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>False</td>
</tr>
</tbody>
</table>

## Binary Sensor (Heater Intermediate Switch) : bsensor.cob

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_state</td>
<td>RAM</td>
<td>R</td>
<td>Boolean</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>False</td>
</tr>
</tbody>
</table>

## Analog Sensor (Temperature Storage) : analsens.cob

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_value</td>
<td>RAM</td>
<td>R</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

## Binary Sensor (Switch Ack) : bsensor.cob

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_state</td>
<td>RAM</td>
<td>R/W</td>
<td>Boolean</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>False</td>
</tr>
</tbody>
</table>

## Analog Sensor (Temperature Ack) : analsens.cob

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_value</td>
<td>RAM</td>
<td>R/W</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>#</td>
<td>CONTEXT</td>
<td>ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>------------------------------------------</td>
<td>----</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A0</td>
<td>Refrigerator/Freezer (AC) : refrigfr.cxl</td>
<td>74</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>OBJECT</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Context Control (Context Control) : cntxctrl.cob</td>
<td>02</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>object_list</td>
<td>ROM</td>
<td>R</td>
<td>Data</td>
<td>7x2</td>
<td>3</td>
<td>3</td>
<td>02 01 07 02 03 05 06 07 08 08 06 09 08 0a</td>
</tr>
</tbody>
</table>

| 02 | Analog Control (Temperature Setting) : analctrl.cob | 07    |

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>min_value</td>
<td>ROM</td>
<td>R</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>M</td>
<td>max_value</td>
<td>ROM</td>
<td>R</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>40</td>
</tr>
<tr>
<td>D</td>
<td>default_value</td>
<td>ROM</td>
<td>R</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>32</td>
</tr>
<tr>
<td>C</td>
<td>current_value</td>
<td>RAM</td>
<td>R/W</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>R</td>
<td>report_condition</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x4</td>
<td>3</td>
<td>3</td>
<td>43 ed 31</td>
</tr>
<tr>
<td>H</td>
<td>report_header</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x6</td>
<td>3</td>
<td>3</td>
<td>a0 74 02 45 43 f5</td>
</tr>
<tr>
<td>A</td>
<td>dst_address</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x4</td>
<td>3</td>
<td>3</td>
<td>00 02 00 01</td>
</tr>
<tr>
<td>P</td>
<td>previous_value</td>
<td>RAM</td>
<td>R</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

| 05 | Binary Switch (AC Switch) : bswitch.cob | 05    |

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_position</td>
<td>RAM</td>
<td>R/W</td>
<td>Boolean</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>False</td>
</tr>
</tbody>
</table>

| 07 | Binary Sensor (AC Intermediate Switch) : bsensor.cob | 06    |

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_state</td>
<td>RAM</td>
<td>R</td>
<td>Boolean</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>False</td>
</tr>
</tbody>
</table>

| 08 | Analog Sensor (Temperature Storage) : analysens.cob | 08    |

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_value</td>
<td>RAM</td>
<td>R</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

| 09 | Binary Sensor (Switch Ack) : bsensor.cob | 06    |

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_state</td>
<td>RAM</td>
<td>R/W</td>
<td>Boolean</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>False</td>
</tr>
</tbody>
</table>

| 0A | Analog Sensor (Temperature Ack) : analysens.cob | 08    |

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_value</td>
<td>RAM</td>
<td>R</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
The following Contexts, Objects and IVs were used on the computer to simulate a CEBus device:

### CEBus Device Simulator

<table>
<thead>
<tr>
<th>#</th>
<th>CONTEXT</th>
<th>OBJECT</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Node Control (Device Control) : nodectl.cob</td>
<td>+1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IV</td>
<td>NAME</td>
<td>MEM</td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>manuf_name</td>
<td>ROM</td>
</tr>
<tr>
<td></td>
<td>h</td>
<td>system_addr</td>
<td>NVM</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>mac_addr</td>
<td>NVM</td>
</tr>
<tr>
<td></td>
<td>o</td>
<td>context_list</td>
<td>ROM</td>
</tr>
<tr>
<td></td>
<td>f</td>
<td>configured</td>
<td>RAM</td>
</tr>
<tr>
<td></td>
<td>i</td>
<td>setup</td>
<td>RAM</td>
</tr>
<tr>
<td></td>
<td>u</td>
<td>user_feedback</td>
<td>RAM</td>
</tr>
<tr>
<td>02</td>
<td>Context Control (Context Control) : cntxctrl.cob</td>
<td>+2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IV</td>
<td>NAME</td>
<td>MEM</td>
</tr>
<tr>
<td></td>
<td>o</td>
<td>object_list</td>
<td>ROM</td>
</tr>
<tr>
<td>03</td>
<td>Data Memory (Event Manager) : datamem.cob</td>
<td>+3</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>IV</td>
<td>NAME</td>
<td>MEM</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>current_index</td>
<td>RAM</td>
</tr>
<tr>
<td></td>
<td>l</td>
<td>memory_block</td>
<td>RAM</td>
</tr>
</tbody>
</table>

### CEBus Device Simulator

<table>
<thead>
<tr>
<th>#</th>
<th>CONTEXT</th>
<th>OBJECT</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Light (Switch) : light.cxl</td>
<td>+1</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>IV</td>
<td>NAME</td>
<td>MEM</td>
</tr>
<tr>
<td></td>
<td>o</td>
<td>object_list</td>
<td>ROM</td>
</tr>
<tr>
<td>06</td>
<td>Binary Switch (Light Switch) : bswitch.cob</td>
<td>+6</td>
<td>05</td>
</tr>
<tr>
<td></td>
<td>IV</td>
<td>NAME</td>
<td>MEM</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>current_position</td>
<td>RAM</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>report_condition</td>
<td>NVM</td>
</tr>
<tr>
<td></td>
<td>H</td>
<td>report_header</td>
<td>NVM</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>dest_address</td>
<td>NVM</td>
</tr>
<tr>
<td></td>
<td>P</td>
<td>previous_value</td>
<td>RAM</td>
</tr>
<tr>
<td>07</td>
<td>Binary Sensor (Intermediate Switch) : bsensor.cob</td>
<td>+7</td>
<td>06</td>
</tr>
<tr>
<td></td>
<td>IV</td>
<td>NAME</td>
<td>MEM</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>current_state</td>
<td>RAM</td>
</tr>
</tbody>
</table>
### CEBus Device Simulator

<table>
<thead>
<tr>
<th>#</th>
<th>CONTEXT</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>Refrigerator/Freezer (Heater) : reflagr.cx1</td>
<td>74</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OBJECT</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Context Control (Context Control) : cntxctrl.cob</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>object_list</td>
<td>ROM</td>
<td>R</td>
<td>Data</td>
<td>3x2</td>
<td>3</td>
<td>3</td>
<td>02 01 07 02 05 05</td>
</tr>
</tbody>
</table>

| 02 | Analog Control (Temperature Setting) : analctrl.cob | 07 |

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_value</td>
<td>RAM</td>
<td>R/W</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>R</td>
<td>report_condition</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x4</td>
<td>3</td>
<td>3</td>
<td>43 ed 31</td>
</tr>
<tr>
<td>H</td>
<td>report_header</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x6</td>
<td>3</td>
<td>3</td>
<td>al 74 0a 45 43 55</td>
</tr>
<tr>
<td>A</td>
<td>dest_address</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x4</td>
<td>3</td>
<td>3</td>
<td>00 01 00 03</td>
</tr>
<tr>
<td>P</td>
<td>previous_value</td>
<td>RAM</td>
<td>R/W</td>
<td>Integer</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

| 05 | Binary Switch (Switch) : bswitch.cob | 05 |

<table>
<thead>
<tr>
<th>IV</th>
<th>NAME</th>
<th>MEM</th>
<th>RW</th>
<th>TYPE</th>
<th>SIZE</th>
<th>RL</th>
<th>WL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>current_position</td>
<td>RAM</td>
<td>R/W</td>
<td>Boolean</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>False</td>
</tr>
<tr>
<td>R</td>
<td>report_condition</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x4</td>
<td>3</td>
<td>3</td>
<td>43 ed 00</td>
</tr>
<tr>
<td>H</td>
<td>report_header</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x6</td>
<td>3</td>
<td>3</td>
<td>al 74 09 45 43 55</td>
</tr>
<tr>
<td>A</td>
<td>dest_address</td>
<td>NVM</td>
<td>R/W</td>
<td>Data</td>
<td>1x4</td>
<td>3</td>
<td>3</td>
<td>00 01 00 03</td>
</tr>
<tr>
<td>P</td>
<td>previous_value</td>
<td>RAM</td>
<td>R</td>
<td>Boolean</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>False</td>
</tr>
</tbody>
</table>
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], dispna, ctload, voice : Output;
    dtmfc, lc : Output;
)

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

BEGIN
    bnk = (addr[]==H"1060" & !iocs);
    dispua = (addr[]==H"1064" & !iocs);
    dispb = (addr[]==H"1065" & !iocs);
    ctload = (addr[]==H"1063" & !iocs);
    voice = (addr[]==H"1062" & !iocs);
    dtmfa = (addr[]==H"1066" & !iocs);
    dtmfb = (addr[]==H"1067" & !iocs);
    line = (addr[]==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 %
dacl.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.
### APPENDIX G

**PARTS LIST**

#### SHEET 1 (PROCESSOR)

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

#### SHEET 2 (PLD/LCD)

<table>
<thead>
<tr>
<th>PART ID</th>
<th>PART NUMBER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP1</td>
<td>OPT40X2</td>
<td>LCD Display</td>
</tr>
<tr>
<td>R9</td>
<td>10K</td>
<td>Potentiometer</td>
</tr>
<tr>
<td>U8</td>
<td>CLO7064</td>
<td>Erasable Programmable Logic Device</td>
</tr>
<tr>
<td>U9</td>
<td>7408</td>
<td>Quadruple 2-input AND gate</td>
</tr>
<tr>
<td>PART ID</td>
<td>PART NUMBER</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>BR1</td>
<td>6B4B41</td>
<td>Bridge rectifier</td>
</tr>
<tr>
<td>T1</td>
<td>120VAC/12.6VDC</td>
<td>Transformer</td>
</tr>
<tr>
<td>T2</td>
<td>18:18 turns</td>
<td>Transformer</td>
</tr>
<tr>
<td>CR1</td>
<td>1N5336B</td>
<td>Diode</td>
</tr>
<tr>
<td>CR2</td>
<td>1N5336B</td>
<td>Diode</td>
</tr>
<tr>
<td>C4</td>
<td>470μF 16V</td>
<td>Capacitor</td>
</tr>
<tr>
<td>C5</td>
<td>470μF 16V</td>
<td>Capacitor</td>
</tr>
<tr>
<td>C6</td>
<td>470μF 16V</td>
<td>Capacitor</td>
</tr>
<tr>
<td>C7</td>
<td>470μF 16V</td>
<td>Capacitor</td>
</tr>
<tr>
<td>C8</td>
<td>10μF 16V</td>
<td>Capacitor</td>
</tr>
<tr>
<td>C9</td>
<td>1μF 35V</td>
<td>Capacitor</td>
</tr>
<tr>
<td>VR1</td>
<td>LM2940-8</td>
<td>8V 1amp low dropout voltage regulator</td>
</tr>
<tr>
<td>VR2</td>
<td>LM2940-5</td>
<td>5V 1amp low dropout voltage regulator</td>
</tr>
<tr>
<td>U10</td>
<td>MC34064</td>
<td>Microprocessor supervisor</td>
</tr>
<tr>
<td>U11</td>
<td>MC34064</td>
<td>Microprocessor supervisor</td>
</tr>
<tr>
<td>R7</td>
<td>220</td>
<td>Resistor</td>
</tr>
<tr>
<td>R8</td>
<td>4K7</td>
<td>Resistor</td>
</tr>
<tr>
<td>C3</td>
<td>100pF</td>
<td>Capacitor</td>
</tr>
<tr>
<td>S1</td>
<td>PBS100</td>
<td>Push button switch</td>
</tr>
</tbody>
</table>
### SHEET 4 (SPEECH AND TONE CIRCUITS)

<table>
<thead>
<tr>
<th>PART ID</th>
<th>PART NUMBER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>C10</td>
<td>10µF</td>
<td>Capacitor</td>
</tr>
<tr>
<td>C11</td>
<td>0.1µF</td>
<td>Capacitor</td>
</tr>
<tr>
<td>C12</td>
<td>0.1µF</td>
<td>Capacitor</td>
</tr>
<tr>
<td>C13</td>
<td>0.1µF</td>
<td>Capacitor</td>
</tr>
<tr>
<td>C14</td>
<td>0.1µF</td>
<td>Capacitor</td>
</tr>
<tr>
<td>C15</td>
<td>0.1µF</td>
<td>Capacitor</td>
</tr>
<tr>
<td>M1</td>
<td>V8600</td>
<td>Speech Synthesizer</td>
</tr>
<tr>
<td>R10</td>
<td>2K</td>
<td>Resistor</td>
</tr>
<tr>
<td>R11</td>
<td>100K</td>
<td>Resistor</td>
</tr>
<tr>
<td>R12</td>
<td>100K</td>
<td>Resistor</td>
</tr>
<tr>
<td>R13</td>
<td>390</td>
<td>Resistor</td>
</tr>
<tr>
<td>R14</td>
<td>4K7</td>
<td>Resistor</td>
</tr>
<tr>
<td>R15</td>
<td>15K</td>
<td>Resistor</td>
</tr>
<tr>
<td>S1</td>
<td>PBS100</td>
<td>Push button switch</td>
</tr>
<tr>
<td>U12</td>
<td>7408</td>
<td>Quadruple 2-input AND gate</td>
</tr>
<tr>
<td>U13</td>
<td>MT8888CE</td>
<td>DTMF transceiver</td>
</tr>
<tr>
<td>U18</td>
<td>7400</td>
<td>Quadruple 2-input NAND gates</td>
</tr>
<tr>
<td>U19</td>
<td>7404</td>
<td>Hex inverter</td>
</tr>
<tr>
<td>U20</td>
<td>7404</td>
<td>Hex inverter</td>
</tr>
<tr>
<td>U21</td>
<td>7400</td>
<td>Quadruple 2-input NAND gates</td>
</tr>
<tr>
<td>Y2</td>
<td>3.579545 MHz</td>
<td>Crystal</td>
</tr>
<tr>
<td>PART ID</td>
<td>PART NUMBER</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>U20</td>
<td>7404</td>
<td>Hex inverter</td>
</tr>
<tr>
<td>U17</td>
<td>7408</td>
<td>Quadruple 2-input AND gate</td>
</tr>
<tr>
<td>U19</td>
<td>7404</td>
<td>Hex inverter</td>
</tr>
<tr>
<td>U18</td>
<td>7400</td>
<td>Quadruple 2-input NAND gates</td>
</tr>
<tr>
<td>U14</td>
<td>MH88434-P</td>
<td>Data Access Arrangement</td>
</tr>
<tr>
<td>U22</td>
<td>74LS123</td>
<td>Dual monostable multivibrator with clear</td>
</tr>
<tr>
<td>R17</td>
<td>100K</td>
<td>Resistor</td>
</tr>
<tr>
<td>R18</td>
<td>2K</td>
<td>Resistor</td>
</tr>
<tr>
<td>R19</td>
<td>300</td>
<td>Resistor</td>
</tr>
<tr>
<td>R20</td>
<td>4K7</td>
<td>Resistor</td>
</tr>
<tr>
<td>R21</td>
<td>6K8</td>
<td>Resistor</td>
</tr>
<tr>
<td>R22</td>
<td>47K</td>
<td>Resistor</td>
</tr>
<tr>
<td>C16</td>
<td>10μF</td>
<td>Capacitor</td>
</tr>
<tr>
<td>C17</td>
<td>1μF</td>
<td>Capacitor</td>
</tr>
<tr>
<td>C18</td>
<td>100μF</td>
<td>Capacitor</td>
</tr>
<tr>
<td>C19</td>
<td>47μF</td>
<td>Capacitor</td>
</tr>
<tr>
<td>C20</td>
<td>10μF</td>
<td>Capacitor</td>
</tr>
<tr>
<td>S3</td>
<td>SW153</td>
<td>SPDT switch</td>
</tr>
</tbody>
</table>
REFERENCES

17. TTL Logic Data Book, Texas Instrument, Dallas, TX, 1988.