Table of Contents

Overview:  
- How to use this manual  
- Programmable Devices by Category  

Theory of Operations:  
- Hardware  
- Software  

Specifications:  
- Hardware Interface Specifications  
- Software Interface Specifications  
- Electrical Specifications  
- Mechanical Specifications  

User Modifications:  
- User Specified Programming Algorithms  
  Omni-Programmer Index File  
  Method I  
  Method II  
  Assembly Language Assist Routines  
    For input to Stream  
    For input to Mapword  
    Assemble Word  
- Porting Procedure  

Trouble Shooting:  
- Procedure  
- Problem Reporting  
- Problem Report Form  
- Customer Response Form  
- Schematics  
  Logic Board Schematics  
  Socket Board Schematics  
- User Diagnostics  
  BW Instructions with IBM PC  
  BW Instructions with Kaypro  

03/01/84   Omni-Programmer Technical Manual   Release 1.0
Overview

The Varix Corporation Omni-Programmer product line is a combination of hardware and software which is designed to generate the necessary control voltages to place data into programmable integrated circuits. The hardware manipulates the currents and voltages for the creation of programming waveforms and the software controls the sequence of hardware events required for chip programming. Since the hardware has no chip dependent circuitry in it, it remains unchanged while software configures the socket pins appropriately for the targeted programmable device. The software runs on a microcomputer and issues commands to the Omni-Programmer hardware via a high speed parallel interface. During the programming process, the microcomputer must be 100% dedicated to the programming function since much of the critical algorithm timing is controlled by instruction timing. When not involved in programming, the attached computer is available for data down load and up load to larger computers, data printing, or general development activities.

There are two major categories of programmable integrated circuits -- read only memory (ROM) and programmable logic devices (PLD). ROM's represent the larger group and use one of two data storage techniques: fuse link, in which a tiny fuse inside the chip is destroyed when data is inserted and left intact otherwise; and charge storage, in which a small capacitor on the chip is either charged or discharged to represent the binary state of each bit. There are four technologies generally used to manufacture programmable integrated circuits. They are TTL bipolar, NMOS, CMOS, and ECL.

TTL bipolar is the oldest of the technologies and uses fuse link techniques for storing data. TTL bipolar's only advantage is high speed. The chips using this technology use a lot of power and are fairly expensive compared to MOS. NMOS is the highest density technology, is fairly low power, and is inexpensive. Charge storage technology is always used. Up to 512,000 bits can be put on a single chip with today's technology. EPROMs and EEPROMs are in this group. The greatest disadvantage of NMOS is its low speed. CMOS is similar to NMOS but is extremely low power. It uses either charge storage or fuse link technology tends to be fairly expensive and ranges from slow to fairly fast. The most rapid new developments are in this area. ECL is an infrequently used technology reserved for those applications in which blazing speed is the only consideration. They are fuse link devices which eat power, store only small amounts of data, and are exorbitantly expensive. Still, they offer a 10 times speed improvement over other device types.

While the major use of ROM devices is the permanent storage of data, PLD's are oriented toward the control of data flow. The data storage elements in a PLD are typically configured in a pattern to replace logic devices in a circuit board. Consequently, PLD's are always fuse link devices and are generally TTL bipolar technology. There is ongoing development of PLD's in CMOS, ECL, and charge storage (EEPLA) devices, but none are currently in production.
Due to the large number of programmable devices available on the market, Varix has divided them into categories delineated by function and technology type. Each of these categories is called a license device group (LDG). There are currently 10 LDG's in the Omni product line.

The Omni-Programmer is designed to program the entire universe of programmable devices through software programs. These programs are ordinarily supplied by Varix Corporation, but they can be written by the user if so desired. Guidance on user written programming algorithms is in the user modification section of this manual. There is an occasional device whose required voltages are outside the specifications of the Omni-Programmer which are detailed in section 3, in which case the device cannot be programmed. Varix is committed to providing the software necessary for almost all standard production devices in the most timely manner possible.

This manual provides the information necessary to modify that software, port Omni-Programmer to a new microcomputer, and troubleshoot the Omni software or hardware if problems should arise.
Overview cont.

Programmable Devices By Catagory:

<table>
<thead>
<tr>
<th>Varix Software</th>
<th>Description</th>
<th>Technology</th>
<th>Storage Mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDG01</td>
<td>EPROMs (UV erasable programmable read only memory) most common, highest density, low speed</td>
<td>NMOS</td>
<td>Charge Storage</td>
</tr>
<tr>
<td>LDG02</td>
<td>TTL Bipolar PROMs</td>
<td>TTL Bipolar</td>
<td>Fuse Link</td>
</tr>
<tr>
<td>LDG03</td>
<td>TTL Bipolar PROMs (Current programmed) same as LDG02 except current is used for programming the fuses with data</td>
<td>TTL Bipolar</td>
<td>Fuse Link</td>
</tr>
<tr>
<td>LDG04</td>
<td>Obsolete Devices</td>
<td>All Types</td>
<td>All Types</td>
</tr>
<tr>
<td>LDG05</td>
<td>Microprocessors</td>
<td>NMOS</td>
<td>Charge Storage</td>
</tr>
<tr>
<td>LDG06</td>
<td>PALs (programmable array logic) a small scale, limited PLD</td>
<td>TTL Bipolar</td>
<td>Fuse Link</td>
</tr>
<tr>
<td>LDG07</td>
<td>PLAs (programmable logic arrays) General PLDs plus a few special application units like programmable multiplexors</td>
<td>TTL Bipolar</td>
<td>Fuse Link</td>
</tr>
<tr>
<td>LDG08</td>
<td>CMOS</td>
<td>CMOS</td>
<td>Charge Storage</td>
</tr>
<tr>
<td>LDG09</td>
<td>EAROMs (electrically alterable read only memory) special byte by byte alterable devices</td>
<td>NMOS</td>
<td>Charge Storage</td>
</tr>
<tr>
<td>LDG10</td>
<td>EEPROMs (electrically erasable ROM) can be erased and reprogrammed electrically</td>
<td>NMOS</td>
<td>Charge Storage</td>
</tr>
</tbody>
</table>

03/01/84 Omni-Programmer Technical Manual Release 1.0
Theory of Operations

Hardware

The Omni-Programmer is a dumb box whose purpose is to provide controlled voltages and currents to the pins of an integrated circuit under test. All of the level settings, ramp rates, pulse widths, and durations of voltages are controlled in real time by the software in the attached microcomputer.

There are three programmable voltage supplies in the Omni whose output voltage can be selectively set by software. Their current limit can also be controlled. There are 256 voltage steps and 256 current steps for each of the three supplies. The supplies are referred to as v1, v2, and v3. The current being delivered by each supply can be measured in steps of 256. An 8 bit D/A converter is used to adjust the voltage + current limit on each supply. Current measurement is accomplished by comparing a variable threshold voltage against the voltage drop of a series resistor in each power supply circuit. In addition to the three voltages, any pin can be set to TTL high or TTL low.

The three power supply voltages are multiplexed to various pins on the programming sockets and are selectively enabled with transistor switches. Each switch is controlled by a bit from an octal latch on the Omni logic board. Voltage control bits on the Omni logic board are arranged as an array of sixteen 8 bit output ports and six 8 bit input ports. Voltage and current D/A converters, LEDs, and control bits are contained in the balance of the I/O ports. In all, 32 I/O ports are assigned to an SP0300 Omni-Programmer, 20 for output, 8 for input.

The interface between the Omni-Programmer and its attached microcomputer is designed to allow several configurations so that it can be readily adapted through cable wiring to most easily match the microcomputer. Basically, there is an 8 bit address, 8 bits of output, 8 bits of data in, control lines, and ground reference in the cable. These buses can be run separately, requiring 16 bits of output, 8 bits of input and only one strobe to indicate when the next data transfer should be made. The input data in this mode will asynchronously represent the data from the input port selected by the address bus. At the other extreme, the entire bus may be multiplexed on 8 data lines. In this case, an address strobe is used to indicate valid address on the bus, a data strobe is used to indicate valid data, and an input strobe is used to turn the bus around for reading. Both the address and data are latched by the Omni in this configuration.

NOTE: due to the high data transfer rates required for real time manipulations of voltages on pins, it is impractical to use any kind of serial interface scheme. The required transfer rate exceeds 1 Mbaud.
Software

The software for the Omni-Programmer runs exclusively on the attached microcomputer. With the exception of critical timing code, it is written in the high level language "C" to promote portability from one microcomputer to another. The software is divided into three sections:

One: the hardware interface is 1000 bytes of assembly language code which communicates between the microcomputer/omni hardware and the software. Only this section is modified when porting Omni from one microcomputer to another so long as both are using the same microprocessor and operating system.

Two, the programming algorithm is 4000 bytes of "C" and assembler which creates the waveforms and controls the programming algorithm for the device being programmed.

Three, the user interface is 52,000 bytes of "C" code which generates the displays, accepts and executes user commands, and handles all system functions.

Both the hardware interface and programming algorithm software can be modified by the user. Techniques for doing this are described in the user modification section. Communications between the various modules is achieved through fixed jump vectors and status words in each module. This allows each module to operate entirely on its own and offers no restrictions on what language the module is written in so long as it adheres to the fixed interface specifications.

In addition to the actual programming code, there are a group of special utilities which enhance the programming environment. Omni comm supports communications between the microcomputer and a host machine so that chip data can be received from a remote source. Monolithic Memory's PALASM allows the creation of programmable logic device patterns from Boolean logic equations. Statistics prints historical information on Omni programming activity. BW offers diagnostic testing on the hardware.

One of the major advantages of the Varix Omni-Programmer is the flexibility it gives the user in writing software -- either as utilities, or as part of the programming environment, to manage the programming process for maximum efficiency.
Specifications

Hardware Interface Specifications

The I/O port is sent via the address bus from the microcomputer. The data on the data bus is used to select bits within the I/O port. The output ports are divided into primary, secondary, and auxiliary ports. The primary ports are assigned one bit per socket pin where that bit indicates a reset condition (zero volts) when low ("0") and some positive voltage when high ("1"). The secondary ports control which voltage is active when a primary port bit is high. In the absence of a secondary port for a particular pin, the high state creates a logic level "1" on the socket through a 10K pull-up resistor to 5v. This 10K resistor also provides the pull-up for open collector output devices. The state in which the 10K resistor to 5v is active is called the "disable" condition. Any output pin from a chip should be "disabled" prior to reading data for it.

The secondary ports assign either one or two bits to a socket pin depending on how many voltages are available for that pin. In the case of a single secondary bit, a "0" indicates disable condition and "1" selects a voltage (typically v2). In the two bit decode case, 00 is disable, 01 is v1, 10 is v2, and 11 is either v3 or clk. The clk is a 3MHz driven square wave for use on microprocessor chips.

The auxiliary ports are D/A data, LED control, and calibration logic control. The read ports read the state (0 or 1) of each pin as compared to the threshold voltage. Consequently, the actual voltage on a pin can be determined with .118 volts by adjusting the threshold voltage until a transition occurs on the relevant pin.
Write Ports (primary)

<table>
<thead>
<tr>
<th>I/O Port</th>
<th>Bit</th>
<th>Logic Level</th>
<th>48 Pin</th>
<th>28 Pin</th>
<th>24 Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>Reset *</td>
<td>21</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Reset *</td>
<td>22</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Reset *</td>
<td>23</td>
<td>13</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Reset *</td>
<td>25</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Reset *</td>
<td>26</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Reset *</td>
<td>27</td>
<td>17</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Reset *</td>
<td>28</td>
<td>18</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Reset *</td>
<td>29</td>
<td>19</td>
<td>3</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>Reset *</td>
<td>20</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Reset *</td>
<td>19</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Reset *</td>
<td>18</td>
<td>8</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Reset *</td>
<td>17</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Reset *</td>
<td>16</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Reset *</td>
<td>15</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Reset *</td>
<td>14</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Reset *</td>
<td>13</td>
<td>24</td>
<td>2</td>
</tr>
<tr>
<td>02</td>
<td>0</td>
<td>Reset *</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Reset *</td>
<td>34</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Reset *</td>
<td>31</td>
<td>21</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Reset *</td>
<td>33</td>
<td>27</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Reset *</td>
<td>12</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Reset *</td>
<td>36</td>
<td>25</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Reset *</td>
<td>37</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Reset *</td>
<td>30</td>
<td>20</td>
<td>8</td>
</tr>
<tr>
<td>03</td>
<td>0</td>
<td>Reset Disable</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Reset *</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Reset *</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Reset Disable</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Reset *</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Reset Disable</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Reset *</td>
<td>44</td>
<td>2</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>NOT USED</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NOTE: * - pin controlled by secondary I/O port with corresponding pin # shown on the following pages.
### Write Ports (primary)

<table>
<thead>
<tr>
<th>I/O Port</th>
<th>Bit</th>
<th>Logic Level</th>
<th>Pin</th>
<th>Pin</th>
<th>Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>04</td>
<td>0</td>
<td>Reset</td>
<td>Disable</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Reset</td>
<td>Disable</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Reset</td>
<td>Disable</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Reset</td>
<td>Disable</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Reset</td>
<td>Disable</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Reset</td>
<td>*</td>
<td>11</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Reset</td>
<td>*</td>
<td>38</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Reset</td>
<td>*</td>
<td>32</td>
<td>26</td>
</tr>
<tr>
<td>05</td>
<td>0</td>
<td>Reset</td>
<td>Disable</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Reset</td>
<td>Disable</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Reset</td>
<td>Disable</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Reset</td>
<td>*</td>
<td>48</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Reset</td>
<td>Disable</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Reset</td>
<td>Disable</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Reset</td>
<td>Disable</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Reset</td>
<td>Disable</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td></td>
<td>NOT USED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07</td>
<td></td>
<td>NOT USED</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Write Ports (secondary)

<table>
<thead>
<tr>
<th>I/O Port</th>
<th>Bit</th>
<th>Logic Level</th>
<th>Pin</th>
<th>Pin</th>
<th>Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td>0</td>
<td>Disable</td>
<td>clk</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Disable</td>
<td>clk</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Disable</td>
<td>v2</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Disable</td>
<td>v2</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Disable</td>
<td>v2</td>
<td>32</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Disable</td>
<td>v2</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Disable</td>
<td>v3</td>
<td>38</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Disable</td>
<td>v3</td>
<td>48</td>
<td>3</td>
</tr>
<tr>
<td>09</td>
<td>0</td>
<td>Disable</td>
<td>v2</td>
<td>31</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Disable</td>
<td>v2</td>
<td>18</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Disable</td>
<td>v2</td>
<td>17</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Disable</td>
<td>v2</td>
<td>16</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Disable</td>
<td>v2</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Disable</td>
<td>v2</td>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Disable</td>
<td>v2</td>
<td>12</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Disable</td>
<td>v2</td>
<td>12</td>
<td>23</td>
</tr>
</tbody>
</table>

**NOTE:** If pin is "RESET", logic level here is don't care.
Specifications

Hardware Interface Specifications con't.

Write Ports (secondary)

<table>
<thead>
<tr>
<th>I/O Port</th>
<th>Bit</th>
<th>Logic Level</th>
<th>48</th>
<th>28</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>00</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OA</th>
<th>1, 0</th>
<th>Disable</th>
<th>v1</th>
<th>v2</th>
<th>NC</th>
<th>21</th>
<th>11</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3, 2</td>
<td>Disable</td>
<td>v1</td>
<td>v2</td>
<td>NC</td>
<td>22</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>5, 4</td>
<td>Disable</td>
<td>v1</td>
<td>v2</td>
<td>CLK</td>
<td>23</td>
<td>13</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>7, 6</td>
<td>Disable</td>
<td>v1</td>
<td>v2</td>
<td>v3</td>
<td>25</td>
<td>15</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ob</th>
<th>1, 0</th>
<th>Disable</th>
<th>v1</th>
<th>v2</th>
<th>NC</th>
<th>26</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3, 2</td>
<td>Disable</td>
<td>v1</td>
<td>v2</td>
<td>NC</td>
<td>27</td>
<td>17</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>5, 4</td>
<td>Disable</td>
<td>v1</td>
<td>v2</td>
<td>NC</td>
<td>28</td>
<td>18</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>7, 6</td>
<td>Disable</td>
<td>v1</td>
<td>v2</td>
<td>NC</td>
<td>29</td>
<td>19</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OC</th>
<th>1, 0</th>
<th>Disable</th>
<th>v1</th>
<th>1.5</th>
<th>v2</th>
<th>v3</th>
<th>11</th>
<th>22</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3, 2</td>
<td>Disable</td>
<td>v1</td>
<td>v2</td>
<td>NC</td>
<td>20</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5, 4</td>
<td>Disable</td>
<td>v1</td>
<td>v2</td>
<td>NC</td>
<td>19</td>
<td>9</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7, 6</td>
<td>Disable</td>
<td>v1</td>
<td>v2</td>
<td>NC</td>
<td>30</td>
<td>20</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OD</th>
<th>1, 0</th>
<th>Disable</th>
<th>v1</th>
<th>v2</th>
<th>-5v</th>
<th>33</th>
<th>27</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3, 2</td>
<td>Disable</td>
<td>v1</td>
<td>v2</td>
<td>v3</td>
<td>36</td>
<td>25</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>5, 4</td>
<td>Disable</td>
<td>NC</td>
<td>v2</td>
<td>v3</td>
<td>44</td>
<td>2</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>7, 6</td>
<td>Disable</td>
<td>v1</td>
<td>v2</td>
<td>NC</td>
<td>13</td>
<td>24</td>
<td>2</td>
</tr>
</tbody>
</table>

NOTE: If pin is reset, logic levels are don't care.

NC - No connection (effect is disable).
## Specifications

### Hardware Interface Specifications con't.

#### Write Ports

<table>
<thead>
<tr>
<th>I/O Port</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OE</td>
<td>NOT USED</td>
<td></td>
</tr>
<tr>
<td>OF</td>
<td>NOT USED</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0-7</td>
<td>v1 Voltage Set</td>
</tr>
<tr>
<td>11</td>
<td>0-7</td>
<td>v2 Voltage Set</td>
</tr>
<tr>
<td>12</td>
<td>0-7</td>
<td>v3 Voltage Set</td>
</tr>
<tr>
<td>13</td>
<td>0-7</td>
<td>v1 Current Limit Set</td>
</tr>
<tr>
<td>14</td>
<td>0-7</td>
<td>v2 Current Limit Set</td>
</tr>
<tr>
<td>15</td>
<td>0-7</td>
<td>Threshold Set</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>LED 48 Pin Socket (&quot;D&quot; = ON)</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>LED 28 Pin Socket</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>LED 24 Pin Socket</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>LED &quot;Connected&quot;</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>LED &quot;Programming&quot;</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>LED &quot;Errcr&quot;</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>&quot;1&quot; = v1 Current Load Enable</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>&quot;1&quot; = v2 Current Load Enable</td>
</tr>
</tbody>
</table>

---

03/01/84 Omni-Programmer Technical Manual Release 1.0
<table>
<thead>
<tr>
<th>I/O Port</th>
<th>Bit</th>
<th>Pin</th>
<th>Pin</th>
<th>Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>21</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>22</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>23</td>
<td>13</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>25</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>26</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>27</td>
<td>17</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>28</td>
<td>18</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>29</td>
<td>19</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>20</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>19</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>18</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>17</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>16</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>15</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>14</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>13</td>
<td>24</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>34</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>31</td>
<td>21</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>33</td>
<td>27</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>12</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>36</td>
<td>25</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>37</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>30</td>
<td>20</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>44</td>
<td>2</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>GND</td>
<td>GND</td>
<td>GND</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>43</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>42</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>41</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>39</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>11</td>
<td>22</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>38</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>32</td>
<td>26</td>
<td></td>
</tr>
</tbody>
</table>
Specifications  Hardware Interface Specifications con't.

**Read Ports con't.**

<table>
<thead>
<tr>
<th>I/O Port</th>
<th>Bit</th>
<th>Pin</th>
<th>48 Pin</th>
<th>28 Pin</th>
<th>24 Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
<td></td>
<td>45</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>46</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>47</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td>48</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>NOT USED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>NOT USED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>NOT USED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td>NOT USED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td>NOT USED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td>NOT USED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>NOT USED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>0-7</td>
<td></td>
<td>Reads Programmed Threshold Set</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td>NOT USED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td></td>
<td>v1 Current Sense</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>v2 Current Sense</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>v3 Current Sense</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The table on the following page shows the relationship between the pins on the 48 pin socket and the pins on the other two sockets. The voltages available on those pins are also shown.

The first four columns contain pin numbers for socket #1 (6/10"). The first column relates to a 48 pin device, the second column corresponds to what a 40 pin device orientation would be in the same socket. The next three columns are the assignments that are hardware set for the pins specified. All pin # parameters are specified relative to the 48 pin socket.
<table>
<thead>
<tr>
<th>pin</th>
<th>reset</th>
<th>disable</th>
<th>set</th>
<th>alt1</th>
<th>alt2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>port 5 bit 4 0</td>
<td>port 5 bit 4 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>port 5 bit 5 0</td>
<td>port 5 bit 5 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>port 5 bit 6 0</td>
<td>port 5 bit 6 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>port 5 bit 7 0</td>
<td>port 5 bit 7 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>port 3 bit 0 0</td>
<td>port 3 bit 0 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>port 3 bit 1 0</td>
<td>port 3 bit 1 1</td>
<td>port 3 bit 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>port 8 bit 0 0</td>
<td>port 8 bit 0 1</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>port 3 bit 2 0</td>
<td>port 3 bit 2 1</td>
<td>port 3 bit 2 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>port 8 bit 1 0</td>
<td>port 8 bit 1 1</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>port 3 bit 3 0</td>
<td>port 3 bit 3 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>port 3 bit 4 0</td>
<td>port 3 bit 4 1</td>
<td>port 3 bit 4 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>port 8 bit 2 0</td>
<td>port 8 bit 2 1</td>
<td>v2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>port 3 bit 5 0</td>
<td>port 3 bit 5 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>port 4 bit 5 0</td>
<td>port 4 bit 5 1</td>
<td>port 4 bit 5 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>port 0 bit 1 0</td>
<td>port 4 bit 5 1</td>
<td>port 4 bit 5 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>port 0 bit 0 0</td>
<td>port 4 bit 5 1</td>
<td>port 4 bit 5 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>v1/1.5</td>
<td>port 4 bit 5 1</td>
<td>port 4 bit 5 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>v2</td>
<td>port 4 bit 5 1</td>
<td>port 4 bit 5 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>port 2 bit 4 0</td>
<td>port 2 bit 4 1</td>
<td>port 2 bit 4 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>port 9 bit 7 0</td>
<td>port 2 bit 4 1</td>
<td>port 2 bit 4 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>v2</td>
<td>port 2 bit 4 1</td>
<td>port 2 bit 4 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>port 1 bit 7 0</td>
<td>port 1 bit 7 1</td>
<td>port 1 bit 7 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>port D bit 7 0</td>
<td>port 1 bit 7 1</td>
<td>port 1 bit 7 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>port D bit 6 0</td>
<td>port 1 bit 7 1</td>
<td>port 1 bit 7 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pin</td>
<td>reset</td>
<td>disable</td>
<td>set</td>
<td>alt1</td>
<td>alt2</td>
</tr>
<tr>
<td>-----</td>
<td>-------</td>
<td>---------</td>
<td>-----</td>
<td>------</td>
<td>------</td>
</tr>
</tbody>
</table>
| 14  | port 1 bit 6 0 | port 1 bit 6 1  
port 9 bit 5 0 | port 1 bit 6 1  
port 9 bit 5 1 | v2 | v2 |
| 15  | port 1 bit 5 0 | port 1 bit 5 1  
port 9 bit 4 0 | port 1 bit 5 1  
port 9 bit 4 1 | v2 | v2 |
| 16  | port 1 bit 4 0 | port 1 bit 4 1  
port 9 bit 3 0 | port 1 bit 4 1  
port 9 bit 3 1 | v2 | v2 |
| 17  | port 1 bit 3 0 | port 1 bit 3 1  
port 9 bit 2 0 | port 1 bit 3 1  
port 9 bit 2 1 | v2 | v2 |
| 18  | port 1 bit 2 0 | port 1 bit 2 1  
port 9 bit 1 0 | port 1 bit 2 1  
port 9 bit 1 1 | v2 | v2 |
| 19  | port 1 bit 1 0 | port 1 bit 1 1  
port C bit 5 0  
port C bit 4 0 | port 1 bit 1 1  
port C bit 5 1  
port C bit 4 1 | v1 | v2 |
| 20  | port 1 bit 0 0 | port 1 bit 0 1  
port C bit 3 0  
port C bit 2 0 | port 1 bit 0 1  
port C bit 3 1  
port C bit 2 1 | v1 | v2 |
| 21  | port 0 bit 0 0 | port 0 bit 0 1  
port A bit 1 0  
port A bit 0 0 | port 0 bit 0 1  
port A bit 1 1  
port A bit 0 1 | v1 | v2 |
| 22  | port 0 bit 1 0 | port 0 bit 1 1  
port A bit 3 0  
port A bit 2 0 | port 0 bit 1 1  
port A bit 3 1  
port A bit 2 1 | v1 | v2 |
| 23  | port 0 bit 2 0 | port 0 bit 2 1  
port A bit 5 0  
port A bit 4 0 | port 0 bit 2 1  
port A bit 5 1  
port A bit 4 1 | v1 | v2 | clk |
<table>
<thead>
<tr>
<th>pin</th>
<th>reset</th>
<th>disable</th>
<th>set</th>
<th>alt1</th>
<th>alt2</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>port 3 bit 7 X</td>
<td>ground</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>port 0 bit 3 0</td>
<td>port 0 bit 3 1</td>
<td>port 0 bit 3 1</td>
<td>port 0 bit 3 1</td>
<td>port 0 bit 3 1</td>
</tr>
<tr>
<td></td>
<td>port A bit 7 0</td>
<td>port A bit 7 0</td>
<td></td>
<td>port A bit 7 1</td>
<td>port A bit 7 1</td>
</tr>
<tr>
<td></td>
<td>port A bit 6 0</td>
<td>port A bit 6 1</td>
<td></td>
<td>port A bit 6 0</td>
<td>port A bit 6 1</td>
</tr>
<tr>
<td>26</td>
<td>port 0 bit 4 0</td>
<td>port 0 bit 4 1</td>
<td>port 0 bit 4 1</td>
<td>port 0 bit 4 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>port B bit 1 0</td>
<td>port B bit 1 0</td>
<td></td>
<td>port B bit 1 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>port B bit 0 0</td>
<td>port B bit 0 1</td>
<td></td>
<td>port B bit 0 0</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>port 0 bit 5 0</td>
<td>port 0 bit 5 1</td>
<td>port 0 bit 5 1</td>
<td>port 0 bit 5 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>port B bit 3 0</td>
<td>port B bit 3 0</td>
<td></td>
<td>port B bit 3 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>port B bit 2 0</td>
<td>port B bit 2 1</td>
<td></td>
<td>port B bit 2 0</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>port 0 bit 6 0</td>
<td>port 0 bit 6 1</td>
<td>port 0 bit 6 1</td>
<td>port 0 bit 6 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>port B bit 5 0</td>
<td>port B bit 5 0</td>
<td></td>
<td>port B bit 5 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>port B bit 4 0</td>
<td>port B bit 4 1</td>
<td></td>
<td>port B bit 4 0</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>port 0 bit 7 0</td>
<td>port 0 bit 7 1</td>
<td>port 0 bit 7 1</td>
<td>port 0 bit 7 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>port B bit 7 0</td>
<td>port B bit 7 0</td>
<td></td>
<td>port B bit 7 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>port B bit 6 0</td>
<td>port B bit 6 1</td>
<td></td>
<td>port B bit 6 0</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>port 2 bit 7 0</td>
<td>port 2 bit 7 1</td>
<td>port 2 bit 7 1</td>
<td>port 2 bit 7 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>port C bit 7 0</td>
<td>port C bit 7 0</td>
<td></td>
<td>port C bit 7 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>port C bit 6 0</td>
<td>port C bit 6 1</td>
<td></td>
<td>port C bit 6 0</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>port 2 bit 2 0</td>
<td>port 2 bit 2 1</td>
<td>port 2 bit 2 1</td>
<td>port 2 bit 2 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>port 9 bit 0 0</td>
<td>port 9 bit 0 1</td>
<td></td>
<td>port 9 bit 0 1</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>port 4 bit 7 0</td>
<td>port 4 bit 7 1</td>
<td>port 4 bit 7 1</td>
<td>port 4 bit 7 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>port 8 bit 4 0</td>
<td>port 8 bit 4 1</td>
<td></td>
<td>port 8 bit 4 1</td>
<td></td>
</tr>
<tr>
<td>pin</td>
<td>reset</td>
<td>disable</td>
<td>set</td>
<td>alt1</td>
<td>alt2</td>
</tr>
<tr>
<td>-----</td>
<td>-------</td>
<td>------------------</td>
<td>----------------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
</tbody>
</table>
| 33  | port 2 bit 3 0 | port 2 bit 3 1  
port D bit 1 0  
port D bit 0 0 | port 2 bit 3 1  
v1 | port 2 bit 3 1  
v2 | port 2 bit 3 1  
v2 |
| 34  | port 2 bit 1 0 | port 2 bit 1 1  
port 9 bit 6 0 | port 2 bit 1 1  
v2 | port 2 bit 1 1  
v2 | port 2 bit 1 1  
v2 |
| 35  | port 2 bit 0 0 | port 2 bit 0 1  
port 8 bit 3 0 | port 2 bit 0 1  
v2 | port 2 bit 0 1  
v2 | port 2 bit 0 1  
v2 |
| 36  | port 2 bit 5 0 | port 2 bit 5 1  
port D bit 3 0  
port D bit 2 0 | port 2 bit 5 1  
v2 | port 2 bit 5 1  
v2 | port 2 bit 5 1  
v2 |
| 37  | port 2 bit 6 0 | port 2 bit 6 1  
port 8 bit 5 0 | port 2 bit 6 1  
v2 | port 2 bit 6 1  
v2 | port 2 bit 6 1  
v2 |
| 38  | port 4 bit 6 0 | port 4 bit 6 1  
port 8 bit 6 0 | port 4 bit 6 1  
v3 | port 4 bit 6 1  
v3 | port 4 bit 6 1  
v3 |
| 39  | port 4 bit 4 0 | port 4 bit 4 1 |                |               |               |
| 40  | port 4 bit 3 0 | port 4 bit 3 1  |                |               |               |
| 41  | port 4 bit 2 0 | port 4 bit 2 1  |                |               |               |
| 42  | port 4 bit 1 0 | port 4 bit 1 1  |                |               |               |
| 43  | port 4 bit 0 0 | port 4 bit 0 1  |                |               |               |
| 44  | port 3 bit 6 0 | port 3 bit 6 1  
port D bit 5 0  
port D bit 4 0 | port 3 bit 6 1  
v2 | port 3 bit 6 1  
v2 | port 3 bit 6 1  
v2 |
<table>
<thead>
<tr>
<th>pin</th>
<th>reset</th>
<th>disable</th>
<th>set</th>
<th>alt1</th>
<th>alt2</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>port 5 bit 0 0</td>
<td>port 5 bit 0 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>port 5 bit 1 0</td>
<td>port 5 bit 1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>port 5 bit 2 0</td>
<td>port 5 bit 2 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>port 5 bit 3 0</td>
<td>port 5 bit 3 1</td>
<td></td>
<td>port 5 bit 3 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>port 8 bit 7 0</td>
<td></td>
<td>port 8 bit 7 1</td>
<td></td>
<td>v3</td>
</tr>
</tbody>
</table>
Specifications con't.

### Hardware Interface Specifications

<table>
<thead>
<tr>
<th>Interface</th>
<th>min</th>
<th>max</th>
<th>units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strobe pulse widths (address, data)</td>
<td>100</td>
<td>1</td>
<td>nanoseconds</td>
</tr>
<tr>
<td>Data in stable</td>
<td>.3</td>
<td>1</td>
<td>microseconds</td>
</tr>
<tr>
<td>Data setup time</td>
<td>100</td>
<td></td>
<td>nanoseconds</td>
</tr>
<tr>
<td>Data hold time</td>
<td>10</td>
<td></td>
<td>nanoseconds</td>
</tr>
<tr>
<td>Delay from data into output strobe</td>
<td>100</td>
<td></td>
<td>nanoseconds</td>
</tr>
</tbody>
</table>

03/01/84  Omni-Programmer Technical Manual  Release 1.0
Specifications

Software Interface Specification

Under the OMNI operating system, memory is allocated as follows:

- 0103H thru 0502H  Hardware Control Code (HCC)
- 0503H thru 1602H  Programming Algorithm (PA)/User Defined PA (UDP)
- 1603H thru FFFFH  OMNI Operating System (OS)

The Programming Algorithm is the software that is normally loaded to program or read a desired device such as a 2716 EPROM. It consists of a series of instructions that jump to various address pointers in the HCC which in turn carry out hardware set-up and execution operations like "set pin 11 to voltage 1". These addresses are called Function Pointers. They are described in the following paragraphs using this format: Address in hex, Function name, functional description, and where applicable, range and typical default condition for a 2716 EPROM. When a jump to a function is performed the stack must contain appropriate information for the function: the proper sequence would be first the function pointer address, then the function parameter(s) (abbreviated "param #").

0103H resall

Reset all pins to initialization state. This function causes all pins to be reset.

0106H disall

Disable all pins; with pins in this state they can neither be written to nor read.
Specifications Software Interface Specification con't.

0109H respin

Resets the pin number in Param 1: range of pins 1-48. Reset sets the pin to 0 volts.

010CH dispin

Disables the pin number in Param 1; range of pins 1-48. Disabling a pin sets the pin voltage to 5 volts through a 10 Kohm pullup resistor.

010FH setpin

This function sets the pin number in Param 1 to its primary voltage or clock assignment; range of pins 1-48. Pin assignments can be found in the CHIPMAP found at the end of this section.

0112H alt1pin

This function sets the pin number in Param 1 to its first alternate voltage or clock assignment; range of pins 1-48 (not all pins have alternate conditions). Pin assignments can be found in the CHIPMAP.

0115H alt2pin

This function sets the pin number in Param 1 to its second alternate voltage or clock assignment; range of pins 1-48 (not all pins have alternate conditions). Pin assignments can be found in the CHIPMAP.

0127H rdpin

Reads the pin number in Param 1; range of pins 1-48. A read must be preceded by a disable to the desired pin or the read will return what the pin has been set to.

012AH setv1

This function sets the primary voltage, referenced above, to the voltage determined by: .118 volts multiplied by (X) the value in Param 1; range of Param 1 = 0 to 255, range of voltage value=0 to 28 volts. Default value for a 2716 is Param 1 = 226 (27 volts); this value is usually not changed while programming the 2716.
Specifications

Software Interface Specifications con't.

012DH setv2
This function works just like setv1 does. Its value may change many times while programming a 2716 depending on the other functions it is being used with. It is the first alternate voltage.

0130H setv3
This function works just like setv1 and setv2 do. It is used like setv2 while programming but may not be changed as much because it is not available on many pins (see CHIPMAP). It is the second alternate voltage. When using the alt2pin function on pin 33 this voltage sets pin 33 to -5 volts for use with the AMD part number 27S27.

0133H setcv1
This function sets the current limit for the primary voltage, v1, to the current determined by .004 amperes X the value in Param 1; range of Param 1 = 0-255; range of current = 0-1000 milliamperes. The default condition for the 2716 is Param 1=255.

0136H Setv2
This function works just like setcv1 for the first alternate voltage but it limits the current to 255 milliamperes and each increment has a value of .001 amperes; range of Param = 0 - 255; range of current = 0 - 255 milliamperes. The default condition for a 2716 is Param 1 = 255.

0139H setthr
This function sets the threshold voltage which is used while reading a pin to determine a high or low state. It's value is set just like setv1 where each increment represents .118 volts; range of Param 1 = 0 - 255; range of voltage = 0 - 28 volts.

0145H connect
This function is used only to see if the OMNI Programming Unit is attached to the Kaypro or other controlling processor. When this function is called a routine is started that either returns an error message or returns to the control program if no connect problems are detected.
Specifications  Software Interface Specifications con't.

014BH delay

This function creates a delay equal to the value of Param 1 X 100 microseconds. Range of Param 1 is 0-255; range of delay is 0 to 25500 microseconds. This function is used to control the amount of time a programming voltage is applied to a particular pin.

The code that becomes the UDP replaces the standard Programming Algorithm and therefore becomes the interface between the Operating System and the device for which the UDP is being written. Thus, some precautions are necessary to ensure no harm is done to the device and that the desired functions can be performed by the user. Within the section of memory known as the UDP, there are some pre-assigned locations that must be left untouched for the operating system to access them properly. The UDP can be locked at as having three sections as follows:

0503H thru 0512H  Setup Conditions
0513H thru 051FH  Device Type Definition
0520H thru 0520H  User Written Program Algorithm

The setup conditions are spaces in memory that set up the device for initialization, reading, writing, error reporting and configuration as follows:

0503H devsetup (chip index)
0506H devread (chip start address, memory start address, count)
0509H devwrite (chip start address, memory start address, count)
050CH error condition code reporting
050EH error condition address

The jumps to these addresses are done by the OS; they are filled by the OS from inputs to prompt the user answers on initialization. The parameters will be loaded onto the stack in the order listed (param1, param2, param3).
The next three bytes are also filled by the OS.

0510H number of pins on device (24 for 2714)
0511H socket used for this device (1=6/10", 2=4/10", 3=3/10")
0512H device type (1=PROM, 2=PLA, 3=PGA, 4=PAL)

The section that begins with 0513H varies in length and interpretation by device type. For PROMs and addresses contain the following information:

0513H promlen Max number of words to be addressed (for 2716-2048)
0515H promwid Word length (2716 word length = 8 bits)
0517H promblk Initial state of bits (OFFH for 2716)

For FPLAs the addresses contain the following information:

0513H plalen Number of P-terms possible
0515H plaiwid Number of input bits possible
0516H placwid Number of output bits possible
0517H plabias Blank state of output bias
0518H plaand Blank state of AND matrix
0519H placr Blank state of OR matrix
051AH plaistart Starting input number
051BH plaostart Starting output number

For PALs the addresses contain the following information:

0513H palinp Number of PAL inputs
0514H palprcd Number of PAL products
0515H palbln Number of blown links
0516H palilist Pointer to bit array of inputs
0517H palplist Pointer to bit array of products
0518H palblst Coordinate array of blown links

For FPGA (Field Programmable Gate Arrays) the addresses contain the following information:

0513H pgalen PGA length
0515H pgawid PGA width
051x tbd to be defined
## Electrical Specifications

### Voltage Supply Circuits:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>min</th>
<th>max</th>
<th>units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output voltage range</td>
<td>0</td>
<td>30</td>
<td>volts</td>
</tr>
<tr>
<td>Max output voltage</td>
<td>28</td>
<td>30</td>
<td>volts</td>
</tr>
<tr>
<td>Voltage steps</td>
<td>.118</td>
<td></td>
<td>volts/steps</td>
</tr>
<tr>
<td>Current limit (v1)</td>
<td>0</td>
<td>1.02</td>
<td>amps</td>
</tr>
<tr>
<td>(v2)</td>
<td>0</td>
<td>255</td>
<td>mA</td>
</tr>
<tr>
<td>(v3)</td>
<td>0</td>
<td>1.2</td>
<td>amps</td>
</tr>
<tr>
<td>Current limit steps (v1)</td>
<td></td>
<td>4</td>
<td>mA/step</td>
</tr>
<tr>
<td>(v2)</td>
<td></td>
<td>1</td>
<td>mA/step</td>
</tr>
<tr>
<td>Current step error</td>
<td></td>
<td>5%</td>
<td></td>
</tr>
<tr>
<td>Slew rate (v1)</td>
<td>.2</td>
<td>.3</td>
<td>v/microsecond</td>
</tr>
<tr>
<td>(v2)</td>
<td>.4</td>
<td>.5</td>
<td>v/microsecond</td>
</tr>
<tr>
<td>(v3)</td>
<td>.2</td>
<td>.3</td>
<td>v/microsecond</td>
</tr>
<tr>
<td>Current sense (v1, v3)</td>
<td>0</td>
<td>1.2</td>
<td>amps</td>
</tr>
<tr>
<td>(v2)</td>
<td>0</td>
<td>250</td>
<td>mA</td>
</tr>
</tbody>
</table>

### Socket pin switches:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>min</th>
<th>max</th>
<th>units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current on v2 switches</td>
<td>0</td>
<td>.5</td>
<td>amp</td>
</tr>
<tr>
<td>Current on v1, v3 switches</td>
<td>0</td>
<td>3</td>
<td>amps</td>
</tr>
<tr>
<td>Saturation voltage change v1, v3</td>
<td>.2</td>
<td>.4</td>
<td>volts</td>
</tr>
<tr>
<td>(minimum current to max current) v2</td>
<td>.2</td>
<td>.3</td>
<td>volts</td>
</tr>
<tr>
<td>(GND)</td>
<td>.1</td>
<td>.3</td>
<td>volts</td>
</tr>
<tr>
<td>Current limit on gnd switch</td>
<td>6</td>
<td>15</td>
<td>mA</td>
</tr>
<tr>
<td>Slew rate</td>
<td>20</td>
<td>40</td>
<td>v/microsecond</td>
</tr>
<tr>
<td>-5v circuit current</td>
<td>-100</td>
<td></td>
<td>mA</td>
</tr>
</tbody>
</table>

03/01/84 Omni-Programmer Technical Manual Release 1.0
Specifications

Electrical Specifications con't.

Voltage Calibration Circuit

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference voltage</td>
<td>9.95</td>
<td>10.05</td>
<td>volts</td>
</tr>
<tr>
<td>Threshold voltage</td>
<td>0</td>
<td>30.1</td>
<td>volts</td>
</tr>
<tr>
<td>Threshold voltage error</td>
<td></td>
<td>.1%</td>
<td></td>
</tr>
<tr>
<td>Threshold voltage steps</td>
<td>.118</td>
<td></td>
<td>volts/step</td>
</tr>
</tbody>
</table>
Specifications

Mechanical Specifications

Overall Dimensions  16" W x 11" D x 5" H

Table space required  16" W x 14" D

Weight  8 lbs.

Operating Environment:
Temperature  32 - 95° F
             0 - 35° C
Humidity  20%-80% relative
Voltage  110VAC ± 10%
Power Requirements  50 watts
User Specified Programming Algorithms

The Omni-Programmer algorithm area is ordinarily used for loading the software algorithm used to program a particular group of chips. The Omni user interface and operating system uses the index file OMNI.IDX to find the program which should be loaded into the available algorithm space. The available space is only 4000 bytes so any user program must be no larger than that.

The user algorithm can be written in any language. It must adhere to the software interfaces specified in the previous section, be absolute binary Z80 machine code targeted for a bottom address of 503 (Hexadecimal), and reside in a file with the first byte of the file being the first byte of the program. If the filename is then placed appropriately in OMNI.IDX, the user algorithm will execute when called.

Varix programming algorithms are all written in the high level language "C". In order to improve the speed of programming, the innermost loops have often been replaced with assembly language assist routines. Functions like incrementing to the next chip address and assembling the data output into a single word are examples of these situations.

There are two basic approaches to writing device algorithms used by Varix Corporation. The first uses calls to the hardware interface routines described in the previous section to set each pin on the chip in the desired manner. EXAMPLE: Placing a 10 bit address on the address bus of a PROM requires a sequence of 10 calls to either DISPIN or RESPIN dependent on whether the pin is TTL high or TTL low. While the resulting code is very straight forward, it is also somewhat slow.

The second approach involves a combination of high level code and tables which describe the chip. In the above example with a 10 bit address, a table would be created which listed the chip pins making up the address bus. A single call to MAPWORD would then cause the data pattern to be placed on the appropriate pins.

Once the code and tables are created, the following system sequence should be used to create the programming algorithm.

A>cii -m -X300 b:<filename>.c
A>m80 =b:<filename>.asm
A>180 /p:503, hprcm, b:<filename>, cmnlib/s, libc/s, b:<filename>/n/x/e
A>zsid
   *i<filename>.hex
   *r-403
   */
A> SAVE 16 B:<filename>.000

03/01/84 Omni-Programmer Technical Manual Release 1.0
User Specified Programming Algorithms con't.

The link process (L80) will complete with a message in the following format:

```
[0 503 1439]
```

if the third number exceeds 1503, your programming algorithm is too large and must be reduced in size.

M80 and L80 are the Z80/8080 macro assembler and linker from Microsoft, Inc. CII is the 'Aztec C Compiler from Manx Software Systems, Inc. ZSID is the symbolic debugger from Digital Research, Inc.

Due to the stand alone nature of the programming algorithm module, it is possible to use any compiler, assembler, linker so long as the interface specifications are adhered to. A development package can be purchased from Varix which contains the development software customarily used by Varix along with command files.

Once the algorithm is written, it must be listed in the OMNI.IDX file in order to be invoked. The format of the file is shown on the following page. Typically, only the filename with a pound sign in front followed by the name (such as chip device number) with which you wish to invoke the file. The name must be preceded by a zero, which is ordinarily used as an index into a programming algorithm which may program several chips. EXAMPLE: To invoke the file CALIB.OOO by typing CALIBRATE under the Omni-Programmer software add the following entries to OMNI.IDX:

```
#CALIB.OOO
OCALIBRATE ; Special note: all invocation names must use upper case letters.
```

The entire OMNI.IDX file looks like:

(see following page)
; Omni-Programmer Index File 10/5/83
;
; Special characters are:
; # Family name (general device type)
; @ User interface filename
; % Command table filename
; : Personality module filename
; ; Comment line (blank lines are comments also)
;
; A personality name is preceded by the index (in hex) of its module
; in the personality file.

;-------------------------- Proms --------------------------
#EPROM
@OMPROM.OVR
%OMPROM.TBL
:OMEPRM.000
02516 12532 42564 02716 22732 32732A 52764
:OMEPRM.001
027128 127256
:OMEPRM.002
068732-0
168732-1
268764 268766

;-------------------------- Gang-programmer Proms --------------------------
#GANG

@OMGANG.OVR
%OMGANG.TBL

:OMGPRM.000
02716 12732 22732A
:OMGPRM.001
02764 127128 227256
It is valuable to note that the OMNI.IDX file can be modified by using any editor to change the names of the chips which must be typed in for programming. Suppose, for example, that one wished to use internal 10 digit part numbers to identify chips. A simple change from 02716 12732 to 0300-0462-001 1300-0462-002 would cause the Intel 2716 and 2732 chips to be called up by their 10 digit number. You can even put both forms in if desired. If you put:

```
#TEPROM.001
02716 0300-0462-001 02KEPROM 0A
```

into your OMNI.IDX file, then entering any of the four names, 2716, 300-0462-001, 2KEPROM, or A, would result in programming a 2716.
User Modifications

USER SPECIFIED ALGORITHMS

User specified algorithms METHOD I:

(see following pages)

TEPROM1.C
METHOD I Example:

User Modifications

```c
#include "a:stdio.h"
#include "a:config.h"

/* production version */

/* Setup the function pointers for the hardware interface routines */

#include "a:omhwd.h"

/* set up variables to hold pin numbers */

int ce;
int oe;
int vpp;
int pgm;
int p0pgm;
int c0pgm;
int cevpp;
int ct1;
int ct2;
int vcc;
int vcca;

/* address pin maps */

int maxaddr, /* maximum address pin number */

/* 2516/2716 address pins */

/* 2532 address pins */
char amap2[12] = {20, 19, 18, 17, 16, 15, 14, 13, 33, 34, 31, 30};

/* 2732/2732a address pins */
char amap3[12] = {20, 19, 18, 17, 16, 15, 14, 13, 33, 34, 31, 30};

/* 2564 address pins */

/* 2764 address pins */
char *adrtab;

/* define voltages needed by the eproms */
```
# define VCC 46
# define VIH 46
# define VPP1 211
# define VPP2 179
# define THRESH 17
# define CHPCCH 25
# define CURR1 255
# define CURR2 255

int vccv;
int vihv1;
int vihv2;
int vpp1v1;
int vpp1v2;
int vpp2v1;
int vpp2v2;

#define V1 1
#define V2 2
#define V3 3

/*
   define the run and reset buttons
*/
#define RUNBUTTON 42 /* run button */
#define RSBUTTON 43 /* reset button */

/*
   define the programming intervals
*/
#define TP1 3 /* .3 milliseconds */
#define TP2 9 /* .9 milliseconds */
#define TP3 36 /* 3.6 milliseconds */
#define TP4 500 /* 50.0 milliseconds */

/*
   set up led bits
*/
#define LEDC 0x00
#define PROGRAM 0x20
#define PGERR 0x10
#define SKT3 0x04
#define SKT2 0x02
#define SKT1 0x01

/*
   temporary variables for code optimization
*/
int setup = NO; /* successful devsetup flag */
int rtcode;  // function return code
int status;  // procpin status temp
int address; // chip address
int cmdaddr; // last address accessed
int tempaddr; // temporary address
int index;  // for loop index
int gotdata; // temp for verify
int data;  // data byte
char $buffer; // pointer into data buffer
int saddr; // temp for addr parameter
char $auxbuf; // temp for auxbuf parameter
int slength; // temp for length parameter
int stp; // current programming interval
int err;   // error programming for 1 msec
int errb;  // error programming for 3 msec
int tdvnb = 0; // temp for device number index
int mask; // address bit mask
int (*dri)(); // pointer to devread function
int (*dwe)(); // pointer to devwrite function
int (*pvn)(); // pointer to progverf function
int (*cod)(); // pointer to check data funct.

/*
set up pointer to device error number and configuration table
*/
int *deverror = 0x50c;
int *erraddr = 0x50d;
CONFIG *config = 0x510;

/*
devsetup -- initial hardware setup

entry: devnbr -- the index number of theeprom
        0 -- 2516/2716
        1 -- 2532
        2 -- 2732
        3 -- 2732u
        4 -- 2564
        5 -- 2764

exit: return the following:
       ERROR -- omni programmer not connected or devnbr out
       of range
       OK -- no problems

algorithm: initialize the hardware
           disable all pins
           check devnbr to make sure it is in range
           set up the voltages for the target eeprom
           map the pins to their socket relative values
           return

notes: all pins are disabled on exit
*/
int devsetup (devnbr) {
    int devnbr;
    register int i;
    int dr2716 ();
    int dw2716 ();
    int pv2716 ();
    int cd2716 ();
    int dr2532 ();
    int dw2532 ();
    int pv2532 ();
    int cd2532 ();
    int dr2732 ();
    int dw2732 ();
    int pv2732 ();
    int cd2732 ();
    int dw2732 ();
    int dr2564 ();
    int dw2564 ();
    int pv2564 ();
    int cd2564 ();
    int dr2764 ();
    int dw2764 ();
    int pv2764 ();

    tdevnbr = devnbr;

    (*init) ();
    (*disall) ();
    *deverror = 0;
    *erraddr = 0;
    rtncode = OK;
    setup = NO;
    (*setcv1) (CURRV1);
    (*setcv2) (CURRV2);

    /*
     * is anybody out there?
     */
    if (!(connect) ()) {
        *deverror = CNCTERR;
        rtncode = ERROR;
    } else {
        (*ledon) (CHECK);
    }

    (*ledoff) (PROGRAM : PGERR : SKT1 : SKT2 : SKT3);

    /*
     * make sure the socket is empty
     */
if (rtnode == OK) {
  ifsetthr) (CHPCHK);
  if (!empty()) {
    *deerror = NOTEMPTY;
    rtnode = ERROR;
  }
}

/*
 set up voltages for the target device
 */
if (rtnode == OK) {
  vccv = calibrate (VCC, V3);
  vihv1 = calibrate (VIH, V1);
  vihv2 = calibrate (VIH, V2);
  vppiv1 = calibrate (VPP1, V1);
  vppiv2 = calibrate (VPP1, V2);
  vpp2v1 = calibrate (VPP2, V1);
  vpp2v2 = calibrate (VPP2, V2);
}

switch (tdevnbr) {
case 0:
  case 1:
  case 2:
  case 3:
    config->pincont = 24;
    break;
  case 4:
  case 5:
    config->pincont = 26;
    break;
  default:
    *deerror = PARMERROR;
    rtnode = ERROR;
}

/*
 map the pins to their socket relative values
 */
switch (tdevnbr) {
case 0:
  ceqgm = 30; /* pin 18 */
  ca = 32; /* pin 20 */
  cwp = 33; /* pin 21 */
  vcc = 36; /* pin 24 */
  adrtab = amap1;
config->cinfo.cprom.promlen = 2048;
mask = 0x07ff;
dr = dr2716;
dw = dw2716;
pv = pv2716;
cd = cd2716;
break;
case 1:
    pdpgm = 32;  /* pin 20 */
    vpp = 33;  /* pin 21 */
    vcc = 36;  /* pin 24 */
    adrtab = amap2;
config->cinfo.cprom.promlen = 4096;
mask = 0x0fff;
dr = dr2532;
dw = dw2532;
pv = pv2532;
cd = cd2532;
break;
case 2:
case 3:
    csppgm = 30;  /* pin 18 */
    csrvpp = 32;  /* pin 20 */
    vcc = 36;  /* pin 24 */
    adrtab = amap3;
config->cinfo.cprom.promlen = 4096;
mask = 0x0fff;
dr = dr2732;
pv = pv2732;
cd = cd2732;
if (downbr == 2)
    dw = dw2732;
else
    dw = dw2732;
break;
case 4:
    vpp = 11;  /* pin 1 */
    cs1 = 12;  /* pin 2 */
    pdpgm = 32;  /* pin 22 */
    vcc = 36;  /* pin 26 */
    cs2 = 37;  /* pin 27 */
    vcc = 38;  /* pin 28 */
    adrtab = amap4;
config->cinfo.cprom.promlen = 8192;
mask = 0x1fff;
dr = dr2564;
dw = dw2564;
pv = pv2564;
cd = cd2564;
break;
case 5:
    vpp = 11;  /* pin 1 */
    cs = 30;  /* pin 20 */
    cs = 32;  /* pin 22 */
pgm = 37;  /* pin 27 */

40
```c
vcc = 3.0; /* pin 28 */
adrtab = amap5;
config->cinio subparagraphs = 8192;
mask = 0x1fff;
dr = dr2764;
dw = dw2764;
pv = pu2764;
cd = cd2716;
break;
}

if (rcode != OK) {
    (void) (PGERR);
} else {
    (void) (SKT1);
    setup = YES;
}

return (rcode);
}

/*
deuread - read the 8 bit byte addressed
entry: addr - the address of the byte to read
exit: return the following:
ERROR - pin number out of range or invalid parameter
the value of the addressed byte if successful
algorithm: check the range of addr, if out of range
power on the chip
apply the address to the address pins
read the output pins
return the value read from the output
notes: it is assumed that the pins are disabled on entry,
all pins are disabled on exit.
*/

int deuread (addr, auxbuf, length)
int addr;
char *auxbuf;
int length;
{
    int readbyte ();
    saddr = addr;
    sauxbuf = auxbuf;
slength = length;
    /*
    check to see if we are reading the gang buttons
    */
```


```c
/*
 * if (!(*connect) ()) {
 *     if (saddr == RUNBUTTON) {
 *         if (((*rdpin) (RUNBUTTON) == NO))
 *             return (YES);
 *         else
 *             return (NO);
 *     }
 *     if (saddr == RESETBUTTON) {
 *         if (((*rdpin) (RESETBUTTON) == NO))
 *             return (YES);
 *         else
 *             return (NO);
 *     }
 * }
 */

/*
 * are we ready to go?
 */

if (promsetup () < 0) {
    (*ledon) (PGERR);
    return (ERROR);
}

(*disall) ();  
pracpin (readbyte);

(*disall) ();  
return (slength);
}

int promsetup ()
{
/
 * is the hardware set up?
 */

if ((setup == NO) || (*getthr ()) == 0) {
    if (devsetup (tdevnbr) < 0)
        return (ERROR);
}

/*
 * is anybody out there?
 */

if (!(*connect) ()) {
    *deverror = CNCTERR;
    return (ERROR);
}

(*ledoff) (PGERR);
```
is the chip in the socket and in the right place?

```c
static
int dr2716 ()
{
    (#alt2pin) (vcc);
    (#setpin) (vpp);
    (#respin) (vppm);
    (#respin) (oe);
    return;
}

static
int dr2532 ()
{
    (#alt2pin) (vcc);
    (#setpin) (vpp);
    (#respin) (pdpmm);
    return;
}

static
int dr2732 ()
{
    (#alt2pin) (vcc);
    (#respin) (cmpm);
    (#respin) (oevpp);
    return;
}

static
int dr2564 ()
```
{*
  (*setpin) (vcc);
  (*alt2pin) (vcc);
  (*alt2pin) (ups);
  (*respin) (cs1);
  (*respin) (cs2);
  (*respin) (pdp);
  return;
}

static
int dr2764 ()
{
  (*setpin) (vcc);
  (*setpin) (ups);
  (*respin) (ce);
  (*respin) (ae);
  return;
}

/*
 * devwrite - write a value to a 8 bit byte
 * entry:  data - the value of the byte to write
 *         addr - the address of the byte to write
 * exit:   return the following:
 *         ERROR - invalid parameter or unsuccessful write
 *         data - return the data if successful
 * algorithm:
 * notes:  it is assumed that all pins are disabled on entry.
 *         all pins are disabled on exit.
 */

int devwrite (addr, auxbuf, length)
int addr;
char *auxbuf;
int length;
{
  saddr = addr;
  sauxbuf = auxbuf;
  slength = length;
  if (promsetup () < 0) {
    (*ledon) (PGERR);
    return (ERROR);
  }
}
static (PRGRAM);

(*d)();

(*disall)();
(*ledoff) (PRGRAM);

if (rtncode < 0) {
    %deerror = PROGERR;
    (%ledon) (PGERR);
}
else
    %deerror = 0;

return (rtncode);
}

static int dw2716 ()
{
    int progpin ();

    (%setv1) (vppv1);
    (%respin) (cepgm);
    (%alt2pin) (vcc);
    (%setpin) (vpp);

    rtncode = procpin (progpin);

    (%setv1) (vihv1);
    (%dispin) (vpp);
    return;
}

static int dw2532 ()
{
    int progpin ();

    (%setv1) (vppv1);
    (%alt2pin) (vcc);
    (%setpin) (vpp);

    rtncode = procpin (progpin);

    (%setv1) (vihv1);
    (%dispin) (vpp);
    return;
}

static int dw2732 ()
{
    register int i;
}
int progpipn ();

(*resall) ();
for (i = 0; i <= vpp1v2; i++) {
    (*delay) (2);
    (*setv2) (i);
}
(*resall) ();
(*alt2pin) (vcc);
(*setpin) (oevpp);
rtncode = progpipn (progpipn);
(*dispin) (oevpp);
(*setv2) (vihv2);

static
int dwa2732 ()
{
    register int i;

    int progpipn ();

    (*resall) ();
    for (i = 0; i <= vpp2v2; i++) {
        (*delay) (2);
        (*setv2) (i);
    }
    (*resall) ();
    (*alt2pin) (vcc);
    (*setpin) (oevpp);
    rtncode = progpipn (progpipn);
    (*dispin) (oevpp);
    (*setv2) (vihv2);
    return;
}

static
int dw2564 ()
{
    int progpipn ();

    (*setv1) (vpp1v1);
    (*setpin) (vcc);
    (*alt2pin) (vccal);
    (*setpin) (vpp);
    (*respin) (cs1);
    (*respin) (cs2);
    rtncode = progpipn (progpipn);

    int progpipn ();

    (*resall) ();
    for (i = 0; i <= vpp1v2; i++) {
        (*delay) (2);
        (*setv2) (i);
    }
    (*resall) ();
    (*alt2pin) (vcc);
    (*setpin) (oevpp);
    rtncode = progpipn (progpipn);
static int dw2764 ()
{
    int progpin ();

    (*setv) (vpp2v1);
    (*setpin) (vcc);
    (*setpin) (vpp);
    (*respin) (ce);

    rtncode = procpin (progpin);

    (*setv) (vihv1);
    (*dispin) (vpp);
    return;
}

int procpin (function)
int (*function) ()
{

    oldaddr = 0xFFFF;
    address = saddr;
    buffer = sauxbuf;

    for (index = 0; index < slength; index++) {
        addrinc ();

        if (((function) () < 0) {
            *erraddr = address;
            return (ERROR);
        }

        oldaddr = address;
        address++;    buffer++;
    }
    return (slength);
}

static int progpin ()
{

    (*cd) ();
    if (gotdata == *buffer)
return (OK);
stp = TP1;
erra = progverf ();
stp = TP2;
errb = progverf ();
if (erra == NO) {
    stp = TP3;
    progverf ();
}
if (errb == NO) {
    stp = TP4;
    if (progverf () == NO) {
        *deverror = PROGERR;
        return (ERROR);
    }
}
return (OK);

int progverf ()
{
    (*pv) ();
    if (gotdata != *buffer)
        return (NO);
    else
        return (YES);
}

static int pv2716 ()
{
    data = *buffer;
dataiset ();
    (*dispin) (cepgm);
    (*delay) (stp);
    (*respin) (cepgm);
dataidis ();
cd2716 ();
}

static int cd2716 ()
{
    (*respin) (oe);
gotdata = dataiset ();
    (*dispin) (oe);
}
static
int pv2532 ()
{

data = %buffer;
dataiset ();
(*respin) (pdpgm);
(*delay) (stp);
(*dispin) (pdpgm);
dataidis ();
cd2532 ();
}

static
int cd2532 ()
{
(*setvl) (vihvl);
(*delay) (9);
(*respin) (pdpgm);
gotdata = dataiset ();
(*dispin) (pdpgm);
(*setvl) (upplvl);
}

static
int pv2732 ()
{

data = %buffer;
dataiset ();
(*respin) (cepgm);
(*delay) (stp);
(*dispin) (cepgm);
dataidis ();
cd2732 ()
}

static
int cd2732 ()
{
(*respin) (cevpp);
(*respin) (cepgm);
}
gotdata = datalget();
(*dspin) (cwpgm);
(*sctpin) (nervpp);
}

static int pv2564 ()
{

data = $buffer;

datalset();

(*respin) (pdpgm);
(*delay) (stp);
(*dspin) (pdpgm);

data1dis();

cd2564();
}

static int cd2564 ()
{

(*setv1) (vihv1);
(*delay) (9);
(*respin) (pdpgm);
gotdata = datalget();
(*dspin) (pdpgm);
(*setv1) (vpp1v1);
}

static int pv2764 ()
{

data = $buffer;

datalset();

(*respin) (pgm);
(*delay) (stp);
(*dspin) (pgm);

data1dis();

cd2716();
}
static int readbyte ()
{
    #buffer = dataget ();
    return (OK);
}

/* End of tepromi.c */
User Modifications

USER SPECIFIED ALGORITHMS

User specified algorithms METHOD II:

(see following pages)

MMITWA.C/MMITWA.TAB
User Modifications

METHOD II Example:

```c
#include "atxtdia.h"
#include "a.config.h"

/* Set up the function pointers for the hardware interface routines */

#include "a.menu.h"

/* set up variables to hold pin numbers */

int v0,
int v1,
int v2,
int vcc,

/* tables for chip programming */

#include "a.menu.tab"

/* define voltages needed by the prom */

#define VIL 0    /* 0 volts */
#define VCC 4.5  /* 5.31 volts */
#define VDDP 10  /* 11.22 volts */
#define VOUT1 17  /* 11.45 volts */
#define VOUT2 25  /* 12.75 volts */
#define THRESH 17  /* 2.01 volts */
#define CHECK 25  /* 2.35 volts */
#define CURRUL 255 /* maximum current limit */
#define CURRUL2 255 /* maximum current limit */

int vil;
int vcc1;
int vcc1p;
int vout1;
int vout2;
```
#define V1   1
#define V2   2
#define V3   3

/*
define the programming intervals
*/
#define ID   4  /* 400 micro seconds */

/*
set up led bits
*/
#define CHEST 0x00
#define PROGRAM 0x20
#define PGEND 0x10
#define SKT2 0x04
#define SKT1 0x02
#define SKT 0x01
#define MAXTRY 10

/*
temporary variables for code optimization
*/

int setup = NO;  /* successful device setup flag */
int dbit;      /* data bit */
int stcode;    /* function return code */
int status;    /* proc pin status temp */
int addr;      /* chip address */
int oldaddr;   /* last address accessed */
int tempaddr;  /* temporary address */
int index;     /* for loop index */
int count;     /* temp for verify */
int data;      /* data byte */
char *buffer;  /* pointer into data buffer */
int addr;      /* temp for addr parameter */
char *auxbuf;  /* temp for auxbuf parameter */
int length;    /* temp for length parameter */
int devnumbr = 0;  /* temp for device number index */
int mask;      /* address bit mask */
char *aptr;    /* pointer into data pin array */
char *apaddr;  /* pointer to address string */
char *datatri; /* pointer to data string */
char *prepra;  /* pointer to pre-read string */
char *postra;  /* pointer to post-read string */
char *wptrs;   /* pointer to write string */
char *rddata;  /* pointer to read data */
char *offset;  /* pointer to offset in write string */
int proswid;   /* width of prosw */

/*
set up pointer to device error number and configuration table
*/
int deverror = 0x50c;
int erraddr = 0x50c;
CONFIG config = 0x610;

/*
 * devsetup - initial hardware setup
 * entry: devnbr - the index number of the eprom
 * exit: return the following:
 *        ERROR - mii program not connected or devnbr out of range
 *        OK - no problems
 * algorithm: initialize the hardware
 *            disable all pins
 *            check devnbr to make sure it is in range
 *            set up the voltages for the target eprom
 *            map the pins to their socket relative values
 * notes: all pins are disabled on exit
 */

int devsetup (devnbr)
int devnbr;
{
    register int i;

devnbr = devnbr;

    /*init());
    (*disall))();
    *deverror = 0;
    *erraddr = 0;
    rncode = OK;
    setup = NO;
    (**tcvi) (CURRU1);
    (**tcvu2) (CURRU2);

    /*
    in anybody out there?
    */

    if (!(*connect)()) {
        *deverror = CNCTERR;
        rncode = ERROR;
    }
    else {
        (*ledon) (CHECK);
    }

    return 0;
/*
  make sure the socket is empty
*/

if (trancode == OK) {
  (@setthr) (CHPOHK);
  if (!empty ())) {
    #deverror = NOTEMPTY;
    trancode = ERROR;
  }
}

/*
  set up voltages for the target device
*/

if (trancode == OK) {
  vIL = VIL;
  vech = calibrate (VCC, V9);
  vccp = calibrate (VCCP, V9);
  vout2 = calibrate (VOUT2, V2);
  vout1 = calibrate (VOUT1, V1);

  (@disable) ();
  (@set1) (vech);
  (@setv2) (vech);
  (@setv3) (vech);
  (@settar) (THRESH);
}

/*
  map the pins to their socket relative values
*/

addr = $addrlist[devnbr][03];
data = $datalist[devnbr][02];
uprds = $uprds[devnbr][03];
pwrs = $pwrs[devnbr][02];
porta = $porta[devnbr][03];
portb = $portb[devnbr][03];
config->pinent = numpins[devnbr];
config->info.prom.prolen = prilen[devnbr];
config->info.prom.promid = promid = prwld[devnbr];
config->socket = socsize[devnbr];
config->info prom promblk = 0;
if (devnbr == 0) {
  %(pwr1 + %(pwr3+1)) = vout2;
}
else {
  %(pwr1 + %(pwr3+2)) = vout1;
}
#(score | (score)) = work;
*(score | (score)) = work;
*(score | (score)) = work;

if (tracode = OK) {
    (*l-en) {PCERR};
}
else {
    return = YES;
    if (socksize[dmnbr] = 1)
        (*l-en) {SKT1};
    else
        (*l-en) {SKT3};
}

return (tracode);

/*

derv - read the 8 bit byte addressed
entry: addr - the address of the byte to read
exit: return the following:
    ERROR - pin number out of range or invalid parameter
    the value of the addressed byte if successful
algorithm:
    check the range of addr, if out of range
    return ERROR
    power on the chip
    apply the address to the address pins
    read the output pins
    return the value read from the output

notes: it is assumed that the pins are disabled on entry,
    all pins are disabled on exit.

*/

int deriv (addr, auxbuf, length)
int addr;
char *auxbuf;
int length;

int data;

addr = addr;
auxbuf = auxbuf;
length = length;

/*
are we ready to go?
*/

if (prosetup () < 0) {
    (*l-en) {PCERR};

57
return (ERROR);
}
addr = 0xFFFE;
stream(pwords);
for (addr = addr; addr < (addr+length); addr++) {
    mapword(0, addr, naddr);
data = assemble(porddata);
    *nbuf++) = data;
    addr += length;
}
stream(pwords);
Github() ;
return (length);

int browse_setup ()
{
    // is the hardware set up?
    if (1) {
        if (devset up (devnum) (C) )
            return (ERROR);
    }
    // is anybody out there?
    if (! !connect) () {
        devarror = CHKERR;
        return (ERROR);
    }
    #edef (PCERR);
    #edef (CHERR);
    // is the chip in the socket and in the right place?
    if (! !setthr (CHPCHK) )
        if (empty ()))) {
            devarror = NOCHIP;
            return (ERROR);
        }
    if (! !sethpia ( voc) )
        devarror = ORIENTATION;
        return (ERROR);
    #edef (THRESK);
/*
** dewrite - write a value to a 32 bit byte
**
** entry: data -- the value of the byte to write
**        addr -- the address of the byte to write
**
** exit: return the following:
**        ERROR - invalid parameter or unsuccessful write
**        data -- return the data if successful
**
** notes: it is assumed that all pins are disabled on entry.
**        all pins are disabled on exit.
**
*/

int dewrite (addr, sdbuf, length)
{
    int data;
    register tthr;
    int nstri;
    int promdata;
    int i;

    saddr = addr;
    sdbuf = sdbuf;
    length = length;
    rtcode = length;

    /* are we ready to go? */

    if (promsetup () < 0) {
        ($ledon) (PS ERR);
        return (ERROR);
    }

    ($ledon) (PR G RAM);

    /* this is where the write algorithm goes */

    oldaddr = 0xFFFE;
    for (saddr = addr; saddr < addr + length & rtcode > 0; saddr++) {
        mapword(oldaddr, saddr, paddr);
        oldaddr = saddr;
        stream (ppreads);
    }

    return (OK);
}
promdata = assemble(prddata);
stream (opords);
data = waitbuf++. shfr = 1;
if(data != promdata)
for (i=0; i<promwid; i++)
if (data & shfr) {

copyid((opdata + i*6), (oprs + i*(pwr*3)));
copyid((opdata + i*6 + 1), (oprs + i*(pwr + 1)));
for (nchr = 0; (opdata & shfr) & nchr < MAXTRY
stream(purs);
stream(ppords);
promdata = assemble(prddata);
stream(ppords);
if (promdata & shfr) {
stream(purs);
send();
send();
send();
send();
}
}
shfr = shfr << 1;
/*set3() (vcch);*/
stream (opords);
promdata = assemble(prddata);
stream (opords);
if (promdata != data) {
erraddr = cuaddr;
rcode = ERROR:
}
/*disall();*/
/* End of write algorithm */
/*loadfl (PROGAR);*/
if (rcode <> 0) {
#deerror = PROCERR;
#ldegen (PGERR);
}
else
#deerror = 0;
return (rcode);
/* End of template */
METHOD II Example con’t.

```c
/* FILE = MMITWn24D */

/* Half Tick pin table */

/* Device       Size       Socket reg num pin */
/* number */

0 633141 256 x 4 3 n 16
1 633241 112 x 4 3 n 16
2 633341 2048 x 4 3 n 16
3 633441 4096 x 4 3 n 20

static int sockets[4] = {3,3,3,3};
static int priens[4] = {256,512,2048,4096};
static int prsiz[4] = {4,4,4,4};

static char prers[4][103] = {
0x00,0x00,0xff,0x00,0x00,0xff,0x01,0x00,0xff,0xff00,
0x00,0x00,0xff,0x00,0x00,0xff,0x01,0x00,0xff,0xff00,
0x00,0x00,0xff,0x00,0x00,0xff,0x01,0x00,0xff,0xff00,
0x00,0x00,0xff,0x00,0x00,0xff,0x01,0x00,0xff,0xff00,

};

static char pers[4][103] = {
0x00,0x00,0xff,0x00,0x00,0xff,0x01,0x00,0xff,0xff00,
0x00,0x00,0xff,0x00,0x00,0xff,0x01,0x00,0xff,0xff00,
0x00,0x00,0xff,0x00,0x00,0xff,0x01,0x00,0xff,0xff00,
0x00,0x00,0xff,0x00,0x00,0xff,0x01,0x00,0xff,0xff00,

};

static char wers[4][6] = {
3,9,14,24,32,35),
(3,6,11,21,29,33),
(3,6,11,21,29,32),
(3,7,14,24,32,35),

};

static char wersr[13][30] = {
0x00,0x00,0xff,0x00,0x00,0xff,0x01,0x00,0xff,0x012,0xff,0x100,0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xff,0x0xf
/* NOTE 4551642 USES V2 FOR PROGRAMMING VOLTAGE Vpp */

static char rdata[4][9] = {
(4.0, 0x02, 0x01, 0, 0x01, 0, 0x04, 0, 0x40),
(4.0, 0x02, 0x01, 0, 0x04, 0, 0x04, 0, 0x40),
(4.0, 0x01, 0, 0x04, 0, 0x40, 0, 0x10),
(4.0, 0x04, 0, 0x40, 0, 0x10, 1, 0x04),
};
Assembly Language Assist Routines

For input to Stream

Production Assembly Routine Stream

; This routine takes a table of pin manipulation information which is
; very timing sensitive and massages it in the background so that it can
; be streamed to the hardware in its most efficient form.

; The input stream must be pointed to by the first parameter on the stack
; and be in the following format:

; port address   -- address of the omni port to be changed
; set mask       -- ones are bits to be set in that port
; clr mask       -- zeros are bits to be reset in that port
; .              -- repeat as many times as needed
; . . . . . . . .
; . . . . . . . . . . . . . . . . .
; OXFF            -- end string with all ones

; The destination string will start at the local location "pumpdata" and
; will be in the following format:

; length         -- length of this data stream
; port address   -- port to be modified
; port data      -- data to be put at port
; .              -- repeat to length specified in first byte

; The algorithm executes as follows:

; calling sequence    stream(ptr to stream)

; get port address whose data is to be modified
; get current data at that port from icostate
; modify bits indicated by set mask and clr mask
; put port address into pump string
; put desired data into pump string
; repeat until zero encountered in stream
; put count into first byte of pump string
; call pump to send the pump string out at high speed
; return

; ON EXIT, register bc preserved; hl, de destroyed

; Get the addresses for the hardware interface routines

03/01/84       Omni-Programmer Technical Manual       Release 1.0

64
METHOD II con't.

For input to Mapword

Mapword

; Mapword takes an 16 bit word and a map table and efficiently maps the bits onto the pins indicated by the map table
; the map table looks like this:

; #entries the number of entries in this word
; port ± hardware change if bit 0 of mapword is 0
; setmask > hardware change if bit 0 of mapword is 1
; clrmask / repeat for maximum number of mapped bits
; port ±
; setmask >
; clrmask /
;
;
; there are three entry parameters for mapword which are on the stack as per 'C' conventions.

; old word - integer containing the previous value mapped by mapword
; this is used to optimize mapword to modify only those bits which are different from one time to the next. If data is to be written for the first time, old word needs to equal the exclusive or of map word.

; map word - integer value containing bit pattern to be mapped

; pointer to map table - pointer to list of masks to implement hardware changes to make under various conditions

; mapword calling sequence is:

; mapword(old word, map word, pointer to map table);
; on return, the Omni pins will be set up with the new data pattern;
; there are no return codes

; tmpstream: db 0ffh
    ds 48
METHOD II con't.

For input to Mapword con't.

; The algorithm uses both register sets to promote efficiency, and after
; the setup code, the following registers are used:
; de = the differences between the old data and the new data
; hl = the new data
; a = the offset from the current position of hl' to current bit position
; hl' = pointer to the current position in the map table
; de' = pointer to current position in the temporary stream data
; b' = a temporary offset holding register
For data collection from Assemble

; Assemble Word
;
; This routine takes a list of port and bit mask assignments and
; assembles a word of data from them based on the state of the pins the
; list maps to.
;
; The assemble list has the following format:
;
; count
; port ± port address of the MSB to be read
; bit mask / position in port of bit to read
; . ± repeat two entries at a time up to 8
; . / ending with the least significant bit
;
; The calling sequence for assemble is:
;
; data = assemble(rddatalst);
;
; on entry:
; rddatalst is a pointer to the table shown above
;
; on exit:
; data contains the assembled data
PORTING OMNI SOFTWARE TO A NEW HOST MICROCOMPUTER

Omni software is targeted for the Kaypro II or IBM PC. The procedure that follows is required for operating the Omni Programmer on a host machine other than those two mentioned. It is conservatively estimated to be a 2 week effort to completely modify and test the code. Varix will assist with technical support as necessary to answer questions.

REQUIREMENTS:

1. The processor of the target host must be Z-80 or 8088 and the operating system must be CPM/80 or MSDOS respectively.

2. The machine must support a 16 bit parallel port with the following characteristics:

- 8 bits output only port (address)
- 8 bits bidirectional (data)
- 2 strobe signals

PROCEDURE FOR PORT OF OMNI:

1. Modify the following assembly language routines in OMHDW.MAC:

   (a) Change read/write port addresses in init, pin, apin, pout, apout, pump.
   (b) Adjust timing routines for processor speed - delay, short delay.
       (code is written for 2.5 MHz Z-80, 5 MHz 8088)

2. Reassemble the OMHDW.MAC routine as follows:

   ```
   M80 = OMHDW.MAC
   L80 OMHDW, OMHDW/n/x/e
   ZSID
   *L -103
   *A
   A>SAVE 16 OMHDW.OVR
   ```

   NOTE: Omni software is distributed in one of the following formats:
   - 8" floppy, single sided, single density for CPM
   - 5 1/4" floppy, IBM-PC MSDOS 2.1
   - 5 1/4" floppy, Kaypro II format, CPM

PROCEDURE FOR PORT OF OMNI COM

1. Modify interrupt initialization to support communications vector on new machine.

2. I/O driver must be modified to support the communications controller if a port other than a Z-80 SIO.
Trouble Shooting

PROCEDURES

- Varix Warranty
- Customer Problem Reporting

Problems to be Resolved by:

- Customer Problem Report
- Communication by Telephone
- Written Correspondence

- Customer Response Form

Assistance Policy

Varix will be of assistance to customer problems under warranty or granted through our maintenance contract. Please contact Varix to correct any problems you have regarding hardware, software, or service.

We also encourage our customers to forward any comments that may contribute to or enhance the use of our Varix products.

You may elect to use the Customer Problem Report, the Customer Response Form, or choose to pay us a visit. We take pride in serving our customers.
Trouble Shooting

User reference to: Varix Warranty

For a period of one year after the purchase of the Omni-Programmer, Varix Corporation will provide the following at no charge to the customer:

- Immediate replacement of defective hardware with operational equipment.
- Immediate correction or replacement software to remedy any user identified programming errors attributable to Varix.
- Upgrade of software to meet commitments to the customer made at the time of purchase.

After the warranty period, software upgrades which will both correct existing problems and provide all current software enhancements can be purchased. Defective hardware can be replaced or repaired for a single fixed charge. There will be a 90 day warranty period for each of these services during which any problems found will be fixed free of charge.

Alternately, the customer may purchase a yearly maintenance contract which will provide extension of the hardware warranty for that year and provide regular updates of software, including both problem fixes and enhancements, for that year. Minimum of 3 times/year.

The foregoing warranties are in lieu of all other warranties express or implied including but not limited to warranties of merchantability and fitness for a particular purpose and are the only warranties made by Varix Corporation in connection with the equipment. Related documentation furnished by Varix Corporation and the installation assistance, if any, rendered by Varix Corporation. In no event will Varix Corporation be responsible for consequential damages.
Customer Problem Reporting

Customer reference to the **Customer Problem Report Form** may expedite matters in a problem solving situation.

Also, reference to the Varix **Omni-Programmer User Manual** may assist with your in-house problem solving. In some instances this manual may aid in the definition or description of your problem.

The **Customer Problem Report Form**, included in this section, will afford the customer a current status of his immediate problem and when resolution is achieved will give him a documented source for future reference.

When a problem exists the user should complete this form by:

1) identifying the type of hardware or software in question,

2) give a complete and thorough description of the problem, and

3) list any additional information that may be of assistance to our customer service in recreating the problem at Varix for further study.

The information given above will allow Varix personnel the opportunity to readily assist the customer.

Please see "Contacting Varix" in this section to secure further procedure.
Troubleshooting

Contacting Varix

Communication by Telephone:

a) have your Customer Problem Report ready to help you when you phone,

b) specify to our Varix operator whether your problem is hardware or software related,

c) use your form to write down any response or advice the customer service engineer has given you. Use the "Resolution" area provided on the form, and

d) keep your completed form for return calls or future reference.

Written Correspondence:

a) send all correspondence to our mailing address:
   Varix Corporation
   122 Spanish Village #608
   Dallas, TX  75248
   ATTN: Customer Service

b) enclose with your cover letter a copy of the Customer Problem Report that you have completed.

c) The resolution area of the form will then be completed by Varix and returned to you.

d) Keep your returned and completed form for further correspondence and/or future reference.
Trouble Shooting: Schematics

Omni-Programmer Schematics

The following pages contain the schematics for the Omni-Programmer models SP0300 and GP1140. They are provided as reference for further understanding of the interface specifications.

Nomenclature used in these schematics is:

J1 indicates a part of a connector J1
p.6,9 are page references for the signal connection
System ground

U31 IC #31 Q1 transistor #1
R15 resistor #15 C5 capacitor #5
D7 diode #7 L4 LED #4

O3/01/84 Omni-Programmer Technical Manual Release 1.0
User Diagnostics

BW is a Burn Test Program. It is a functional test of the Omni and will execute while loaded into either Drive A: or B:. It is available in any of the standard formats in which Omni software is distributed and a copy is included with this manual.

BW working with an IBM-PC.

EXECUTABLE program.

Insert DOS Disk - (MSDOS must be running)

To indicate the default drive from A> to B>-
Type B: i.e. A>B: (press Enter)

BW procedure:

A>EXEC:BW (press Enter) or B>EXEC:BW (press Enter)

When the menu appears:

B = Continuous Burn In
W = Extended Wigpin Test
C = Current Limit Test **Nct used fcr functional test**
T = Test 1.5 Volt Circuit
E = Cross-effect Test
X = Exit

Depress B (Enter)

This will give you a visual test both on the computer screen and with the lights on the Omni-Programmer flashing continuously. While the Continuous Burn In file is running if there is a failure then depress any key and return to the menu.

Using the Menu continue to select the W, T, and E files to aid in locating the failure. When the failure is identified make a descriptive note as to the location and type of problem and call the Varix Customer Service Department for further assistance.
User Diagnostics: Burn Test Program BW cont.

BW working with a Kaypro computer.

A>COM:BW (press Return) or B>COM:BW (press Return)

To access Drive B from Drive A:

A>B: (press Return)

The screen will require a y/n response to Continuous Burn In. Y will initiate the Continuous Burn In file and a N response will give you the menu.

While running the Continuous Burn In file should there be a failure then the RESET switch on the back of the Kaypro must be depressed. Bring up BW again and this time around answer N to get the BW menu.

When the menu appears:

W = Extended Wigpin Test
C = Current Limit Test **Not used for functional test**
T = Test 1.5 Volt Circuit
E = Cross-effect Test
X = Exit

Using the menu as a guide select the W, T, and/or E file to aid in locating the failure. When the failure is identified make a descriptive note as to the location and type of problem and call the Varix Customer Service Department for further assistance.
title 'production hardware interface for the kaypro ii'
.z80
cseg

; set up the jump table

jp resall
jp dissall
jp respin
jp dispin
jp setpin
jp alt1pin
jp alt2pin
jp qenable
jp qdispin
jp qsetpin
jp qalt1pin
jp pump
;                
jp qalt2pin
jp rdpin
jp setv1
jp setv2
jp setv3
jp setcv1
jp setcv2
jp setthdr
jp getthdr
jp ledon
jp ledoff
jp connect
jp init
jp delay
jp shortdelay
jp pin
jp apin
jp pout
jp apout

; icostate contains the bit values written to each of the output pins.

icostate:          db  00h, 00h, 00h, 00h, 00h, 00h, 00h
                   db  00h, 00h, 00h, 00h, 00h, 00h, 00h

; pintab maps pin numbers to port addresses and bit positions.
; the first entry is empty so that the indexes will be 1 relative.
; each pin has the following information: primary port, primary port
; bit mask, secondary output port, number of bits in second port, 
; bit mask in secondary port.

pintab: db  00h, 00h, 00h, 00h ; dummy entry
| db | 05h, 10h, 00h, 00h, 00h | ;pin # 1 |
| db | 05h, 20h, 00h, 00h, 00h | ;pin # 2 |
| db | 05h, 40h, 00h, 00h, 00h | ;pin # 3 |
| db | 05h, 80h, 00h, 00h, 00h | ;pin # 4 |
| db | 03h, 01h, 00h, 00h, 00h | ;pin # 5 |
| db | 03h, 02h, 08h, 01h, 01h | ;pin # 6 |
| db | 03h, 04h, 08h, 01h, 02h | ;pin # 7 |
| db | 03h, 08h, 00h, 00h, 00h | ;pin # 8 |
| db | 03h, 10h, 08h, 01h, 04h | ;pin # 9 |
| db | 03h, 20h, 00h, 00h, 00h | ;pin # 10 |
| db | 04h, 20h, 0ch, 02h, 01h | ;pin # 11 |
| db | 02h, 10h, 09h, 01h, 80h | ;pin # 12 |
| db | 01h, 80h, 0dh, 02h, 40h | ;pin # 13 |
| db | 01h, 40h, 09h, 01h, 20h | ;pin # 14 |
| db | 01h, 20h, 09h, 01h, 10h | ;pin # 15 |
| db | 01h, 10h, 09h, 01h, 08h | ;pin # 16 |
| db | 01h, 08h, 09h, 01h, 04h | ;pin # 17 |
| db | 01h, 04h, 09h, 01h, 02h | ;pin # 18 |
| db | 01h, 02h, 0ch, 02h, 10h | ;pin # 19 |
| db | 01h, 01h, 0ch, 02h, 04h | ;pin # 20 |
| db | 00h, 01h, 0ah, 02h, 01h | ;pin # 21 |
| db | 00h, 02h, 0ah, 02h, 04h | ;pin # 22 |
| db | 00h, 04h, 0ah, 02h, 10h | ;pin # 23 |
| db | 03h, 80h, 00h, 00h, 04h | ;pin # 24 |
| db | 00h, 08h, 0ah, 02h, 40h | ;pin # 25 |
| db | 00h, 10h, 0bh, 02h, 01h | ;pin # 26 |
| db | 00h, 20h, 0bh, 02h, 04h | ;pin # 27 |
| db | 00h, 40h, 0bh, 02h, 10h | ;pin # 28 |
| db | 00h, 80h, 0bh, 02h, 40h | ;pin # 29 |
| db | 02h, 80h, 0ch, 02h, 40h | ;pin # 30 |
| db | 02h, 04h, 09h, 01h, 01h | ;pin # 31 |
| db | 04h, 80h, 08h, 01h, 10h | ;pin # 32 |
| db | 02h, 08h, 0dh, 02h, 01h | ;pin # 33 |
| db | 02h, 02h, 09h, 01h, 40h | ;pin # 34 |
| db | 02h, 01h, 08h, 01h, 08h | ;pin # 35 |
| db | 02h, 20h, 0dh, 02h, 04h | ;pin # 36 |
| db | 02h, 40h, 08h, 01h, 20h | ;pin # 37 |
| db | 04h, 40h, 08h, 01h, 40h | ;pin # 38 |
| db | 04h, 10h, 00h, 00h, 00h | ;pin # 39 |
| db | 04h, 08h, 00h, 00h, 00h | ;pin # 40 |
| db | 04h, 04h, 00h, 00h, 00h | ;pin # 41 |
| db | 04h, 02h, 00h, 00h, 00h | ;pin # 42 |
| db | 04h, 01h, 00h, 00h, 00h | ;pin # 43 |
| db | 03h, 40h, 0dh, 02h, 10h | ;pin # 44 |
| db | 05h, 01h, 00h, 00h, 00h | ;pin # 45 |
| db | 05h, 02h, 00h, 00h, 00h | ;pin # 46 |
| db | 05h, 04h, 00h, 00h, 00h | ;pin # 47 |
| db | 05h, 08h, 08h, 01h, 80h | ;pin # 48 |

; define the port select values

v1volt equ 10h ;select v1 voltage
v2volt equ 11h ;select v2 voltage
v3vclt equ 12h ;select v3 voltage
v1curr equ 13h ;select v1 current limit
v2curr equ 14h ;select v2 current limit
cutthr equ 15h ;output threshold
inthr equ 0dh ;input threshold
ledport equ 16h ;leds and current enables

; define return values
;
true equ 1
false equ 0
errcr equ -1

; define the indexes into pintab
;
primary equ 0 ;primary port
prmask equ 1 ;primary port bit mask
secndy equ 2 ;secondary port (0 if none)
scbits equ 3 ;number of bits in secondary port (1 or 2)
sclcc equ 4 ;location of bits in secondary port (0 - 7)

; define the address of the configuration table
;
contab equ 050ch

; define the parallel interface ports
;
dbpar equ 0ah ;port b data on parpic
cbpar equ 0bh ;port b control on parpic
dbsys equ 1eh ;port b data on syspic
cbsys equ 1fh ;port b control on syspic
dapar equ 08h ;port a data on parpic
capar equ 09h ;port a control on parpic
dbpar equ 0ah ;port b data on parpic
cbpar equ 0bh ;port b control on parpic
dasys equ 1ch ;port a data on syspic
casys equ 1dh ;port a control on syspic
dbsys equ 1eh ;port b data on syspic
cbsys equ 1fh ;port b control on syspic
; set up a byte to hold the state of the leds
;
leds: db 03fh
;
; resall - reset all pins
; entry: none
; exit: none

resall: push bc ;save bc
ld b,0
ld hl,icstate ;get the address of icstate
rloop: ld a,b ;get the port number
cp 6 ;check the range
jp z,rend ;stop if equal to 6
ld e,0 ;zero the port
ld (hl),e
push bc ;save in case apcut destroys their values
push hl
call apcut ;output zero to the specified port
pop hl ;restore
pop bc
inc hl
inc b ;increment to next port
jr rloop
rend: pop bc ;restore bc
ret

; disall - disable all pins
; entry: none
; exit: none

disall: push bc ;save bc
ld b,8 ;set up first port number
ld hl,icstate + 8 ;get the address of the 8th port in icstate
dloop1: ld a,b ;get the port number
ld (hl),e ;zero the port
push hl ;get the address
call apcut ;output zero to the specified port
pop hl ;restore
pop bc
inc hl
inc b ;increment to next port
jr dlccp1

dend1: ld b,0 ;set up first port number
    ld hl,iostate ;get the address of iostate

dlccp2: ld a,b ;get the port number
    cp 6 ;check the range
    jr z,dend2 ;stop if equal to 6
    ld e,0ffh ;set every bit in the port
    ld (hl),e
    push bc ;save in case apcut destroys their values
    push hl
    call apcut ;output zero to the specified port
    pop hl ;restore
    pop bc
    inc hl
    inc b ;increment to next port
    jr dlccp2

dend2: pop bc ;restore bc
    ret

; respin - reset a pin to its low value
; entry: parameter #1 - pin number to be reset
; exit: none

respin: call parm1
    push bc ;save bc
    call primary ;get the primary port information
    call disprimary ;disable primary bit
    pop bc ;restore bc
    ret

; dispin - set a pin to its tri-state value
; entry: parameter #1 - pin number to be disabled
; exit: none

dispin: call parm1
    push bc ;save bc
    call primary ;get the primary port information
    push bc ;save for later
inc hl
ld a,(hl) ;secondary port
cr a
jr z,disprm
dec hl

ld d,0 ;disable decode
call setsecond ;set up secondary bits
disprm: pop bc ;restore the primary port information
call enaprimary ;enable primary bit
pop bc ;restore bc
ret

; setpin - set a pin to its high value
; entry: parameter # 1 - pin number to be set
; exit: none
;
setpin:
call parm1
push bc ;save bc
call primary ;get the primary port information
push bc ;save for later
ld d,1 ;set decode
call setsecond ;set up secondary bits
pop bc ;restore the primary port information
call enaprimary ;enable primary bit
pop bc ;restore bc
ret

; alt1pin - set a pin to its alternate high value
; entry: parameter # 1 - pin number to be set
; exit: none
;
alt1pin:
call parm1
push  bc  ;save bc

call  primary  ;get the primary port information
push  bc  ;save for later

ld   d,2  ;alt1 decode
call  setsecond  ;set up secondary bits

pcp  bc  ;restore the primary port information
call  enaprimary  ;enable primary bit

pop  bc  ;restore bc
ret

;  alt2pin - set a pin to its alternate high value
;  entry: parameter # 1 - pin number to be set
;  exit: none
;

alt2pin:
    call  parm1
push  bc  ;save bc

call  primary  ;get the primary port information
push  bc  ;save for later

ld   d,3  ;alt2 decode
call  setsecond  ;set up secondary bits

pcp  bc  ;restore the primary port information
call  enaprimary  ;enable primary bit

pop  bc  ;restore bc
ret

;  qenable - enable the primary port for a pin
;  entry: parameter # 1 - pin number to be enabled
;  exit: none
;

qenable:
    call  parm1

;  push  bc  ;save bc
qdispin - set secondary port for a pin to its tri-state value
entry: parameter #1 - pin number to be disabled
exit: none

qdispin:
;      call parm1
;      push bc          ;save bc
;      call primary
;      inc hl
;      ld a,(hl)        ;secondary port
;      or a
;      jr z,qdisprm
;      dec hl
;      ld d,0           ;disable decode
;      call setsecond  ;set up secondary bits
;qdisprm:
;      pop bc          ;restore bc
;      ret

qsetpin - set the secondary port of a pin to its high value
entry: parameter #1 - pin number to be set
exit: none

qsetpin:
;      call parm1
;      push bc          ;save bc
;      call primary      ;get the primary port information
;      ld d,1            ;set decode
;      call setsecond    ;set up secondary bits
;      pop bc            ;restore bc
; ret

; qalt1pin - set the secondary port of a pin to its alternate high value
; entry: parameter #1 - pin number to be set
; exit: none

qalt1pin:
  call parm1
  push bc ;save bc
  call primary ;get the primary port information
  ld d,2 ;set decode
  call setsecond ;set up secondary bits
  pcp bc ;restore bc
  ret

; qalt2pin - set the secondary port of a pin to its alternate high value
; entry: parameter #1 - pin number to be set
; exit: none

qalt2pin:
  call parm1
  push bc ;save bc
  call primary ;get the primary port information
  ld d,3 ;set decode
  call setsecond ;set up secondary bits
  pc p bc ;restore bc
  ret

; primary - get the primary port information
; entry: register e - pin number
; exit: register b - primary port number
; register c - primary port mask

primary:
  ld a,e ;multiply pin by 5
sla e ;get the port information for the pin
sla e
add a,e
ld e,a
ld hl,pintab ;port table
ld d,0
add hl,de ;address in pintab of information
ld b,(hl) ;primary port
inc hl
ld c,(hl) ;pin mask
ret

; disprimary - disable the primary bit
; entry: register b - primary port number
; register c - primary port mask
; exit: none

disprimary:
ld e,b ;get the current state of all the bits
ld d,0
ld hl,iostate
add hl,de
ld a,c
cpl
and (hl) ;merge the select bits into the byte
ld (hl),a ;save back into iostate
ld e,a ;set up to call pout
ld a,b
call apout
ret

; enapprimary - enable the primary bit
; entry: register b - primary port number
; register c - primary port mask
; exit: none

enapprimary:
ld e,b ;get the current state of all the bits
ld d,0
ld hl,iostate
add hl,de
ld a,c ;get the mask for cring
cr (hl) ;clear the secondary bits
ld (hl),a ;save back into icstate
ld e,a ;set up to call apout
ld a,b
call apout
ret

; setsecond - set up secondary bits
;
; entry: register hl - address of byte preceding secondary port
; register d - type of setup:
; 00 - disable
; 01 - set
; 10 - alt 1
; 11 - alt 2
;
; exit: none
;
setsecond:
inc hl
ld b,(hl) ;secondary port number
inc hl
ld a,(hl) ;number of bits in secondary port
inc hl
ld c,(hl) ;secondary pin mask
cr a ;check for no secondary port
ret z

cp 2
jr nz,cnebit ;one bit decode on secondary port

ld a,c
sla a
cr c
cpl
ld e,a

ld a,d ;get decode type
cr a
jr nz,dec01

ld a,e ;and mask in register a
ld c,0 ;or mask in register c
jr decode

dec01:
cp 1
jr nz,dec10

ld a,e ;and mask in register a
jr decode ;or mask already in register c

dec10:
cp 2
jr nz,dec11
ld a,e ;and mask in register a
sla c ;cr mask in register c
jr decode
dec1: ld a,e
    cpl
    ld c,a ;cr mask in register c
    cpl
    jr decode
conebit: ld a,c
cpl
    ld e,a
dec1: ld a,e
    cpl
    ld c,0 ;cr mask in register c
    jr decode
dec1: ld a,e
    cpl
    ld c,0 ;cr mask in register c
    jr decode
decode: ld e,b ;get the current state of all the bits
    ld d,0
    ld hl,icstate
    add hl,de
    and (hl) ;clear the secondary bits
    cr c ;set the decode pattern
    ld (hl),a
    ld e,a ;set up to call apcut
    ld a,b
    call apcut
    ret

; rdpin - read the value of a pin
; entry: parameter #1 - pin number to be read
; exit: return true if the pin = 1
;       false if the pin = 0
; rdpin: call parm1
    push bc ;save bc
    call primary ;get the primary port information
push bc ;save for later
ld a,b ;set up to call pin
call apin ;get the value for the pin
pcp bc
and c ;check the value of the pin
pcp bc
jr nz,rtrue ;bit was not set
ld hl,false
ret
rtrue: ld hl,true ;bit was set
ret

; setv1 - set the voltage of v1
; entry: parameter # 1 - the number of volts to set v1 to
; exit: none
; notes: the voltage can be set to anything between 0 and 30.60 volts in increments of 0.12 volts
;
setv1: ld a,v1vclt ;v1 output port is in register a
jr setreg ;output the voltage level

; setv2 - set the voltage of v2
; entry: parameter # 1 - the number of volts to set v2 to
; exit: none
; notes: the voltage can be set to anything between 0 and 30.60 volts in increments of 0.12 volts
;
setv2: ld a,v2vclt ;v2 output port is in register a
jr setreg ;output the voltage level

; setv3 - set the voltage of v3
; entry: parameter # 1 - the number of volts to set v3 to
; exit: none
; notes: the voltage can be set to anything between 0 and 30.60 volts in increments of 0.12 volts
;
setv3: ld a,v3vclt ;v3 output port is in register a
jr setreg ;output the voltage level

; setcv1 - set the current limit at v1
; entry: parameter # 1 - the number of milliamps to set v1 to
; exit: none
; notes: the voltage can be set to anything between 0 and 1020
;        milliamps in increments of X milliamps

setcv1: ld a,v1curr ;v1 output port is in register a
jr setreg ;output the current level

; setcv2 - set the current at v2
; entry: parameter # 1 - the number of milliamps to set v2 to
; exit: none
; notes: the voltage can be set to anything between 0 and 1020
;        milliamps in increments of X milliamps

setcv2: ld a,v2curr ;v2 output port is in register a
jr setreg ;output the current level

; setthr - set the threshold voltage
; entry: parameter # 1 - the number of volts to set the threshold to
; exit: none
; notes: the voltage can be set to anything between 0 and 30.60
;        volts in increments of 0.12 volts

setthr: ld a,cutthr ;threshold output port is in register a

; setreg - set a voltage or current register
; entry: register a - register address
;        parameter # 1 - voltage or current level
; exit: none
setreg: call parm1
push bc ;save bc
call apcut ;output the register value
pcp bc ;restore bc
ret

; getthr - set the threshold voltage
; entry: none
; exit: return threshold voltage in hl
; notes: the voltage can be set to anything between 0 and 30.60 volts in increments of 0.12 volts

getthr: push bc ;save bc
ld a,inthrb ;threshold input port is in register a
call apin ;input the voltage level
ld l,a ;put return value in hl
ld h,0
or h ;set the zero flag
pcp bc ;restore bc
ret

; ledon - turn on the leds
; input: the bits set for turning on leds and current limit enables
; output: none

ledon:
call parm1
ld a,e
push bc ;save bc
xcr Offh ;invert to clear bits
ld e,a
ld a,(leds)
and e
ld (leds),a
ld e,a ;set up for pcut
ld a,ledport
call apcut
ledoff: - turn off the LEDs

; input: the bits set for turning off LEDs and current limit enables
; output: none

ledoff:
call parm1
push bc
; save bc
ld a,(leds)
cr
def
ld (leds),a
def
de,a
; set up for pcut
def
def,ledpcrt
call apcut

pop bc
; restore bc
ret

connect:
push bc
; save bc
ld a,inthr
; save the current threshold value
call apin
push af
ld e,5ah
def
ld a,cutthr
; output test value to threshold
call apcut
def
de,0ffh
; output current on v1 to remove 5ah from
def
def,v1curr
; the outputs of the PIC
call apcut
def
def,inthr
; read it back in
call apin
def
def,b,a
pcp af
push bc
; save for later
ld e,a
; restore original threshold
call apcut
pcp bc
; get test value
ld a,b
cp 5ah
jr nz,cerr
ld hl,1
jr exit
cerr:
ld hl,0
exit:
ld a,h
cr
def
pcp bc
init:

; init - hardware initialization
;
entry: none
;
exit: none
;
notes: this subroutine would do any initialization required for
the parallel i/o port and for the delay subroutine
;

algorithm:
; the b port of the z80-pic at addresses 08h - 0bh
; is initialized for output with interrupts disabled.
; this port contains the address for cmniprog.
;
the b port of the system z80-pic at addresses
01ch - 01fh is initialized for output with interrupts
disabled for inputting the data for cmniprog. this-
port is reinitialized for input when reading data
from cmniprog.
;
init: ld a,0fh ; set up mode 0
cut (cbpar),a ; output to port b of parpic
cut (cbsys),a ; output to port b of syspic

ld a,07h ; set up interrupts disabled
cut (cbpar),a ; output to port b of parpic
cut (cbsys),a ; output to port b of syspic

ret

;

delay - 100 micro second delay
;
entry: parameter #1 - number of 100 micro second tics to delay
;
exit: none
;
notes: this subroutine should be as accurate as possible. some
devices have a tolerance of only +cr - 10% and the calling
overhead may account for most of it. this routine can be
implemented with hardware interval counters or software loops.
;
notes: registers a, d, e, h, and l are changed
;

delay: ; cuter lloop - this lloop takes 100
; micro seconds per iteration

call parm1
inc hl
ld d,(hl) ; get the number of tics in de

clccp: push de ; 11 t states
ld hl,5 ; 10 t states

lccp: dec hl ; 6 t states
dec a ; 5 t states
dec a ; 5 t states
dec a ; 5 t states
ld a,l ; 4 t states
cr h ; 4 t states
jp nz,lccp ; 10 t states

pccp de ; 10 t states
dec de ; 6 t states
ld a,e ; 4 t states
cr d ; 4 t states
jp nz,lccp ; 10 t states

ret

; shortdelay - 10 microsecond delay
; entry: none
; exit: none
; notes: this subroutine should take at least 10 microseconds to return. it is a one shot deal.
; registers a, d, e, h, and l are changed

shortdelay:

ld hl,0 ; 10 t states
dec a ; 5 t states
ret ; 10 t states

; pin - get the state of a particular pin
; entry: parameter # 1 - prom port select (0 - 15)
; exit: return the input value from the port

pin: call parm1
ld a,e ; port select in a
push bc ; save bc
call apin ; input it
apin: and 7fh ;make sure direction bit is input
ld b,a ;save in b
ld a,0cfh ;make syspc port b input
cut (cbsys),a
ld a,0ffh
cut (cbsys),a
ld a,b
cut (dbpar),a ;output the address to parpic
in a,(dbsys) ;input the data
ret

apin - get the state of a particular pin (does not interface to C)
entry: register a - prom port select (0 - 15)
exit: return the input value from the port in register hl

pout: call parm1
ld a,e ;port select in a
inc hl
inc hl
ld e,(hl) ;pin manipulation mask in e
push bc ;save bc
call apout ;input it
pcp bc ;restore bc
ret

pout - output data to the hardware
entry: parameter # 1 - prom port select (0 - 22)
parameter # 2 - pin manipulation mask
exit: none

pout - output data to the hardware (does not interface to C)
entry: register a - prom port select (0 - 22)
register d - pin manipulation mask
; exit: none
;
apcut: cr 80h ;set direction bit to output
   ld b,a ;put the results in b, the
   ld a,0fh ;make sure syspic port b is output
   cut (cbsys),a
   ld a,b
   cut (dbpar),a ;cutput port select'
   ld a,e ;put the pin mask in a
   cut (dbsys),a
   ret

; pump - pump a stream of data bytes to the parallel interface
;
; entry: parameter #1 - address of byte stream
;
; exit: none
;
; notes:
;
; The format of the byte stream is as follows:
;
; $length$ number of bytes in byte stream
; $addr$ high order bit must be set in address
; $data$ data to be written to omni register
; $addr$ address and data bytes always in pairs
; $data$
; $data$
; $data$
;
; pump: call parm1
   inc hl
   ld h,(hl) ;get the pointer into hl
   ld l,e
   push bc ;save bc for aztec c
   ld b,(hl) ;get the byte count
   inc hl ;point to first byte
   ld a,0fh ;make sure syspic port b is output
   cut (cbsys),a

; this code outputs an address and data pair to the omni-programmer
;
20
pair: ld  c,dbpar ;address port
cuti
ld  c,dbsys ;data port
cuti
jp   nz,pair ;keep going

; return to aztec c
;

pcp   bc
ret

; parm1 - get the first parameter off of the stack
;
; entry: none
;
; exit: low byte of parameter in register e
;
parm1: ld  hl,4 ;get the parameter off of the stack
add  hl,sp ;without disturbing the stack
ld   e,(hl) ;parameter byte in e
ret

end
* All LM391's have V='+29V.

- Jumper used to make connection
*ALL LM339's HAVE V+=+12V,
- JUMPER USED TO MAKE CONNECTION*
* All LM339's have V^+ = 29V.
* ALL LM339's HAVE \( V^* = +2.9 \) V.
ALL LM339's HAVE $V^+ = +29V$. 
CUSTOMER PROBLEM REPORT

HARDWARE  Please Identify Type or Model Listed Below:
  o  Chip Handler  c  GP1140  c  Socket Adaptor
  o  IBM-PC Card  c  SP0300  c  Cable
  o  Kaypro II

SOFTWARE  Please Identify Source:
  c  License Device Group #____  Version __________
  c  Omni Comm
  c  MMI PALASM
  c  Kaypro (Varix does not warrant nor maintain third
  party software)

Date: ________________  Reported by __________

Description of Problem:
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________

Example Sequence for Recreating Problem:
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________

Resolution:  Varix Contact: __________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
CUSTOMER RESPONSE FORM

Your comments help us improve the quality and usefulness of our publications. Please use this form for questions or comments about this publication. If your answer to a question is "no" or requires additional comments, please explain in the space provided below. Give specific page and line references when appropriate.

If you wish a reply, be sure to include your name and address.

Comments and suggestions become the property of Varix Corporation.

<table>
<thead>
<tr>
<th></th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is the Manual well organized?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is the Manual easy to read and</td>
<td></td>
<td></td>
</tr>
<tr>
<td>understand?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is the Manual complete?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is the Manual written for your</td>
<td></td>
<td></td>
</tr>
<tr>
<td>technical level?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Comments:

____________________________________________________________________________________
____________________________________________________________________________________
____________________________________________________________________________________
____________________________________________________________________________________

Thank you for your cooperation.

Please mail the completed form to:

Varix Corporation
122 Spanish Village #608
Dallas, TX 75248 USA

03/01/84 Omni-Programmer Technical Manual Release 1.0