An EPROM programmer for the ET3400 expansion system

Michael J. Milway
University of Wollongong, mjm@uow.edu.au

Follow this and additional works at: https://ro.uow.edu.au/compsciwp

Recommended Citation
An EPROM programmer for the ET3400 expansion system

Michael Milway

Department of Computing Science
University of Wollongong

Abstract

An EPROM programmer has been constructed that can program 2716 type 2K by 8 bit and 2532 type 4K by 8 bit EPROMs. Expansion to 8K by 8 bits EPROMs is possible. The EPROM programmer is used as a peripheral device to the ET3400 Microcomputer Trainer Expansion system.

Data to be programmed may be down line loaded from a host system or read in from another ROM or EPROM. The EPROM programmer software can program an EPROM, verify that programming was correct, read a ROM or EPROM, check for correct erasure and fill an area of memory with OxFF.
An EPROM programmer for the ET3400 expansion system

Michael Milway
Department of Computing Science
University of Wollongong

ABSTRACT

An EPROM programmer has been constructed that can program 2716 type 2K by 8 bit and 2532 type 4K by 8 bit EPROMs. Expansion to 8K by 8 bits EPROMs is possible. The EPROM programmer is used as a peripheral device to the ET3400 Microcomputer Trainer Expansion system.

Data to be programmed may be down line loaded from a host system or read in from another ROM or EPROM. The EPROM programmer software can program an EPROM, verify that programming was correct, read a ROM or EPROM, check for correct erasure and fill an area of memory with 0xFF.

1. Introduction

In many small microcomputers there is a small amount of ROM that contains a monitor program. The rest of the memory is made up of RAM which does not retain its data after power is turned off. Programs must be reloaded after power is reapplied. This is very inconvenient, especially where frequently used software is involved. The solution to this problem is to write the programs into EPROM which will retain its data after power down.

EPROMs are programmed by setting up the address and data and applying a high voltage programming pulse for a set time. Once the data has been programmed in it can only be erased by exposing the EPROM to high intensity ultraviolet light. To facilitate the programming of EPROMs and EPROM programmer has been constructed that is capable of programming the most common types of EPROMs. It was made as a peripheral device for the existing ET3400 microcomputer trainer system, as this is much more flexible than a stand alone device. The ET3400 system is a small teaching system based on the Heathkit ET3400 microcomputer trainer. An expansion bus and card cage has been added enabling various experiment cards to be plugged in.

2. Design Considerations

2.1. Pin Compatibility

Fortunately, as new EPROM types have been developed they have remained fairly pin compatible with the older types. With the exception of a few pins the pinouts of these devices have remained constant. Even the newer 28 pin devices have been designed so that they are compatible with the 24 pin devices.

Mask programmed ROMs also have the same pinouts as EPROMs. This is done so that systems can be developed using easily programmed EPROMs and then mass produced using the much cheaper (in very large quantities) ROMs without any hardware modifications. Thus if the EPROM programmer can read the data from an EPROM it can also read it from a compatible ROM.

It was decided to concentrate solely on the single supply 24 pin EPROM types rather than the older triple supply ones. Single supply EPROMs require a single 5V supply rather than the 5V and +/- 12V supplies required by the triple supply ones. Table 1 gives the
pinouts of single supply 2K by 8 and 4K by 8 24 pin EPROMs and 8K by 8 and 16K by 8 28 pin EPROMs. Note that the 24 pin devices only differ in two pins and the 28 pin devices fit neatly over the top of the 24 pin devices, with a change in only two more pins.

<table>
<thead>
<tr>
<th>pin #</th>
<th>pin #</th>
<th>2716</th>
<th>2532</th>
<th>2764</th>
<th>27128</th>
</tr>
</thead>
<tbody>
<tr>
<td>24 pins</td>
<td>28 pins</td>
<td>2K * 8</td>
<td>4K * 8</td>
<td>8K * 8</td>
<td>16K * 8</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Vpp</td>
<td>Vpp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>A12</td>
<td>A12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>A7</td>
<td>A7</td>
<td>A7</td>
<td>A7</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>A6</td>
<td>A6</td>
<td>A6</td>
<td>A6</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>A5</td>
<td>A5</td>
<td>A5</td>
<td>A5</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>A4</td>
<td>A4</td>
<td>A4</td>
<td>A4</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>A3</td>
<td>A3</td>
<td>A3</td>
<td>A3</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>A1</td>
<td>A1</td>
<td>A1</td>
<td>A1</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>A0</td>
<td>A0</td>
<td>A0</td>
<td>A0</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>D0</td>
<td>D0</td>
<td>D0</td>
<td>D0</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>D1</td>
<td>D1</td>
<td>D1</td>
<td>D1</td>
</tr>
<tr>
<td>13</td>
<td>11</td>
<td>D2</td>
<td>D2</td>
<td>D2</td>
<td>D2</td>
</tr>
<tr>
<td>14</td>
<td>12</td>
<td>D3</td>
<td>D3</td>
<td>D3</td>
<td>D3</td>
</tr>
<tr>
<td>15</td>
<td>13</td>
<td>D4</td>
<td>D4</td>
<td>D4</td>
<td>D4</td>
</tr>
<tr>
<td>16</td>
<td>14</td>
<td>D5</td>
<td>D5</td>
<td>D5</td>
<td>D5</td>
</tr>
<tr>
<td>17</td>
<td>15</td>
<td>D6</td>
<td>D6</td>
<td>D6</td>
<td>D6</td>
</tr>
<tr>
<td>18</td>
<td>16</td>
<td>D7</td>
<td>D7</td>
<td>D7</td>
<td>D7</td>
</tr>
<tr>
<td>19</td>
<td>17</td>
<td>~E/Prog</td>
<td>A11</td>
<td>~CE</td>
<td>~CE</td>
</tr>
<tr>
<td>20</td>
<td>18</td>
<td>A10</td>
<td>A10</td>
<td>A10</td>
<td>A10</td>
</tr>
<tr>
<td>21*</td>
<td>19</td>
<td>~G</td>
<td>~E</td>
<td>~OE</td>
<td>~OE</td>
</tr>
<tr>
<td>22*</td>
<td>20</td>
<td>Vpp</td>
<td>Vpp</td>
<td>A11</td>
<td>A11</td>
</tr>
<tr>
<td>23*</td>
<td>21</td>
<td>A9</td>
<td>A9</td>
<td>A9</td>
<td>A9</td>
</tr>
<tr>
<td>24*</td>
<td>22</td>
<td>A8</td>
<td>A8</td>
<td>A8</td>
<td>A8</td>
</tr>
<tr>
<td>25</td>
<td>23</td>
<td>Vcc</td>
<td>Vcc</td>
<td>N/C</td>
<td>A13</td>
</tr>
<tr>
<td>26</td>
<td>24</td>
<td>Pgm</td>
<td>Pgm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>25</td>
<td>Vcc</td>
<td>Vcc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Signals marked ' ~ ' are active low.
Pins marked * have different signals on different EPROM types.

Table 1 EPROM pinouts

2.2. Software Compatibility

All the EPROMs mentioned in table 1 are programmed in the same manner. The programming voltage Vpp is applied and the correct address and data asserted. The control pins ~CE, ~OE, ~G and ~E/prog are set to the correct level, depending on the EPROM type, and left for 50 milliseconds. After this time the data will have been programmed into the specified location. By having a set of variable control bytes that are initialised when the EPROM type is determined the same routine may be used to program any of the above mentioned EPROM types.
2.3. Other Design Considerations

No chip should be inserted or removed from a socket while power or logic levels are present. It was decided to include a dual pole switch to control both the supply and programming voltages. The address, data and control buffers are also enabled by this switch.

The programming supply voltage, Vpp, of 25 volts is not available on the ET3400 system. This voltage could either be supplied by an external power supply or by an on board DC to DC converter. The latter option, although more complex, was chosen to make the operation of the EPROM programmer easier.

3. Implementation

Figure 1 shows the circuit diagram for the EPROM programmer. It consists of:

(i) An interface to the ET3400 expansion bus.
(ii) A set of buffers for the address, data and control lines.
(iii) The power supply for Vpp.
(iv) Electronic switching to allow Vpp to be applied under program control.
(v) A rotary switch to determine EPROM type.
(vi) A Zero Insertion Force socket to hold the EPROM.

The pinouts of the signals on the expansion bus are shown in brackets beside the signal.

A 74LS42 is used to decode a board select and three address lines to give a single select line for the EPROM programmer. Address line A2 is used as one of the select lines on PIA2 and inverted for use as a select line on PIA1.

The outputs of the PIAs are buffered by the tristate buffers. These buffers are enabled when the EPROM power switch is on. The buffer on the data lines is also bidirectional.

The power supply for Vpp consists of an oscillator (NE555), a pair of current pumps (C2, D1, D2 and C4, D3, D4) and a voltage regulator (Q1). The output of the oscillator has a swing of 12V which is applied to C2 and C4. The other side of C2 follows this voltage, but is constrained by D1 to always be greater than 11.4V. Thus this side swings between 11.4V and 23.4V. D2 will conduct whenever it is forward biased by more than 0.6V, charging C3 to 22.8V. The second current pump adds this voltage to the input to C4, charging C5 to 33.6V. This voltage will drop under load. Q1 will regulate the on C5 down to 25V for use as Vpp. R2 allows for the adjustment of Vpp between 12V and 33.6V.

The programming voltage may be switched to the EPROM under program control. When PB0 of PIA2 is low and the power switch is turned on, Q2 is turned on allowing Vpp into the EPROM. When Q2 is off the Vpp pin of the EPROM is at 5V.

The three position rotary switch is a simple means of changing the address and control lines to suit different EPROM types. The switch has been wired for 2716 and 2532 type EPROMs. The third position is currently unused. The switch is also connected to three inputs on PIA2 to allow the software to determine the EPROM type.

The EPROM power switch switches Vcc and Vpp to the EPROM. It is also used to enable the tristate buffers. Vcc is buffered and fed into PB4 of PIA2 to tell the software if the power is switched on.

Finally three LEDs are connected to PB5 - 7 of PIA2. These LEDs are used to tell whether programming, verifying or reading operations are in progress.
3.1. Addressing

When a PIA is normally wired up its internal registers are addressed as follows:

<table>
<thead>
<tr>
<th>Address</th>
<th>Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Data Register A</td>
</tr>
<tr>
<td>1</td>
<td>Control Register A</td>
</tr>
<tr>
<td>2</td>
<td>Data Register B</td>
</tr>
<tr>
<td>3</td>
<td>Control Register B</td>
</tr>
</tbody>
</table>

This is fine for eight bit data transfers. The addressing of the PIA registers may be modified to enable sixteen bit transfers (such as outputting EPROM addresses) by swapping A0 and A1 on the register select pins. The address map for the PIA then becomes:

<table>
<thead>
<tr>
<th>Address</th>
<th>Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Data Register A</td>
</tr>
<tr>
<td>1</td>
<td>Data Register B</td>
</tr>
<tr>
<td>2</td>
<td>Control Register A</td>
</tr>
<tr>
<td>3</td>
<td>Control Register B</td>
</tr>
</tbody>
</table>

Sixteen bit loads and stores, such as ldx and stx, may now be made. Data Register A will be the high order register and data Register B the low order one.

A PIA may have all the bits in a data register independently configured as input or output. Reading from the input pins and writing to the output pins may be done without affecting the other pins even though they all share the same address.

PIA1 is used to output the EPROM address. It has been connected to allow sixteen bit transfers as described above. Data Register A is used to output the high order address bits and Data Register B outputs the low order bits. Only thirteen bits out of the possible sixteen bits are used to output the address. The three high order bits of Data Register A are used as inputs to signify the EPROM type.

Data Register A of PIA2 is used to transfer data to the EPROM when programming and to transfer data from the EPROM when reading, verifying or checking erasure.

Data Register B of PIA2 is used to control the function of the EPROM programmer. Bit 0 controls the programming voltage Vpp. Bits 1 and 2 control the Enable and Gate inputs to the EPROM respectively. Bit 3 controls the direction of transfers through the data buffer. Bit 4 is used to sense whether the EPROM power switch is on. Bits 5, 6 and 7 control the Read, Write and Verify LEDs respectively. Figure 2 gives an address map of the PIAs.

4. Software

A program has been written in 6800 assembly language to program 2K or 4K EPROMs. The program currently resides in

```
/usr/hardware/eprom/4keprom.s
```

on system B. A copy of this program is included as an appendix. The following commands are available:

```
Program  p<ramstart>[,<count>][,<romstart>]
Verify    v<ramstart>[,<count>][,<romstart>]
Read     r<ramstart>[,<count>][,<romstart>]
Erased?   e[<romstart>][,<count>]
Fill      f<ramstart>[,<count>]
Exit     x[<address>]
```

Arguments in angle brackets, < >, are hexadecimal values and arguments in square brackets, [], are optional. The default values for optional arguments are 0x0800 or 0x1000 for the count when using 2K or 4K EPROMs respectively, 0x0000 for the EPROM start address and the reset vector (0xf800 for the Heathkit trainers) for the exit address.
The program has been written in structured code. The top level is a command interpreter which takes an input line, extracts the command and argument values. The commands are called as separate subroutines. The command subroutines in turn call other subroutines such as 'ready' which tests for EPROM type and power on and 'putline' which writes a line of text to the screen.

4.1. Command Interpreter

On cold start or errors the command interpreter prints out a welcoming message which includes a list of available commands and required arguments. It then reads a line of text from the keyboard into a buffer. The command character is stripped off and saved. The arguments are then extracted and stored as sixteen bit numbers along with an argument count. The arguments must be valid hexadecimal numbers, separated by commas and terminated by a newline character.

The next step is to load the default values for 2K EPROMs. If 4K EPROMs are being used the count will later be overwritten by the default value for 4K EPROMs. If the optional arguments have been supplied they will overwrite the default values and thus be used instead.

A jump table is used to identify the command and call the appropriate subroutine. When the subroutine returns the program branches back to a warm start and repeats the command interpreter loop. If the command is not found the program prints an error message and branches back to a cold start.

4.2. Program

This command programs either a 2K or 4K EPROM. The user may specify the address in RAM of the data to be programmed, a count of how many bytes to program and where to put the data in the EPROM. The count and the EPROM start address are optional.

The 'ready' subroutine is called first. It checks that the EPROM power switch is on and determines the type of EPROM from the position of the rotary switch. If a 4K EPROM is to be programmed it changes the default count value. It also initializes two variables, 'pinit' and 'pprog', with information necessary to program the required type.

The 'ports' subroutine is then called. It programs the PIAs for correct data direction and sets some initial values. As this subroutine returns with the data port configured for reading the EPROM, the direction of this port is then reversed so that data may be written to the EPROM.

The arguments required to determine the start of the RAM buffer, the count and the EPROM start address are read in, overwriting the default values. If these arguments are not supplied then the default value will be used.

The actual programming of the EPROM now follows. The initial control value is written out to the control port. The current EPROM address is written out to the Heathkit display to show the progress of the programmer. The Current EPROM address and data are then written out to the EPROM. The control value is changed to enable programming. After a fifty millisecond delay the control value is changed back to the initial value and the location has been programmed. The RAM and EPROM addresses are incremented and the count decremented. The programming loop is repeated until all bytes have been programmed.

The control value is reset to 0xFF and a farewell message is printed out before returning to the main program.

Note that all the command subroutines follow the same general pattern of calling 'ready' and 'ports', and loading the address and count arguments before performing the required task. The subroutines then reset the control value, print a farewell message and return to the main program.
4.3. Verify

This command verifies that the data has been correctly programmed into the EPROM. 'Ports' and 'ready' are called and the addresses and count loaded if supplied. The control port is set to enable reading of the EPROM. The control value is the same for 2K or 4K EPROMs. The data is read one byte at a time and compared with the corresponding byte in RAM. If an error is found, the EPROM address, data and the correct data are printed in an error message. The subroutine then waits for any key to be pressed before continuing. If an 'x' has been pressed the command is aborted. 'Verify' exits in the same manner as 'program' when it successfully completes or is aborted.

4.4. Erased

This command checks that all the required locations in the EPROM have been erased, i.e. set to 0xFF. It works in the same manner as verify but checks each location for 0xFF rather than comparing it with a location in RAM.

4.5. Read

This command reads the contents of an EPROM or ROM into a specified area of RAM. 'Ports' and 'ready' are called and the appropriate arguments loaded in. The control value is set to enable the EPROM or ROM to be read. The control value is the same for both 2K and 4K devices. Each location is read in turn and dumped into the specified area of RAM. The control register is then set to 0xFF, a farewell message printed and the subroutine returns to the main program.

4.6. Fill

This command fills a specified area of RAM with 0xFF. This is useful when programming an EPROM with several data segments. The RAM buffer is filled with 0xFF and the data segments loaded into the appropriate positions in the buffer. The whole EPROM is then programmed. Unused areas of the EPROM will remain set to the erased value of 0xFF and thus may be programmed at a later date.

Since this command does not use the EPROM programmer hardware 'ports' and 'ready' are not called. The required RAM address and count, if supplied, are loaded and a loop executed filling the required area with 0xFF. The default count is 0x800 so that if 4K EPROMs are to be used the command must either specify the count or the command used twice. The command exits by printing a farewell message and returning to the main program.

4.7. Exit

This command allows the user to exit the EPROM programmer. If the exit address is supplied the command prints a message and jumps to that address. The default exit address is the reset vector for the microcomputer being used. It is read out of locations 0xfffe and 0xffff, which is the reset vector for both the 6800 and 6809 microprocessor.

4.8. Other Routines

'Ports' is used to set up both PIA's prior to a command being executed. The two address ports are set up as outputs, with the top three bits of the high order address port being set up as inputs. The data port is set up as input. The initial value of the control port is set to 0xFF before it is set to be an output port. This ensures that there is no glitch as the port changes from being an input (default setting on reset) to an output port.

'Ready' checks that the EPROM power switch is on and determines what type of EPROM, 2K or 4K, is being used. It starts by assuming that a 2K EPROM has been selected and sets 'pinit' and 'pprog' to suit. It then sits in a loop waiting until the power switch is on and either a 2K or 4K EPROM has been selected. The third setting for an 8K EPROM is ignored by this program. If a 4K EPROM has been selected 'pinit' and 'pprog' are reinitialised and the default count set to 0x1000.

The terminal I/O subroutines used by this program are already available in the EPROM on the ET3400 expansion system master card (called 6800lib) and the subroutines needed to use the ET3400 display are in the monitor ROM.
5. Operation

The following items of hardware are needed to use the EPROM programmer.

ET3400 trainer
Expansion system with separate +5V, +12V and -12V power supply
Master card with at least 4K of RAM (6K for 4K EPROMs)
EPROM programmer card

Downline load the EPROM programmer software from
/usr/hardware/eprom/4keprom.out

on system B. If the data is to be downline loaded then do so at this stage. Use the B(ias) option on dll to load the data in available RAM.

Start the program running at 0x2000. Turn the EPROM power switch off. Select either 2K (2716) or 4K (2532) type using the rotary switch. Insert the EPROM into the Zero Insertion Force Socket with pin 1 to the upper left. Turn the EPROM power switch on. Enter the required command on the terminal keyboard and press return. When an EPROM is to be programmed it should first be checked for erasure. When it has been programmed it should be verified. The erasure command can also be used to single step through the EPROM data to view the contents. It will skip over locations containing 0xFF.

Sample commands:

p2800,100,400 Program 100 hex bytes starting from location 2800 hex in RAM into the EPROM starting at location 400 hex.
p2800 Program all of the EPROM with data starting at location 2800 hex in RAM. The default count and EPROM start address are used.
p2800,400 Program the first 400 hex bytes of the EPROM with data starting at location 2800 hex in RAM.
v2800 Verify the whole EPROM against data starting at location 2800 hex in RAM.
v2800,100,400 Verify that the first programming example worked correctly.
r2800 Read the whole of the EPROM into RAM starting at location 2800 hex.
e Check the whole EPROM for erasure.
f2800 Fill 2k (default count) of RAM with 0xFF starting at location 2800 hex.
f2800,1000 Fill 4K or RAM with 0xFF starting at location 2800 hex.
x Exit to the reset vector (i.e. the Heathkit monitor).
xa000 Exit to ET3400 transparent link.

6. Future Enhancements

The EPROM programmer in its current state can only handle 24 pin devices. If the 24 pin ZIF socket was replaced by a 28 pin ZIF socket and the third position of the rotary switch used, its capacity would be increased to include the 2764 8K by 8 bit EPROM type. 24 pin devices could still be used by inserting them in the 28 pin socket so that their pins corresponded to the equivalent 28 pin device types, see table 1. The third position of the rotary switch would be used to connect the enable signal to pin 20, output enable to pin 22 and A11 to pin 23. The program voltage, Vpp, would be connected to pin 1, A12 to pin 2, -Vppon to pin 27 and the supply voltage, Vcc, to pin 28. Since pin 26 is no connect, Vcc for 24 pin devices may remain connected.

The only software changes required are to check for the 8K position of the rotary switch and reprogram pinit, pprog and the default count to suit the 2764.

It is impractical to extend the design to handle 27128 or 27256 type devices as these require additional address lines and more switching around of control signals. It would be better to design a new EPROM programmer to cater for these types.
Figure 1a

ET-3400
EPROM PROGRAMMER
SHEET 1 OF 2
DRAWN M.J.M.
18/4/1984
Figure 1b
Figure 2, Addresses of PIAs, bit usage and data directions.

<table>
<thead>
<tr>
<th>address port</th>
<th>bit</th>
<th>direction</th>
<th>use</th>
<th>DDR</th>
<th>initial data</th>
</tr>
</thead>
<tbody>
<tr>
<td>9380 data reg</td>
<td>7</td>
<td>&lt;--</td>
<td>8K</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>(1A) high order address</td>
<td>6</td>
<td>&lt;--</td>
<td>4K</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>&lt;--</td>
<td>2K</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>--&gt;</td>
<td>A12</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>--&gt;</td>
<td>A11</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>--&gt;</td>
<td>A10</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>--&gt;</td>
<td>A9</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>--&gt;</td>
<td>A8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9381 data reg</td>
<td>7</td>
<td>--&gt;</td>
<td>A7</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(2A) low order address</td>
<td>6</td>
<td>--&gt;</td>
<td>A6</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>--&gt;</td>
<td>A5</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>--&gt;</td>
<td>A4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>--&gt;</td>
<td>A3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>--&gt;</td>
<td>A2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>--&gt;</td>
<td>A1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>--&gt;</td>
<td>A0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9382 control reg</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1A)</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9383 control reg</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2A)</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9384 data reg</td>
<td>7</td>
<td>&lt;--&gt;</td>
<td>D7</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(1B) EPROM data</td>
<td>6</td>
<td>&lt;--&gt;</td>
<td>D6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>&lt;--&gt;</td>
<td>D5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>&lt;--&gt;</td>
<td>D4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>&lt;--&gt;</td>
<td>D3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>&lt;--&gt;</td>
<td>D2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>&lt;--&gt;</td>
<td>D1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>&lt;--&gt;</td>
<td>D0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Figure 2 continued.

<table>
<thead>
<tr>
<th>Control Reg (1B)</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR = FFH for data write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DDR = OOH for data read</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OOH to load DDR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>04H to transfer data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Reg (2B)</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>--&gt;</td>
<td>~verify</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--&gt;</td>
<td>~write</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--&gt;</td>
<td>~read</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;--</td>
<td>~En</td>
<td>0</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--&gt;</td>
<td>~drn</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--&gt;</td>
<td>~G</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--&gt;</td>
<td>~E</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--&gt;</td>
<td>~Vpp</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Control Reg (2B)</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>OOH to load DDR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>04H to transfer commands/status</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: Initial data for the command/status register should be loaded before the I/O bits are set to output.

--> means into EPROM
<-- means out of EPROM
<--> means bidirectional data flow
Appendix 1, Software source listing
Terminal driven Eprom programmer for use with Heathkit trainers.

Michael Milway
15/8/1983
amended 26/4/1984

Commands are:

Program: p<ramstart>[,<count>[,<romstart>]]
Verify: v<ramstart>[,<count>[,<romstart>]]
Read: r<ramstart>[,<count>[,<romstart>]]
Erased?: e<romstart>[,<count>]
Fill: f<ramstart>[,<count>]
exit: x[<address>]

; equates

dromaddr equ 0x0000 ;default start address for rom
d2count equ 0x0800 ;default count
d4count equ 0x1000 ;default count for 4K
addrh equ 0x9380
addrld equ 0x9381 ; Eprom address registers
addrhc equ 0x9382
addr1c equ 0x9383 ; eprom address control registers
datad equ 0x9384 ; Eprom data register
datac equ 0x9385 ; Eprom data control register
cntrld equ 0x9386 ; Programmer control data register
cntrlc equ 0x9387 ; Programmer control control register
reset equ 0xfffe ; address of reset vector
redis equ 0xfccc ; reset display routine
outbyt equ 0xfe20 ; display byte routine
putline equ 0xa009
getline equ 0xa00c
wr4 equ 0xa00f
wr2 equ 0xa012
putchar equ 0xa015
getchar equ 0xa018
getaddr equ 0xa02d
tohex equ 0xa030
toascii equ 0xa033
tolower equ 0xa036
ishex equ 0xa039
ls16 equ 0xa03c
wait equ 0xa03f

; command

get command line
collect addresses and check for some command errors
org 0x2000

command
ldx #blurb
jsr putline ; print out welcoming message
cmd.warm
ldx #cmd.prmt
jsr putline ; the prompt
ldx #buffer
jsr getline ; get command line
ldx #buffer
clr noargs ; counter for number of arguments
lda a,0(x) ; get command character
inx
sta a,cmd
jsr getaddr ; get first argument
tst a ; valid address?
bnz cmd.1 ; yes
lda a,0(x) ; no, get next character
jmp cmd.99

cmd.1
stx templ
ldx addr
stx addr1 ; save first address
ldx templ
inc noargs ; number of arguments ++
lda a,0(x) ; next char
inx
cmp a,\'\'', ; legal separator
beq cmd.2
jmp cmd.99

jsr getaddr ; yes, next address
tst a ; valid address?
bnz cmd.3 ; yes
jmp cmd.err

cmd.2
stx templ
ldx addr
stx addr2 ; second address
ldx templ
inc noargs
lda a,0(x)
inx
cmp a,\'\'', ; legal separator?
beq cmd.4 ; yes
jmp cmd.99

jsr getaddr ; valid address
tst a
bnz cmd.5 ; yes
jmp cmd.err

cmd.3
stx templ
ldx addr
stx addr3
ldx templ
inc noargs
lda a,0(x)
inx

cmd.4
stx templ
ldx addr
stx addr4
ldx templ
inc noargs
lda a,0(x)
inx

cmd.5
stx templ
ldx addr
stx addr5
ldx templ
inc noargs
lda a,0(x)
inx

cmd.99
cmp a,\'\n\' ; end of line?
beq interp ; yes, have valid command so interpret it

cmd.err
ldx #errmsg
jsr putline ; output error message
jmp command ; start again

errmsg
dc "arguments error\n\n"
cmd.prmt
dc "eprom.2-> \n"
; Now have a command and arguments, so interpret it
interp

EPROM programmer, version 2
Program 2K or 4K EPROMs
Arguments in angle brackets are hex numbers
Arguments in square brackets are optional
Commands are:
Program p<ramstart>[,<count>][,<romstart>]
Verify v<ramstart>[,<count>][,<romstart>]
Read r<ramstart>[,<count>][,<romstart>]
Erased? e<ramstart>[,<count>]
Fill FF<count>
ex<addr>

ldx reset ;default exit
stx xaddr
ldx #d2count
stx count
ldx #dromaddr
stx romaddr ;load default values for 2K
lda a,cmmmd ;get command
jsr tolower ;make lower case
cmp a,'#p' ;program?
bne intr.1 ;no
jsr program
bra intr.9
intr.1
cmp a,'#v' ;verify?
bne intr.2
jsr verify
bra intr.9
intr.2
cmp a,'#r' ;read?
bne intr.3
jsr read
bra intr.9
intr.3
cmp a,'#e' ;erased?
bne intr.4
jsr erased
bra intr.9
intr.4
cmp a,'#f' ;fill?
bne intr.5
jsr fill
bra intr.9
intr.5
cmp a,'#x' ;exit?
bne intr.6
jmp exit ;exit does not return
intr.6
ldx #cmd.ill ;must be illegal command
jsr putline ;illegal command
jmp command ;cold restart
intr.9
jmp cmd.warm ;next command

cmd.ill
da "illegal command\n\0"
dc "\nEPROM programmer, version 2\n"
dc "Program 2K or 4K EPROMs\n"
dc "Arguments in angle brackets"
dc " are hex numbers\n"
dc "Arguments in square"
dc " brackets are optional\n"
dc "Commands are:\n"
dc "Program p<ramstart>[,<count>][,<romstart>]\n"
dc "Verify v<ramstart>[,<count>][,<romstart>]\n"
dc "Read r<ramstart>[,<count>][,<romstart>]\n"
dc "Erased? e<ramstart>[
"
dc ,<count>]\n"
dc "Fill FF<count> ex<addr]\n"
dc "exit x<addr>]\n"
dc "type \"x\" to abort verify \n"
dc "and erased\n"
May 7 11:30 1984 4keprom.s Page 4

dc "type 'return' to continue"
dc "verify and erased\n"   
dc "Default values are 0800H for "   
dc "count, 0000 for romstart\n"   
dc "and processor reset vector"
dc "  for x\n0"

; Main Eprom subroutines here

; title "Program"
;program an eprom
;inputs: noargs number of arguments
;addrl..addr3 initial pointers
;outputs: none
;calls:  ports, ready, putline
;destroys: A,B,CC,X,memory pointers

program

jsr           ports     ;set up ports, data = input.
ld a,#00
sta a,datac
lda a,#0xff
sta a,datad ;data = output.
lda a,#0x04
sta a,datad ;point to data reg
;check that Eprom is selected and power is on
jsr           ready
lda a,noargs   ;how many arguments
bne psome
ldx #toofew
jsr           putline
rts

psome

ldx   addr1  ;set up Eprom address
stx   ramaddr
dec   a
beq   pbeg
ldx   addr2
stx   count
dec   a
beq   pbeg
ldx   addr3 ;point to next location
stx   romaddr

pbeg

lda a,pinit   ;set up for programming eprom
sta a,ctrl

prpt

jsr           redis
lda a,romaddr
jsr           outbyt
lda a,romaddr+1
jsr           outbyt
ldx   romaddr ;get eprom pointer
stx   addrhd  ;address eprom
inx
stx   romaddr ;next location
ldx   ramaddr
lda a,0x00(x) ;get data
sta a,datad ;output to EPROM
inx
stx ramaddr ;next memory location
lda a,pprog ;program byte
sta a,cntrld
ldx #6250 ;count for 50msec
jsr wait
lda a,pinit
sta a,cntrld
ldx count ;one less to go
dex
stx count
bne prptl ;loop until all positions done
lda a,#0xff
sta a,cntrld
ldx #pdone
jsr putline ;say that its done
rts
pdone
dc "Program completed\n\0"
title "Verify" ;verify eprom contents ;inputs addr1..addr3,noargs ;outputs none ;calls: putline,wr4,ready, ports, wr2, putchar ;destroys: all
verify
jsr ports ;set up ports, data = input
;check selected and power on
jsr ready
lda a,noargs
bne vsome
ldx #toofew
jsr putline
rts
vsome
ldx addr1
stx ramaddr
dec a
beq vbegin
ldx addr2
stx count
dec a
beq vbegin
ldx addr3
stx romaddr ;load required values
vbeginn
lda a,#0x79 ;verify led on
;input data
;output enable on
;chip select on
;Vpp off
vbegin
sta a,cntrld
ldx romaddr
stx addrhd ;address eprom
lda a,datad ;get eprom data
ldx ramaddr
cmp a,00(x); compare eprom contents with
beq vgood ; no error
ldx #vbad1
jsr putline
ldx romaddr
jsr wr4
ldx #vbad2
jsr putline
lda a,datad
jsr wr2
ldx #vbad3
jsr putline
ldx ramaddr
lda a,0(x); get proper data
jsr wr2
ldx a,"\n"; print it
jsr getchar; wait for any character to resume
cmp a,"\n"; if x then exit
beq vend
vgood

ldx ramaddr
inx
stx ramaddr
ldx romaddr
inx
stx romaddr; next location
ldx count
dex
stx count
bne vrppl; leave loop if all locations inspected

vend

ldx #vdone
jsr putline; print all well message
lda a,0xff
sta a,cntlrd; reset programmer control byte
rts

vdone dc "Verify completed\n\0"
vbad1 dc "Location \0"
vbad2 dc " contains \0"
vbad3 dc " instead of \0"
title "Check 2K EPROM erased"
; read a 2K EPROM from the EPROM programmer
; check that each location = FFH
; inputs: addr1, addr2, noargs
; outputs: none
; calls: ports, ready, putline, wr4, wr2, putchar, getc
; destroys: all

erased

jsr ports; set up ports
; check that Eprom is selected and power is on
jsr ready
lda b,noargs; how many arguments?
beq ebn; none
ldx addr1; start address
stx romaddr
May 7 11:30 1984 4keprom.s Page 7

```
dec   b
beq   ebgn
ldx   addr2
stx   count       ;how many

ebgn  lda   a,#0x79 ;verify led on
       ;input data
       ;output enable on
       ;chip select on
       ;Vpp off

       sta   a,cntrld
       ldx   romaddr
       stx   addrhd    ;address eprom
       lda   a,datad  ;get data
       cmp   a,#0xff   ;should be FF if erased
       beq   egood
       ldx   #ebad1
       jsr   putline
       ldx   romaddr
       jsr   wr4
       ldx   #ebad2
       jsr   putline
       lda   a,datad
       jsr   wr2       ;write out error message
       lda   a,"\n"
       jsr   putchar
       jsr   getchar    ;x to exit
       cmp   a,"x"     ;
       beq   ebye
       ldx   romaddr
       inx
       stx   romaddr    ;next please
       ldx   count
       dex
       stx   count
       bne   ewh2        ;more?
       ldx   #ebayel
       jsr   putline
       lda   a,#0xff
       sta   a,cntrld    ;turn off read led

egood  ldx   romaddr
       inx
       stx   romaddr
       ldx   count
       dex
       stx   count
       bne   ewh2

ebye   ldx   #ebayel
       jsr   putline
       lda   a,#0xff
       sta   a,cntrld

ebad1  dc "Location \0"
edb  dc " contains \0"
ebyel dc "Erasure check completed\n\0"
title "Read 2K Eprom"
;read a 2K EPROM from the EPROM programmer
;inputs: addr1..addr2, noargs
;outputs: none
;calls: ready, ports, putline
;destroys: all

read  jsr   ports ;set up ports
       ;check that Eprom is selected and power is on
       jsr   ready
       lda   a,noargs ;how many arguments?
       bne   rsome    ;at least one
       ldx   #toofew
```
jsr putline
rts

rsome
ldx addr1
stx ramaddr
dec a
beq rbeg
ldx addr2
stx count
dec a
beq rbeg
ldx addr3
stx romaddr

rbeg

lda a,#0xd9 ;read led on
;input data
;output enable on
;chip select on
;Vpp off

sta a,cntrl

rwh1

ldx romaddr ;get address pointer
stx addrd ;output to Eprom
inx
stx romaddr ;next
lda a,datad ;get Eprom data
ldx romaddr
sta a,0(x) ;dump to ram
inx ;next address
stx romaddr ;next dump address
ldx count
dex
stx count
bne rwh1 ;any more?
lda a,#0xff
sta a,cntrld ;clear control register
ldx #rdone
jsr putline ;say read done

rts

rdone
dc "Read completed\n\n0"
title "Fill with FF"
;Fill ram with 0xFF
;inputs: addr1, addr2, noargs
;outputs: none
;calls: putline
;destroys: A, X, CC

fill

lda a,noargs ;how many arguments?
bne fsome
ldx #toofew
jsr putline

trs

fsome

ldx addr1
stx ramaddr
dec a
beq fbegin
May 7 11:30 1984 4keprom.s Page 9

.ldx addr2
.stx count

fbegin
.lxa a,#0xff

fwhl
.ldx ramaddr
.sta a,0(x)
.inx
.stx ramaddr
.ldx count
.dex
.stx count
.bne fwhl
.ldx #fdone ;say fill done
.jsr putline

fdone
.dc "Fill completed\n\0"

.exit
;exit to monitor or user program
;inputs: addr - exit address
;outputs: none
;calls: putline, wr4, putchar, wait
;destroys: all
.lda a, noargs
.beq xnone ;use default exit
.ldx addr1
.stx xaddr ;use user exit address

xnone
.ldx #exitmsg
.jsr putline
.ldx xaddr
.jsr wr4 ;write out message
.lxa a,"\n"
.jsr putchar
.ldx #6520 ;count for 50 msec
.jsr wait
.ldx xaddr
jmp 0(x) ;finally exit

exitmsg
.dc "branching to address \0"

.title "misc programmer routines"

ports
;set up ports
;inputs: none
;outputs: none
;calls: nothing
;destroys: A, X, CC
.lda a,#0x04
.sta a,cntrlc ;point to data part of programmer
;conrol register

.lda a,#0xff
.sta a,cntrld ;initial control values all 1's
.lda a,#00
.sta a,addr1c
.sta a,addrhc
.sta a,datac
.sta a,cntrlc ;point to ddr of all ports
.ldx #0x1fff ;ddr for address port
.stx addrhd
.sta a,datad ;data port all input
lda a,#0xef ;bit in, 7 bits out
sta a,cntrld
lda a,#0x04
sta a,addrhc
sta a,addr1c
sta a,datac
sta a,cntr1c ;point to data register of all ports
rts
;Ready
;loop until both 2K or 4K selected and power on
;inputs: none
;outputs: none
;calls: nothing
;destroys: A, CC

ready
lda a,#0xa5 ;assume have 2K eprom
sta a,pinit ;set initial value to be used by
;program routine
sta a,pprog ;write led on, output data, G=1, E=0,
;Vpp off
lda a,#0xa6 ;value to be used for programming
sta a,pinit ;E -- 1, Vpp -- on

ready1
lda a,cntrld ;check power
and a,#0x10
bne ready1
lda a,addrhd ;get 2k, 4k, 8k switch settings
and a,#0x20 ;2K?
beq ready2 ;yes
lda a,addrhd
and a,#0x40 ;4K?
bne ready1 ;neither so wait

;4K so replace pinit and pprog with 4K values
lda a,#0xa3 ;init value for 4K eprom
sta a,pinit ;write led on, output data, G=x', E=1
;Vpp off

lda a,#0xa0 ;program value
sta a,pprog
lda #4count ;default count for 4K
stx count
ldx #r4kmsg
jsr putline
rts

r4kmsg
dc "4k eprom selected\n"
toofew
dc "not enough arguments\n"
seg 1

; variable storage area
addr equ 0x102 ;return value from getaddr
templ1 equ 0x108
addr1 equ 0x10a
addr2 equ 0x10c
addr3 equ 0x10e
<table>
<thead>
<tr>
<th>Address Port</th>
<th>Bit (0-7)</th>
<th>Direction</th>
<th>Use</th>
<th>DDR</th>
<th>Initial Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>9380 data reg</td>
<td>7</td>
<td>&lt;--</td>
<td>8K</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>(A1)</td>
<td>6</td>
<td>&lt;--</td>
<td>4K</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>high order address</td>
<td>5</td>
<td>&lt;--</td>
<td>2K</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>9381 data reg</td>
<td>7</td>
<td>--&gt;</td>
<td>A12</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(B1)</td>
<td>6</td>
<td>--&gt;</td>
<td>A6</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>low order address</td>
<td>5</td>
<td>--&gt;</td>
<td>A5</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9382 control reg</td>
<td>7</td>
<td>--&gt;</td>
<td>A4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(A1)</td>
<td>6</td>
<td>--&gt;</td>
<td>A3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9383 control reg</td>
<td>7</td>
<td>--&gt;</td>
<td>A2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(B1)</td>
<td>6</td>
<td>--&gt;</td>
<td>A1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9384 data reg</td>
<td>7</td>
<td>&lt;--</td>
<td>D7</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(A2)</td>
<td>6</td>
<td>&lt;--</td>
<td>D6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>EPROM data</td>
<td>5</td>
<td>&lt;--</td>
<td>D5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9385 control reg</td>
<td>7</td>
<td>&lt;--</td>
<td>D4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(A2)</td>
<td>6</td>
<td>&lt;--</td>
<td>D3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>D2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>D1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>D0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- DDR = FFH for data write
- 00H to load DDR
- 04H to load addresses
- 00H to load DDR
- 04H to transfer data
DDR = 00H for data read
00H to load DDR
04H to transfer data

; data reg
; (B) command/
; status
; 7 --> ~verify 1 1
; 6 --> ~write 1 1
; 5 --> ~read 1 1
; 4 <-- ~En 0 x
; 3 --> ~drn 1 1
; 2 --> ~G 1 1
; 1 --> ~E 1 1
; 0 --> ~Vppon 1 1

; control reg
; (B2)
; 7 --> OOH to load DDR
; 6 04H to transfer commands/status
; 5
; 4
; 3
; 2
; 1
; 0

; note:
; Initial data for the command/status register should be loaded
; before the I/O bits are set to output.
; --> means into EPROM
; <-- means out of EPROM
; <-> means bidirectional data flow
end