

ME27000

Martin Eberhard's Orphan EPROM Programmer II

User's Manual

ME27000

Martin Eberhard's Orphan EPROM Programmer II

Rev C PC Boards, Rev 1.02 Firmware

The ME27000 is a "universal" programmer, designed to program 24-pin and 28-pin EPROMs up to 64K-byte (512K-bit) EPROMs. Aside from the "standard" EPROMs (2708, 2716, 2732, 2764, 27128, 27256, 27512) the ME27000 is designed to program many of the "orphan" NMOS and CMOS EPROMs and EEPROMs - those with non-standard voltages, peculiar pinouts, proprietary programming algorithms, etc., especially those EPROMs that cannot be programmed by most other "universal" EPROM programmers. In particular, the ME27000 supports single-voltage and 3-voltage variants of the 2708 and 2716, all variants of the 2732, programming voltages from 12V through 26V, most 8K-byte EPROMs in 24-pin packages, and "Skinny DIP" packages as well as standard-width packages. It has provisions for an external negative programming voltage supply, to support EPROMs such as the Intersil IM6654, which needs $V_{pp} = -40V$.

The ME27000 supports many 28XXX-type EEPROMs (with or without software locking), as well as several of the 29XXX-type Flash EPROMs.

The list of supported EPROMs in Appendix A only includes EPROMs for which datasheets were found, so that programming compatibility could be verified. Most of these EPROMs (including the Soviet and East German variants) have been tested on the ME27000.

The ME27000 can program with the slow-and-standard methods, as well as pretty much any of the Smart/Quick/Fast/Express algorithms specified by the various EPROM manufacturers. For the supported EPROMs, the algorithms provided are faithful implementations of the algorithms specified by their manufacturers.

The ME27000 requires no special host-side software. It is completely self-contained and menu-driven, requiring only a terminal program (e.g. Hyperterm or Teraterm) that can send and receive ASCII files, and a 9600-baud RS232 serial port. It will accept and produce EPROM image files in either Intel Hex format or Motorola S-Record format.

A unique feature of the ME27000 is the Custom EPROM Editor, which lets you define and save (in onboard EEPROM) up to five custom EPROM specifications. You can specify the functions for many of the pins, the programming voltage, special voltage requirements for Vcc during programming, custom programming algorithm, etc. If your EPROM is not supported, and it has its data pins and address pins A0 through A9 in the standard locations, then you probably can create a custom EPROM spec to program it with the ME27000.

The ME27000 firmware can be updated via its serial port. If a future firmware release supports an EPROM that you need to program, or fixes a bug that's been bugging you, you can easily update the firmware in your ME27000.

The ME27000 can attempt to read the EPROM's ID bytes. If the EPROM supports this feature and the particular EPROM is supported by the ME27000, the the ME27000 will set itself up automatically for the EPROM. If the EPROM does not support the ID feature, then it is a good idea not to use the ID command.

The ME27000's uses a universal 12V AC adapter that is rated for line input from 100V to 240V, 50Hz or 60 Hz, so it should work anywhere in the world. The menus and this manual, however, are only in English.

-Martin Eberhard
13 February 2024

ME27000 Revision History

| PCB | Firmware | Manual | Date | Change Notes |
|-----|----------|--------|-------------|---|
| A | 1.00 | Prelim | 17 JUL 2022 | First complete version |
| B | 1.00 | B-00 | 19 FEB 2023 | Rev B boards, new commands, etc. |
| C | 1.00 | C-00 | 06 APR 2023 | Rev C boards, etc. |
| C | 1.00 | C-01 | 09 APR 2023 | Some typos fixed. Substitute KSC2690 for Intersil Option |
| C | 1.00 | C-02 | 13 APR 2023 | Various typos fixed. Fix Cypress 27C256 & 27C128 ID's. (Note: CY27C128 does not program properly.) Specify different 2.2K 1/2W resistor with correct body size. |
| C | 1.01 | C-03 | 14 APR 2023 | Add resistor network to BOM & assembly instructions |
| C | 1.01 | C-04 | 14 APR 2023 | Correct 2N3906 placement |
| C | 1.01 | C-05 | 14 APR 2023 | Fix RP1 installation in assembly instructions |
| C | 1.01 | C-06 | 18 APR 2023 | Fix diodes in BOM and assembly instructions. (Thanks, Gary!) |
| C | 1.01 | C-07 | 20 APR 2023 | Correct resistor pack part number |
| C | 1.01 | C-08 | 23 APR 2023 | Change C5 & C6 from 10 uF to 47 uF. (This makes the MAX660 work better with EPROMs that draw more current from their -5V pin. This is not an issue with the ADM660, but the higher capacitance does not hurt either.) |
| C | 1.01 | C-09 | 24 APR 2023 | Fix typos in the manual. (No circuit or functional changes.) |
| C | 1.01 | C-10 | 3 MAY 2023 | C5, C6 back to 10 uF. Jumper pin 1 to pin 8 of MAX660 (but not ADM660) |
| C | 1.01 | C-11 | 8 MAY 2023 | Adjust R7, R9, R11 for better Vpp adjustment range |
| C | 1.01 | C-12 | 10 MAY 2023 | Fix typo on page 43. Specify Digikey part number for C3. |
| C | 1.01 | C-13 | 13 FEB 2024 | Better component for C3: 0.2" lead spacing, 50V |
| C&D | 1.02 | D-01 | 15 MAR 2024 | Support Rev D PCBA. Correction for Intersil EPROMs (Delete C31) |
| C&D | 1.02 | D-02 | 18 MAR 2024 | Fix typo in assembly step 8 |

Contents

| | |
|---|----|
| Section 1. Getting Started..... | 1 |
| Section 2. ME27000 Programmer Usage..... | 3 |
| 2.1 Worldwide Operation..... | 3 |
| 2.2 Serial Port Connector Pinout..... | 3 |
| 2.3 USB Port..... | 3 |
| 2.4 LEDs..... | 3 |
| 2.5 Manual Voltage Adjustment..... | 4 |
| 2.6 Intersil Option..... | 4 |
| 2.7 Power Supply Voltage Checking..... | 5 |
| 2.8 Programming an EPROM from a File..... | 5 |
| 2.9 Reading an EPROM into a File..... | 6 |
| 2.10 Copying an EPROM..... | 6 |
| 2.11 File Address Offset..... | 6 |
| 2.12 Buffer Address Offset..... | 7 |
| 2.13 Data Invert..... | 9 |
| Section 3. ME27000 Commands..... | 10 |
| 3.1 EPROM Commands..... | 10 |
| 3.2 File Transfer Commands..... | 12 |
| 3.3 Buffer Commands..... | 14 |
| 3.4 Miscellaneous Commands..... | 15 |
| 3.5 Diagnostic Commands..... | 15 |
| Section 4. Custom EPROM Editor..... | 18 |
| 4.1 CEE General Commands..... | 18 |
| 4.2 CEE Pin Assignment Commands..... | 19 |
| 4.3 CEE Programming Parameter Commands..... | 21 |
| Section 5. Algorithms..... | 24 |
| 5.1 Simple Programming Algorithm..... | 25 |
| 5.2 Smart Programming Algorithms..... | 26 |
| 5.3 EEPROM Software Lock and Unlock Algorithms..... | 28 |
| 5.4 Device ID Algorithm..... | 29 |
| Section 6. ME27000 Theory of Operation..... | 31 |
| 6.1 Architecture..... | 31 |
| 6.2 Microcontroller..... | 31 |
| 6.3 Logic Supplies..... | 31 |
| 6.4 +6.35V Supply..... | 31 |
| 6.5 Negative 5V Supply..... | 32 |
| 6.6 Microcontroller-Controlled High-Voltage Supply..... | 32 |
| 6.7 EPROM Digital Pin Interface..... | 32 |

| | |
|--|----|
| Section 7. Downloading Firmware via the Serial Port..... | 33 |
| 7.1 Firmware Download Instructions | 33 |
| 7.2 Intel Hex File Format for Firmware Downloads | 34 |
| Section 8. ME27000 Programmer Assembly..... | 35 |
| 8.1 Printed Circuit Board Assembly | 36 |
| Section 9. Checkout and Adjustment..... | 41 |
| 9.1 Basic PCBA Checkout | 41 |
| 9.2 Microcontroller Bring-Up | 43 |
| 9.3 Microcontroller-Assisted Checkout and Adjustment | 44 |
| Section 10. Functional Testing..... | 51 |
| 10.1 Basic Buffer Operations and File Transfer | 51 |
| 10.2 EPROM Reading and Programming | 55 |
| Section 11. Printed Circuit Board..... | 59 |
| 11.1 Bill of Materials | 59 |
| 11.2 Component Placement | 61 |
| 11.3 PCBA Schematic | 62 |
| Appendix A. Supported Devices..... | 67 |
| Supported Devices Sorted by Device Type | 67 |
| Supported Devices Sorted by Manufacturer | 72 |
| Device Manufacturer Codes | 75 |
| Device Codes | 76 |

Section 1. Getting Started

This section is a quick overview of basic ME27000 operation. Follow these steps to load a hex file into the ME27000's buffer, and then program the buffer contents into an EPROM.

1. Set up your computer's terminal program (and RS-232 port if you are connecting that way) for 9600 baud, no handshaking.
2. Connect the ME27000 to your computer's USB port, or its RS232 port with a straight-through 9-pin male to 9-pin female cable, plug it in, and turn it on. You should see the startup message in your terminal program's window:

```
*=====*
```

```
*                ME27000                *
```

```
*=====*
```

```
*      Orphan EPROM Programmer II      *
```

```
*      By Martin Eberhard              *
```

```
*      Firmware Version  1.00          *
```

```
*=====*
```

```
Serial Number: C001
```

```
Current Device Type is 00: 2704
```

```
EPROM data invert off
```

```
Type ? for command list
```

```
>
```

(The "Current Device Type" is saved when the ME27000 is turned off.)

3. Type "EL" for a list of supported EPROM Device Types. Find your EPROM Device Type, and note its index number (the 2-digit number from the EL list). Use the "ET" command to select that EPROM Device Type, for example,

```
>ET 1D
```

```
Current Device Type is 1D: TMS2532
```

```
>
```

If you are unsure if this is the correct type of EPROM, use the "ETD" command to see the EPROM pinout, as well as a list of the manufacturer part numbers supported by this EPROM Device Type. For example:

```
>ETD
```

```
Type 1D: TMS2532, size: 4096 x 8
```

```
-----v-----
```

```
-| x          x |-
```

```
-| x          x |-
```

```
-----v-----
```

```
A7 -| 1      24 |- Vcc      Programming Vcc = 5V
```

```
A6 -| 2      23 |- A8
```

```
A5 -| 3      22 |- A9
```

```
A4 -| 4      21 |- Vpp 25.2V
```

```
A3 -| 5      20 |- -CE/-PGM
```

```
A2 -| 6      19 |- A10
```

```
A1 -| 7      18 |- A11
```

```
A0 -| 8      17 |- D7      Supported Devices:
```

```
D0 -| 9      16 |- D6      Hitachi   HM62532
```

```
D1 -| 10     15 |- D5      SGS       M2532
```

```
D2 -| 11     14 |- D4      TI        TMS2532
```

```
GND -| 12     13 |- D3
```

```
-----v-----
```

```
Vpp during read: 5.0V
```

```
Programming pulse on PGM pin
```

```

Programming pulse width: 50 mS
Programming algorithm:
  Pass 1: write each byte once
          Maximum P=1

```

- >
4. Send an Intel hex file of the EPROM image from your terminal program to the ME27000. The ME27000 will echo the data as it is sent. Any errors will be flagged at the end of the line with a question mark followed by a 3-letter error code (e.g. "?Csm" for a checksum error). When the file finishes loading, the ME27000 will give a count of the errors, as well as a count of the records that were successfully loaded into the buffer. If the error count is not 0, then check your serial port connection and setup, and try again.
 5. Insert a blank EPROM of the selected type into the ZIF socket. If it is a 24-pin EPROM, then position it toward the bottom of the socket, farthest from the ZIF socket's handle.
 6. Program the EPROM with the "EP" command.

```
>EP
```

```
Check the Device Type, and that the EPROM is inserted with its pin 1
in the ZIF socket's pin 3.
```

```
Ready to program (Y/N)? Y
```

```
Programming -
```

```
Verifying
```

```
EPROM matches buffer
```

```
>
```

The "-" following the word "Programming" will spin like a propeller while the EPROM is being programmed. Depending on the EPROM Device Type, this may take several minutes.

Once the programming completes, the ME27000 will verify the programming by comparing the EPROM contents to the buffer contents.

7. When the prompt returns and the Busy LED turns off, remove the programmed EPROM from the ZIF socket.

That's all there is to it! Once you get to know the rest of the ME27000 commands, you will be able to read an EPROM and upload its contents as a hex file, edit the EPROM data and write it back to another EPROM, and a whole lot more.

Section 2. ME27000 Programmer Usage

The previous section walked you through the most basic ME27000 Programmer operation. Here are some specifications, and the procedures for some more common EPROM operations.

2.1 Worldwide Operation

The ME27000's AC adaptor is a world-wide adapter, suited for 50 Hz or 60 Hz, 90VAC to 260V. The adapter's output is regulated 12VDC, 1.5A minimum.

2.2 Serial Port Connector Pinout

The DA-9 connector is compatible with standard PC serial port.

| Female DA-9 Pin | Signal |
|-----------------|-------------------------------|
| 2 | Data Out (out of the ME27000) |
| 3 | Data In (in to the ME27000) |
| 5 | Ground |

For a normal PC connection, you will need a standard straight-through male DA-9 to female DA-9 cable like this. (Other pins don't matter.)

| Male DA-9 Pin | Signal | Female DA-9 Pin |
|---------------|-------------------------------|-----------------|
| 2 | Data Out (out of the ME27000) | 2 |
| 3 | Data In (in to the ME27000) | 3 |
| 5 | Ground | 5 |

2.3 USB Port

The ME27000 has a USB Type B connector that can be used instead of the RS-232 connector. This USB port presents itself as a 9600 baud COM port to a PC. Once the USB link is established, the ME27000 behaves exactly the same as when connected via RS-232.

Most versions of Windows already have a driver that supports this USB port. If not, you will need a USB driver that is for the Microchip MCP2221A (which is the USB chip on the ME27000).

The USB port takes precedence over the RS-232 port - if both are connected to a computer, only the USB port will be active.

The blue USB LED will light when the USB port is connected to a USB host and has been enumerated on the USB interface.

2.4 LEDs

Four LEDs tell you the most basic state of the programmer.

- The white 'POWER' LED indicates that the AC adapter is energized and the power switch is on.
- The amber LED indicates that the ME27000 is 'BUSY'. When lit, the EPROM socket is energized, and an EPROM should not be removed or inserted.
- The red "ERROR" LED is lit whenever an error (such as a programming error) has been detected for the most recent command. Typing anything on the keyboard clears this error state.
- The blue 'USB' LED is lit whenever the ME27000 has successfully connected to a USB host.

2.5 Manual Voltage Adjustment

The five voltage adjustments on the ME27000 are normally adjusted to "nominal" voltages, which allow programming of most EPROMs without further adjustment. Four of these adjustments (VR1 through VR4) adjust four Vpp settings, while the fifth adjusts a high Vcc setting that is used to program some EPROMs.

The nominal Vpp settings are adjusted using the AVPP command, while measuring the voltage at TP6 (with the meter grounded on TP 1). Type "AVPP 1", and then adjust VR1 until TP6 measures 12.8V. Then do the same for the other three adjustments.

| AVPP | Adjustment | Nominal TP6 Voltage |
|------|------------|---------------------|
| 0 | - | About 11.5V |
| 1 | VR1 | 12.80V ± 0.05V |
| 2 | VR2 | 13.20V ± 0.05V |
| 3 | VR3 | 21.10V ± 0.05V |
| 4 | VR4 | 25.20V ± 0.05V |

Adjust the programming Vcc by measuring the voltage at TP7 and adjusting VR5. The nominal voltage for this adjustment is 6.25V ± 0.03V.

A few of the more obscure EPROMs supported by the ME27000 require you to adjust the programming Vcc manually. If your selected EPROM Device Type requires this Vcc adjustment, then a message will prompt you to make the adjustment. (When you later select an EPROM that programs with nominal voltages, you will also be prompted to readjust the programming Vcc supply.)

2.6 Intersil Option

The Intersil IM6654 and IM6658 (and perhaps some other EPROMs) require a negative Vpp programming voltage, rather than the positive voltages required by practically every other NMOS EPROM. The ME27000 can program EPROMs that require a negative Vpp with the installation of the *Intersil Option* components, and the attachment of an external, regulated negative power supply (-41 volts for the Intersil IM6654, -31V for the IM6658).

The Intersil Option requires installation of the following components, not normally installed in the ME27000. (The inclusion of these components will not affect programming other EPROM Device Types.)

| √ | Qty | Location | Value | Digikey Part Number |
|---|-----|----------|----------------------------------|---------------------|
| | 1 | R65 | 6.8 1/4W Resistor | S6.8HCT-ND |
| | 1 | R63 | 330 ohm 1/4W Resistor | 330QBK-ND |
| | 1 | R64 | 1K ohm 1/4W Resistor | CF14JT1K00CT-ND |
| | 2 | D20,D21 | 1N4004 Diode | 1N4004-TPMSCT-ND |
| | 1 | C31 | NO COMPONENT | --NO-- |
| | 1 | C32 | 33 µF 50V Electrolytic Capacitor | P5180-ND |
| | 1 | Q23 | 2N6520 Transistor | 2N6520TACT-ND |
| | 1 | Q24 | KSC2690 Transistor | KSC2690AYSFS-ND |
| | 1 | J6 | 3-pin male 0.1" connector | S9411-ND |

When programming an Intersil IM6654 or IM6658 EPROM, connect an external power supply to J6:

| Pin | IM6654 Programming Function | IM6658 Programming function |
|-----|-----------------------------|-----------------------------|
| 1 | 41 volt positive terminal | 31 volt positive terminal |
| 2 | 41 volt negative terminal | 31 volt negative terminal |
| 3 | No Connection | No Connection |

Select the EPROM Device Type (04 for the IM6654 or Type 28 for the IM6658), and program the EPROM.

2.7 Power Supply Voltage Checking

The ME27000 tests Vpp during programming, and will abort programming if the measured voltage is more than about 15% high or low from the expected voltage. However, the external (Intersil Option) supply does not get checked.

Vpp will be low if the EPROM is drawing too much current - because the EPROM is defective, because it is inserted incorrectly, or if the wrong EPROM Device Type is selected. Although the ME27000 will shut down quickly when overcurrent is detected this way, the EPROM may still be damaged.

Vpp will be too high only due to some fault with the ME27000 itself.

2.8 Programming an EPROM from a File

Here is how you program an EPROM from a file, using the (default) automatic file address offset mode, and assuming an EPROM that supports the Device ID function. (Use the ET command to select the EPROM type, for EPROMs that don't support the Device ID function.)

```
{Power-on}
>ET {correct EPROM Device Type}
{insert blank EPROM in the socket}
> {send S-Record or Intel Hex file to the ME27000}
>ID
Device ID: 00200D
Device: ST/SGS M27512
Type 4F
Use this device type (Y/N)? Y
Current Device Type is 4F: 27C512
>EP
Buffer Address Offset: 00
Check the Device Type, and that the EPROM is
inserted with its pin 1 by the socket handle.

Ready to program (Y/N)? Y
Programming -
Verifying
EPROM matches buffer
>
```

2.9 Reading an EPROM into a File

You can read an EPROM, and save the data in a standard format (either Intel Hex or Motorola S-Record) on your computer. Again, use the ET command if your EPROM does not support the Device ID function.)

```
{Power-on}
>ID
Device ID: 002013
Device: ST/SGS M2732A
Type 18
Use this device type (Y/N)? Y
Current Device Type is 18: 2732A-Fast
>ER
Buffer Address Offset: 00
EPROM read into buffer. checksum: B2
>UI {start file capture on your computer before hitting return}
{Intel Hex file follows}
>
```

Use **US** instead of **UI** to create a file in Motorola S-Record format.

2.10 Copying an EPROM

You can read an EPROM, and then write it into any number of blank EPROMs. (This example uses the ET command instead of ID, just to show how it's done.)

```
{Power-on}
>ET {correct EPROM Device Type}
{insert source EPROM in the socket}
>ER
Success
{insert blank EPROM in the socket}
>EP
Buffer Address Offset: 00
Check the Device Type, and that the EPROM is
inserted with its pin 1 by the socket handle.

Ready to program (Y/N)? Y
Programming -
Verifying
EPROM matches buffer
{insert another blank EPROM in the socket}
>EP
Check the Device Type, and that the EPROM is
inserted with its pin 1 by the socket handle.

Ready to program (Y/N)? Y
Programming -
Verifying
EPROM matches buffer
>
```

2.11 File Address Offset

Simple Intel Hex files and Motorola S-record files have 2-byte (4 hex digit) addresses. The address in a hex file can be thought of as having 2 parts: The high byte (the first 2 hex digits) is the page address and can be 00 through FF (depending on EPROM size). The low byte (the second 2 hex digits) is the address within the EPROM page.

The EPROM data may be a computer program intended to run at address 0000, or

it may be intended to run at some other address. If it is intended to run at some other address besides 0000, then the high byte of the 2-byte addresses in the hex file that you will send to the EPROM will not start at 00. Instead, it will start with the high byte of the intended target address.

You can use FAO (File Address Offset) command to compensate for this address, so that the loaded file goes in the buffer starting at 0000, regardless of the address in the file. If you specify automatic File Address Offset mode (by typing the FAO command with no value), then the File Address Offset for downloads will be set to the high address byte from the first received record of the file. Automatic File Address Offset mode is the default after reset.

When you download a hex file to the ME27000, the File Address Offset is subtracted (modulo 10000) from the high address bytes for the data in the file.

If you are reading an EPROM that is intended to run at an address besides 0000, you can generate the correct hex file by setting the File Address Offset before uploading the buffer. The File Address Offset that you specify with the FAO command will be added to the high address byte in every record uploaded. If automatic File Address Offset mode is selected, then the File Address Offset will be 00.

2.12 Buffer Address Offset

The Buffer Address Offset allows you to specify the high byte of the buffer starting address for EPROM operations, including reading (ER command), comparing (EC command), and programming (EP command). The specified Buffer Address Offset is added to the high byte of the EPROM address to compute the high byte of the buffer address for these operations.

Using the Buffer Address Offset, you can load a large file or read a larger EPROM (as large as 64K bytes), and then program it into several smaller EPROMs. For example, if you have already loaded a 4K-byte Intel Hex file into the buffer, then you could program it into four 2708 (1Kx8) EPROMs:

```
>ET 5
Current Device Type is 05: 2708
>BAO 0
Buffer Address Offset: 00h
{insert first 2708 EPROM}
>EP
Check the Device Type, and that the EPROM is
inserted with its pin 1 in the ZIF socket's pin 3.

Ready to program (Y/N)? Y
Programming -
Verifying
EPROM matches buffer
>BAO 4
Buffer Address Offset: 04h
{insert second 2708 EPROM}
>EP
Check the Device Type, and that the EPROM is
inserted with its pin 1 in the ZIF socket's pin 3.

Ready to program (Y/N)? Y
Programming -
Verifying
EPROM matches buffer
```

```
>BAO 8
Buffer Address Offset: 08h
{insert third 2708 EPROM}
>EP
Check the Device Type, and that the EPROM is
inserted with its pin 1 in the ZIF socket's pin 3.
```

```
Ready to program (Y/N)? Y
Programming -
Verifying
EPROM matches buffer
```

```
>BAO C
Buffer Address Offset: 0Ch
{insert fourth 2708 EPROM}
>EP
Check the Device Type, and that the EPROM is
inserted with its pin 1 in the ZIF socket's pin 3.
```

```
Ready to program (Y/N)? Y
Programming -
Verifying
EPROM matches buffer
```

You can also read several smaller EPROMs into the buffer, and then program their combined data into one single larger EPROM:

```
>BAO 0
Buffer Address Offset: 00h
{insert first 2708 EPROM}
>ER
EPROM read into buffer, checksum 12
>BAO 4
Buffer Address Offset: 04h
{insert second 2708 EPROM}
>ER
EPROM read into buffer, checksum BE
>FAO 8
Buffer Address Offset: 08h
{insert third 2708 EPROM}
>ER
EPROM read into buffer, checksum 87
>BAO C
Buffer Address Offset: 0Ch
{insert first 2708 EPROM}
>ER
EPROM read into buffer, checksum 91
>ET 1C
Current Device Type is 1C: TMS2532
{insert blank TMS2532 EPROM}
>BAO 0
Buffer Address Offset: 00h
>EP
Check the Device Type, and that the EPROM is
inserted with its pin 1 in the ZIF socket's pin 3.
```

```
Ready to program (Y/N)? Y
Programming -
Verifying
EPROM matches buffer
```

2.13 Data Invert

If your EPROM will be used in a host system that has inverting data buffers (such as the Memory Merchant MM65K16S S-100 memory board), then you can specify Data Inversion with the "DI 1" command. This will invert the data when written to the EPROM, as well as when it is read back from the EPROM. "DI 0" disables the data invert mode.

Section 3. ME27000 Commands

3.1 EPROM Commands

The EPROM commands generally deal with the EPROM, and the 64K-byte buffer in the ME27000. For Compare, Program, and Read commands, the buffer address always equals the EPROM address plus the Buffer Address Offset (set with the BAO command).

For these EPROM commands, <beg> is the beginning address, both for the EPROM and for the buffer. <cnt> is the byte count for the command. <cnt>=0000 is interpreted as <cnt>=10000 hex (the entire buffer) when the command refers to the buffer, and is interpreted as the full EPROM size when the command refers to the EPROM. If you don't enter values for <beg> and <cnt>, they both default to 0000. If you enter only one value, it is assumed to be <beg>, and <cnt> defaults to 0000. The EPROM commands will always stop after the last address of the selected EPROM Device Type.

>BAO <offset> **Buffer Address Offset**

BAO sets the Buffer Address Offset for reading, comparing, and programming EPROMs to <offset>. The Buffer Address Offset is added to the upper address byte of the buffer's starting address of the ER, EC, and EP commands.

See the **Buffer Address Offset** subsection under **ME27000 Programmer Usage** for further details about the Buffer Address Offset, and how it is used.

>DI {0/1} **Data Invert**

DI 1 selects inverted data mode. (This is useful when the EPROM will be used in a system that has inverting data buffers. When Data Invert is enabled, all data bytes are inverted when the EPROM is programmed, and also when the EPROM is read. DI 0 turns off Data Invert mode.

>EB <beg> <cnt> **Blank-Check EPROM**

EB starts at address <beg> and checks <cnt> bytes of the EPROM to see if they are blank. Any non-blank bytes are reported to the console. If all bytes in the specified range are blank, this command responds with "EPROM is blank". Note that a few EPROMs erase to 00 instead of FF. Also, a few EPROMs erase to an indeterminate state.

>EC <beg> <cnt> **Compare EPROM to Buffer**

EC compares <cnt> bytes of the buffer to the EPROM, starting at address <beg>. Any differences are reported to the console. If all bytes are the same, this command responds with "EPROM matches buffer".

>ED <beg> <cnt> **Display EPROM Contents**

ED displays the specified range of EPROM contents (without affecting the buffer), 16 hex bytes to a line, (except the first line if its least-significant digit is not 0). Each line is preceded by a 4-digit hex address. After the hex display, the same data are displayed in ASCII. Non-printing ASCII characters are represented as a period. If you don't enter values for <beg> and <cnt>, then the entire EPROM contents will be displayed. The checksum of the specified region of the EPROM is printed last.

You can pause and resume the display with the space bar, and abort with either <Control-C> or ESC.

>EE <Device Type> Invoke Custom EPROM Editor

The last 4 EPROM Device Types are custom EPROMs, allowing you to define your own EPROM. The EE command allows you to edit the specified EPROM, which must be one of the last four listed EPROMs. See section 4 for details about editing a custom EPROM.

>EL List supported EPROMs

EL lists all of the EPROM Device Types, for use with the ETD, EE, and ET commands.

>EP <beg> <cnt> Program EPROM from Buffer

EP programs <cnt> bytes of the EPROM with data from the buffer, with the EPROM starting at <beg>, and the buffer address starting at <beg> plus the Buffer Address Offset. The programmed range is verified when programming is complete, and any errors are reported to the console. If the programmed range matches the buffer when done, then this command will respond with "Success".

Before programming, the relevant states of the ME27000 (Programming algorithm, programming cycles) are displayed, and you are asked if you want to proceed.

Typing control-C or ESC during programming will abort cleanly, leaving the EPROM in a reasonable state.

See Section 5, **Programming Algorithms**, for additional details.

>ER <beg> <cnt> Read EPROM into Buffer

ER reads <cnt> bytes of data from the EPROM into the buffer, the EPROM starting at <beg>, and the buffer address starting at <beg> plus the Buffer Address Offset. This command always responds with "EPROM read into buffer" followed by "EPROM checksum: XX".

>ES <beg> <cnt> Compute EPROM Checksum

ES reads and adds together <cnt> bytes of data from the EPROM, starting at address <beg>. Only the low byte of the sum is kept. This command responds with "EPROM checksum: XX".

>ET <index> Select EPROM Device Type

ET sets the current EPROM Device Type for all other operations. Selecting an <index> that specifies a custom EPROM that has not yet been defined produces an error.

>ETD <Device Type> Display EPROM Specs

ETD displays details about the specified EPROM Device Type, including a picture of the EPROM's pinout and its programming specifications, as well as a list of manufacturer's part numbers for EPROMs that are compatible with this Device Type. If no <Device Type> is specified, then the currently-selected EPROM Device Type will be displayed.

With a maximum of 12 entries, the list of supported EPROMs is not complete. If your EPROM is not listed, then compare its specifications to those displayed with the ETD command, to verify compatibility.

>ID Read and Display the EPROM's ID

ID reads, decodes, and displays the device ID, from EPROMs that support this function. If a valid ID is found for a device type that is different than the currently-selected device type, then you are given the option to select this device type.

This function applies +12V to pin 24 of the ZIF socket (the pin that is normally address line A9). 12 volts is out of spec for earlier EPROMs that do not support the ID function. The ME27000 tries to detect if the EPROM does not support the ID function by limiting the current to this pin, measuring the pin's current when 12V is applied, and shutting down quickly if excess current is detected.

>LOCK Lock an EEPROM or Flash Device

This will activate the software lock in an EEPROM or Flash device that supports this feature. It is not allowed for devices that don't support software locking.

>UNLOCK Unlock an EEPROM or Flash Device

This will deactivate the software lock in an EEPROM or Flash device that supports this feature. It is not allowed for devices that don't support software locking.

?E Help with EPROM Commands

?E prints a help screen for these EPROM commands.

3.2 File Transfer Commands**>FAO Automatic File Address Offset Mode (default)**

FAO (with no parameter) selects automatic File Address Offset mode. When downloading a file to the ME27000, the File Address Offset will be taken from the high address byte of the first received record. When uploading a file from the ME27000, the File Address Offset will be 00.

>FAO <offset> Set File Address Offset

FAO sets the File Address Offset for uploads and downloads to <offset>. The File Address Offset is added to the upper address byte when uploading buffer data, and subtracted from the address of downloaded Intel Hex records and Motorola S-Records.

See the **File Address Offset** section for details about the File Address Offset, and how it is used during uploading and downloading.

>UI <beg> <cnt> Upload Buffer as Intel Hex

UI prints <cnt> bytes of the buffer contents, starting at address <beg>, on the console as Intel Hex files. The File Address Offset is added to the high address byte for each record. All records (except perhaps the last one) are 16 bytes long, and the transfer ends with an Intel Hex end-of-file (type 01) record. To upload into a file, start the file capture after you type UI, but before you type <return>.

If no value is provided for <cnt> then the byte count will be set to the size of the currently selected EPROM Type.

>US Upload Buffer as Motorola S-Records

US prints <cnt> bytes of the buffer contents, starting at address <beg>, on the console as Motorola S-record files. The File Address Offset is added to the high address byte for each record. All records (except perhaps the last one) are 16 bytes long, and the transfer ends with a Motorola S-record end-of-file (S9) record. To upload into a file, start the file capture after you type US, but before you type <return>.

If no value is provided for <cnt> then the byte count will be set to the size of the currently selected EPROM Type.

>: Begin Intel Hex record

Buffer addresses are calculated by subtracting the File Address Offset from the address in each record, and then incrementing after each data byte of the record has been handled.

If the record type is 00 (a data record), then all data whose target buffer address is between 0000 and FFFF will be loaded into the buffer.

If the record's checksum does not match the computed checksum, then "? Csm" will be printed, and the error count will be incremented.

Invalid hex characters (including lowercase a-f) print "? Hex" and cause the error count to be incremented.

Record types other than 00 (data) and 01 (end-of-file) print "? Rec" and cause the error count incremented.

Byte count other than 00 for a type 01 record (end-of-file) print "? Rec" and cause the error count to be incremented.

The prompt is not normally displayed after receipt of an Intel Hex record. If the record type is either 00 (data) or 01 (end-of-file), and the record byte count is 00, then the record count, the count of records loaded into the buffer, and error count are displayed and then cleared, and the prompt is displayed.

>S Begin Motorola S-record

(Note that no commands start with S.)

Buffer addresses are calculated by subtracting the File Address Offset from the address in each record, and then incrementing after each data byte of the record has been handled.

If the record type is S1 (a data record), then all data whose target buffer address is between 0000 and FFFF will be loaded into the buffer.

An S5 (record count) record will compare the ME27000's record count to the record count in the record. If they do not match, then "? Cnt" is printed, and the error count is incremented.

If the checksum in the record does not match the computed checksum, then "? Csm" is printed and the error count is incremented.

Invalid hex characters (including lowercase a-f), will print "? Hex" and cause the error count to be incremented.

Any record type besides S1 (data), S5 (record count), and S9 (end-of-file) will print "? Rec" and cause the error count incremented.

An end-of-file record may be a full S9 record (with byte count,

address, and checksum fields), or it may be just 'S9'. (This abbreviated S9 record is sometimes found in old S-record files.)

If the byte count for a type S5 (record count) or S9 (end-of-file) record is not 00, then "? Rec" is printed, and the error count is incremented.

The prompt is not normally displayed after receipt of a Motorola S-record. If the record type is either S1 (data) or S9 (end-of-file), and the record byte count is 00, then the record count, the count of records loaded into the buffer, and error count are displayed and then cleared, and the prompt is displayed.

?F Help with File Transfer Commands

?F prints a help screen for these file transfer commands.

3.3 Buffer Commands

>BD <beg> <cnt> Display Buffer Contents

BD displays the specified range of buffer contents, 16 hex bytes to a line, (except the first line if its least-significant digit is not 0). Each line is preceded by a 4-digit hex address. After the hex display, the same data are displayed in ASCII. Non-printing ASCII characters are represented as a period. If you don't enter values for <beg> and <cnt>, then the entire buffer contents will be displayed. The checksum of the specified region of the buffer is printed last.

You can pause and resume the display with the space bar, and abort with either <Control-C> or ESC.

>BE <addr> Edit Buffer

BE allows you to edit the buffer contents starting at address <addr>. The address and its contents are first printed on the console. If you type <return>, the contents will remain unchanged. If you type a hexadecimal number and then <return>, the value you type will replace the buffer contents at that address.

After you type <return>, the contents of the next address in the buffer are displayed, allowing you to modify that address. This continues until you type <control-C> or ESC.

Every address that ends with 0 or 8 will start a new line, displaying first the address, then the data.

>BF <val> Fill Buffer with Value

BF fills the entire buffer with <val>. If you don't enter a value for <val>, then the buffer will be filled with 00.

This command responds with "Buffer filled with <val>".

?B Help with Buffer Commands

?B prints a help screen for these buffer commands.

3.4 Miscellaneous Commands

>DS Display all Settings

Displays various settings for the ME27000.

>ECHO {0/1} Set Terminal Echo

ECHO 0 turns terminal echo off; ECHO 1 turns it on.

>RESET Reset ME27000 Programmer

RESET is just like power-cycling the ME27000.

>? Help

Typing a question mark prints a help screen that briefly explains the main commands.

>?N General notes on ME27000

This command prints a page of general notes about the ME27000.

>?L View Firmware Loader Notes

?L displays notes on loading new firmware into the ME27000 via the serial port. Firmware loading is discussed in a later section.

^S Pause Serial Port Output

Control-S (XOFF) tells the ME27000 to stop sending data. Any subsequent character (including XON, which is control-Q) will re-enable the ME27000 output, and that character will be discarded by the ME27000.

3.5 Diagnostic Commands

These commands are intended only for diagnosing the ME27000, especially during initial bring-up and when you want to adjust the programming voltages. They allow you to control various signals to the EPROM socket directly, so that you can measure and adjust voltages, and test functionality.

For these commands, if you don't enter a value (0 or 1), then 0 is assumed.

Most of these commands will leave the BUSY light lit. Use the RESET command to turn it off prior to programming any EPROMs.

>AVPP {0-4} Adjust Vpp

AVPP allows you to adjust the five Vpp voltages, using a voltmeter at TP6. AVPP 0 disables the switching power supply, resulting in Vpp about 0.5V lower than the voltage from the wall adapter. The following table shows the nominal adjustment voltages for each setting:

| AVPP | Adjustment | Nominal TP6 Voltage |
|------|------------|---------------------|
| 0 | - | About 11.5V |
| 1 | VR1 | 12.80V \pm 0.05V |
| 2 | VR2 | 13.20V \pm 0.05V |
| 3 | VR3 | 21.10V \pm 0.05V |
| 4 | VR4 | 25.20V \pm 0.05V |

>TAS {0/1} Test -AS pin

If the selected EPROM Device Type has a -AS pin, then this command tests it. TAS 0 will set the -AS signal inactive, at TTL high (>3V). TAS 1 will set the -AS signal active, at TTL low (<0.7V).

>TCE {0/1} Test CE Pin

If the selected EPROM Device Type has a CE pin, then this command tests it. "Active" and "inactive" voltages for this pin depend on the polarity of the CE signal, for the selected EPROM Device Type. TCE 0 will set the CE signal inactive. TCE 1 will set the PGM signal active.

>THI {0/1} Test Stuck-High Pin

If the selected EPROM Device Type has a stuck-high pin, then this command will test it. THI 1 sets the pin to +5V, THI 0 sets low.

>TID {0/1} Test Device ID function

This applies +12V to ZIF pin 24, reads the actual voltage at the pin and reports the A/D converter's resulting value for that pin. The parameter applies either 0000 or 0001 to the address pins. If the ADC result is too low, then the +12V will be turned off.

>TOE {0-2} Test -OE Pin

If the selected EPROM Device Type has a -OE pin, then this command tests it. TOE 0 will set the -OE signal inactive, at TTL high (>3V). TOE 1 will set the -OE signal active, at TTL low (<0.7V). If the selected EPROM Device Type requires +12V on the -OE pin during programming, then TOE 2 will test this voltage on the -OE pin.

>TOFF Cancel all test modes

This command powers off the ZIF socket, de-energizing all pins. It also turns off the busy and error lights.

>TPGM {0/1} Test PGM Pin

If the selected EPROM Device Type has a PGM pin, then this command tests it. "Active" and "inactive" voltages for this pin depend on the polarity of the PGM signal, for the selected EPROM Device Type. TPGM 0 will set the PGM signal inactive. TPGM 1 will set the PGM signal active.

>TPROG Test Programming

TPROG will continuously write the buffer data to the EPROM socket, for the purpose of testing the ME27000 (not for programming the EPROM!) End this command with either Control-C or ESC. All of the EPROM signals and timing will be the same as during normal programming for the selected EPROM Device Type. If the programming algorithm is a "Smart" algorithm (with verification), then this loop will behave as though the EPROM verified on the last possible try. Together with an oscilloscope, this command is useful to test hardware, and to verify that a programming algorithm (particularly a custom algorithm) is correct.

>TREAD Test Reading

TREAD will continuously read the EPROM socket, for the purpose of testing the ME27000 (not for reading the EPROM!) End this command with either Control-C or ESC. All of the EPROM signals and timing will be the same as during normal reading for the selected EPROM Device Type. Data from the EPROM does not get written to the buffer.

>TVBD {0/1} Test Vbb and Vdd Pins

If the selected EPROM Device Type has a Vbb pin and/or Vdd pin, then this command tests them. TVBD 0 turns these pins off (near 0V). TVBD 1 sets the Vbb pin (pin 21) to -5V and the Vdd pin (pin 19) to +12V.

>TVCC {0-2} Test Vcc Pin

TVCC 0 turns Vcc (on pin 24) off. TVCC 1 turns it on to the normal voltage during reading: +5V. TVCC 2 turns pin 24 on to its programming voltage for the selected EPROM Device Type, which is one of the following: 0V, +5V, +6.2V, or +12V.

>TVPP {0-3} Test VPP Pin

This command tests the VPP pin, if it exists. TVPP 0 turns the Vpp pin off (near 0V). TVPP 1 sets the Vpp pin to the required voltage for a normal EPROM read operation (0V or 5V, depending on the selected EPROM Device Type). TVPP 2 sets the Vpp to its programming-inactive state (the voltage between write pulses, if this EPROM Device Type pulses the Vpp pin). This will be either 0V or 5V, depending on the EPROM Device Type. TVPP 3 sets the Vpp pin to its programming voltage.

>WA <val> Write Address Pins

WD writes <val> to the address pins of the EPROM socket. Every '0' bit will drive the corresponding pin to TTL low (<0.7V). Every 1 bit will drive the corresponding pin to TTL high (>3V).

>WD <val> Write Data Pins

WD writes <val> to the eight data pins of the EPROM socket. Every '0' bit will drive the corresponding pin TTL low (<0.7V). Every 1 bit will drive the corresponding pin to TTL high (>3V).

>RD Read Data Pins

RD reads the EPROM data pins and displays the results on the screen. Use a jumper wire to ground or 5V, to test each pin.

?D Help with Diagnostics

?D prints a help screen for these diagnostic commands.

Section 4. Custom EPROM Editor

The last five EPROMs displayed by the EP command are custom EPROMs. If any of these EPROMs has not been defined (meaning named), then it will be listed as "undefined" with the EL command. You can define a custom EPROM using the Custom EPROM Editor (CEE). Enter the CEE by typing EEnn at the main prompt, where nn is the EPROM Device Type number for one of the five custom EPROMs.

If you have an oscilloscope, you can use the TREAD and TPROG commands to observe the EPROM reading and programming waveforms (without an EPROM inserted in the socket) for custom EPROM Device Types that you have created.

The CEE's prompt is "EEnn>", where nn is the number of the EPROM that you are editing. Exit the CEE with the Q or Abort command. Until a name is assigned to a custom EPROM (with the EN command), it will appear as "unassigned" in the list of supported EPROMs.

4.1 CEE General Commands

EEnn>? **Print General Help Menu**

EEnn>?A **Print Pin Assignment Help Menu**

EEnn>?P **Print Programming Parameters Help Menu**

EEnn>COPY <Device Type> **Copy EPROM Specs**

Copy all specifications from EPROM the specified Device Type. Use this command when you want to create a new EPROM Device Type that is similar to an existing one. You may edit the copied EPROM Device Type specifications once it is copied.

EEnn>DELETE **Delete EPROM**

Deletes all parameters for the current EPROM Device Type. The current EPROM Device Type will then become "unassigned" in the EL command.

EEnn>ETD <Device Type> **Display EPROM Specs**

ETD displays details about the specified EPROM Device Type, including a picture of the EPROM's pinout and its programming specifications, as well as a list of manufacturer's part numbers for EPROMs that are compatible with this type. If no <Device Type> is specified, then the EPROM Device Type that you are editing will be displayed.

EEnn>EN <name> **Name EPROM**

Assigns name to current EPROM Device Type. Until a name is assigned to a custom EPROM Device Type, it will appear as "unassigned" in the list of supported EPROM Device Types displayed by the EL command.

EEnn>PINS <24/28> **Specify EPROM pin count**

Only 24- and 28-pin EPROMs are supported.

EEnn>TPROG **Test Programming**

TPROG will continuously write the buffer data to the EPROM socket, for the purpose of testing a custom EPROM Device Type (not for programming the EPROM!) End this command with either Control-C or ESC. All of the EPROM signals and timing will be the same as during normal programming for custom EPROM Device Type that you are editing. If the programming algorithm is a "Smart" algorithm (with verification), then this loop will behave as though the EPROM verified on the last possible try.

Together with an oscilloscope, this command is useful to verify that the custom EPROM Device Type's programming algorithm is correct.

EEnn>TREAD Test Reading

TREAD will continuously read the EPROM socket, for the purpose of testing the custom EPROM setup (not for reading the EPROM). End this command with either Control-C or ESC. All of the EPROM signals and timing will be the same as during normal reading for the EPROM. Data from the EPROM does not get written to the buffer.

EEnn>Q Quit the Custom EPROM Editor

Saves all changes and returns to the main command prompt.

4.2 CEE Pin Assignment Commands

The ZIF socket pin numbers for these commands depend on whether the EPROM is a 24-pin device or a 28-pin device, as set by the PINS command.

Pins 1, 2, and 20 through 27 on the ZIF socket (18 through 22 for a 24-pin device) are initially unassigned, and each may be assigned to one of several possible signals. (Due to hardware limitations, not every signal may be assigned to every pin. However, most EPROMs can be supported with the available pin choices.) If you assign more than one signal to the same pin, then the last one assigned will override previous assignments. There are a few exceptions to this rule:

- Vpp may be assigned to the same pin as either Output Enable or Chip Enable
- PGM may be assigned to the same pin as either Output Enable or Chip Enable

EEnn>A9 <pp> Assign A9 Signal to Pin

Signal A9 is assigned to the specified pin, where <pp> may be {0,1,2,20,21,22,23,24,25,26,27} for a 28-pin device, or {0,18,19,20,21,22,23} for a 24-pin device. 0 means unassign A9.

EEnn>A10 <pp> Assign A10 Signal to Pin

Signal A10 is assigned to the specified pin, where <pp> may be {0,1,2,20,21,22,23,24,25,26,27} for a 28-pin device, or {0,18,19,20,21,22,23} for a 24-pin device. 0 means unassign A10.

EEnn>A11 <pp> Assign A11 signal to pin

Signal A11 is assigned to the specified pin, where <pp> may be {0,1,2,20,21,22,23,24,25,26,27} for a 28-pin device, or {0,18,19,20,21,22,23} for a 24-pin device. 0 means unassign A11.

Enn>A12 <pp> Assign A12 Signal to Pin

Signal A12 is assigned to the specified pin, where <pp> may be {0,1,2,20,21,22,23,24,25,26,27} for a 28-pin device, or {0,18,19,20,21,22,23} for a 24-pin device. 0 means unassign A12.

EEnn>A13 <pp> Assign A13 signal to pin

Signal A13 is assigned to the specified pin, where <pp> may be {0,1,2,20,21,22,23,24,25,26,27} for a 28-pin device, or {0,18,19,20,21,22,23} for a 24-pin device. 0 means unassign A13.

EEnn>A14 <pp> Assign A14 Signal to Pin

Signal A14 is assigned to the specified pin, where <pp> may be {0,1,2,20,21,22,23,24,25,26,27} for a 28-pin device, or {0,18,19,20,21,22,23} for a 24-pin device. 0 means unassign A14.

EEnn>A15 <pp> Assign A15 Signal to Pin

Signal A15 is assigned to the specified pin, where <pp> may be {0,1,2,20,21,22,23,24,25,26,27} for a 28-pin device, or {0,18,19,20,21,22,23} for a 24-pin device. 0 means unassign A15.

EEnn>AS <pp> Assign Address Strobe Signal to Pin

The (active-high) Address Strobe Signal is assigned to the specified pin, where <pp> may be {0,1,2,20,21,22,23,24,25,26,27} for a 28-pin device, or {0,18,19,20,21,22,23} for a 24-pin device. 0 means unassign Address Strobe.

EEnn>CE <pp> Assign Chip Enable Signal to Pin

The Chip Enable Signal is assigned to the specified pin, where <pp> may be {0,1,2,20,21,22,23,24,25,26,27} for a 28-pin device, or {0,18,19,20,21,22,23} for a 24-pin device. 0 means unassign Chip Enable. Note that Chip Enable may share a pin with either Vpp or PGM.

EEnn>CEP {0/1} Assign polarity to Chip Enable Signal

CEP defines the polarity of the Chip Enable signal, where negative polarity is the default. 0 means negative polarity, 1 means positive polarity.

EEnn>HI <pp> Assign Stuck-High Pin

The specified pin will be stuck high, where <pp> may be {0,1,2,20,21,22,23,24,25,26,27} for a 28-pin device, or {0,18,19,20,21,22,23} for a 24-pin device. 0 means unassign Stuck-High pin. Note that any unassigned pins will be stuck-low.

EEnn>OEN <pp> Assign Output Enable Signal to Pin

The (active-low) Output Enable signal is assigned to the specified pin, where <pp> may be {0,1,2,20,21,22,23,24,25,26,27} for a 28-pin device, or {0,18,19,20,21,22,23} for a 24-pin device. 0 means unassign Output Enable. Note that Output Enable may share a pin with either Vpp or PGM. It also may be assigned to be at +12V during programming - see OEV command. (If the -OE pin assigned to be +12V during programming, then that pin may not also be assigned to the Vpp or PGM signals.)

EEnn>PGM <pp> Assign Program Signal to Pin

The Program Signal is assigned to the specified pin, where <pp> may be {0,1,2,20,21,22,23,24,25,26,27} for a 28-pin device, or {0,18,19,20,21,22,23} for a 24-pin device. 0 means unassign Program. Note that PGM may share a pin with either Chip Enable or Output Enable.

EEnn>PGP {0/1} Assign polarity to Program Signal

PGP defines the polarity of the Program signal, where negative polarity is the default. 0 means negative polarity, 1 means positive polarity.

EEnn>PPP <pp> Assign Vpp Signal to Pin

The Vpp Signal is assigned to the specified pin, where <pp> may be {0,1,20,22,23} for a 28-pin device, or {0,18,20,21} for a 24-pin device. 0 means unassign Vpp. Note that Vpp may share a pin with either Chip Enable or Output Enable.

EEnn>VBB {0/1} Assign Vbb to ZIF socket Pin 21

VBB 1 assigns -5V Vbb to ZIF pin 21. VBB 0 unassigns Vbb.

EEnn>VDD {0/1} **Assign Vdd to ZIF socket Pin 19**

VDD 1 assigns +12V Vdd to ZIF socket pin 19. VDD 0 unassigns Vdd.

4.3 CEE Programming Parameter Commands

EEnn>BCK {0/1/2} **Blank Check before Programming**

BCK 0 disables a blank-check before programming. BCK 1 enables blank check for a device that erases to 0x00. BCK 2 enables blank check for a device that erases to 0xFF. (Disable blank check for most EEPROMs, and for any EPROM Device Type that erases to an indeterminate state.)

EEnn>LWD (0/1) **Lock EEPROM When Done Programming**

"LWD 1" Performs the EEPROM software lock algorithm when programming is completed. "LWD 0" disables this feature. This command is ignored unless the Pass 1 algorithm to 10 (which unlocks the EEPROM)

EEnn>PIA <p> **Define Programming Pass 1**

Selects one of 12 possible programming algorithms for pass 1:

| <p> | Algorithm |
|-----|--|
| 0 | None (Pass 2 only) |
| 1 | Program n times with data = 0xFF (EEPROM erase) |
| 2 | Program each byte n times |
| 3 | Program each byte with one pulse that is $n * PPW^1$ long |
| 4 | Same as 3, but Vcc=5V |
| 5 | Program each byte until it matches, then program it an additional n times |
| 6 | Same as 5, but Vcc = 5V |
| 7 | Program each byte until it matches (P times), then program it an additional nP times |
| 8 | Program each byte until it matches, then program it with one additional pulse that is $n * PPW^1$ long |
| 9 | Program each byte until it matches (P times), then program it one additional time with a pulse that is $nP * PPW^1$ long |
| 10 | Unlock EEPROM (Pass 1 only) |
| 11 | SuperFlash erase |

1. PPW is the amount of time set by the combination of the PPW and PTU commands

EEnn>P2A <p> **Define Programming Pass 2**

Selects one of 10 possible programming algorithms for pass 2. (See table for FP1. The max allowed value for pass 2 is 9.)

EEnn>P1N <nn> **Define n for Programming Pass 1**

Defines the n parameter for programming pass 1, where n must be between 0 and 7. See PTU command for time units and FP1 command for an explanation of what n means.

EEnn>P2N <nn> Define n for Programming Pass 2

Defines the n parameter for programming pass 2, where n must be between 0 and 7. See PTU command for time units and FP2 command for an explanation of what n means.

EEnn>P64 <0/1> Write With 64-byte Pages

"P64 1" enables writing to the device with 64-byte pages. "P64 0" disables this function.

When this mode is enabled, writes to the device must include only whole 64-byte pages.

EEnn>OEV {0/1} Set Output Enable Pin to +12V during Programming

OEV 1 will cause the Output Enable pin to be driven to +12V during Programming. This is only allowed if the Output Enable signal is assigned to either ZIF socket pin 21 or pin 22 (pins 19 or 20 for a 24-pin device).

EEnn>PMX <mm> Define Max Value for P, or Number of Programming Passes

If any smart programming algorithm is selected that programs until each byte matches (P), then this command defines the maximum value for P. If the Simple Programming Algorithm is selected (using the SPA command), then this command defines the number of programming passes. $P < 256$.

EEnn>POL {0/1} Enable EEPROM-Type Completion Polling

POL 1 will cause polling after each byte written, reading back the data and waiting for it to match the written data. (This is how many EEPROMs signal the end of their write cycles.)

EEnn>PPW <nn> Define Programming Pulse Width

Defines the programming pulse width, in units defined by the PTU command. <nn> =0 means 250 nS pulse. <nn> must be less than 128.

EEnn>PSS <nn> Define Programming Strobe Separation

Defines the minimum time between programming strobe pulses, in units defined by the PTU command. <nn> =0 means as short as possible. <nn> must be less than 128.

EEnn>PTU {0/1} Define Programming Pulse Time Units

PTU 0 means that values given with the PSS and PPW commands are in units of 10 uS. PTU 1 means that these values are in units of 1 mS.

EEnn>PUL <p> Define Write Pulse Signal

Defines which EPROM signal is used as the write pulse:

| <p> | Signal |
|-----|---|
| 0 | High-voltage pulse on the Vpp signal, returning to 0V between pulses |
| 1 | High-voltage pulse on the Vpp signal, returning to +5V between pulses |
| 2 | Digital programming pulse on the Program (PGM) signal |

EEnn>SPA Specify Simple Programming Algorithm

This will overwrite any Smart programming algorithm specified by FP1 or FP2. The simple programming algorithm writes once per byte for the entire EPROM range, and then repeats this sequence the number of times specified by the PMX command.

EEnn>VCP <v> Define Vcc During Programming

Defines the voltage on the Vcc pin (pin 24 or 28) during programming:

| <v> | Vcc during Programming |
|-----|------------------------|
| 0 | 0V |
| 1 | 5V |
| 2 | 5.5V |
| 3 | 6.0V |
| 4 | 6.25V |
| 5 | 6.50V |
| 6 | 12V |

EEnn>VPP <v> Define Vpp During Programming

Defines the voltage on the Vpp pin during programming:

| <v> | Vpp if ZIF pin 20 | Vpp if ZIF pin 22 or 23 |
|-----|-------------------|-------------------------|
| 0 | Switcher off | Switcher off |
| 1 | 13.45V | 12.80V ±0.05V |
| 2 | 13.85V | 13.20V ±0.05V |
| 3 | 21.70V | 21.10V ±0.05V |
| 4 | 25.90V | 25.20V ±0.05V |

EEnn>VPR <v> Define Vpp During Reading

Defines the voltage on the Vpp pin during read operations:

| <v> | Vpp during Reads |
|-----|------------------|
| 0 | 0V |
| 1 | 5V |

Section 5. Algorithms

The ME27000 has a very flexible programming algorithm - or more accurately, selection of algorithms. All of these algorithms are based around the idea that programming a single byte involves the following steps:

1. Write the address to the EPROM address pins
2. If the EPROM Device Type has an address strobe pin, then strobe it to the inactive state (high), and then back to the active state (low)
3. Write the data to the EPROM data pins
4. Pulse the appropriate signal for the specified amount of time
5. Wait (if so required) for the specified amount of time between pulses, or (if so required) poll (EEPROM-style) for write completion

For each EPROM Device Type, the following constants are defined:

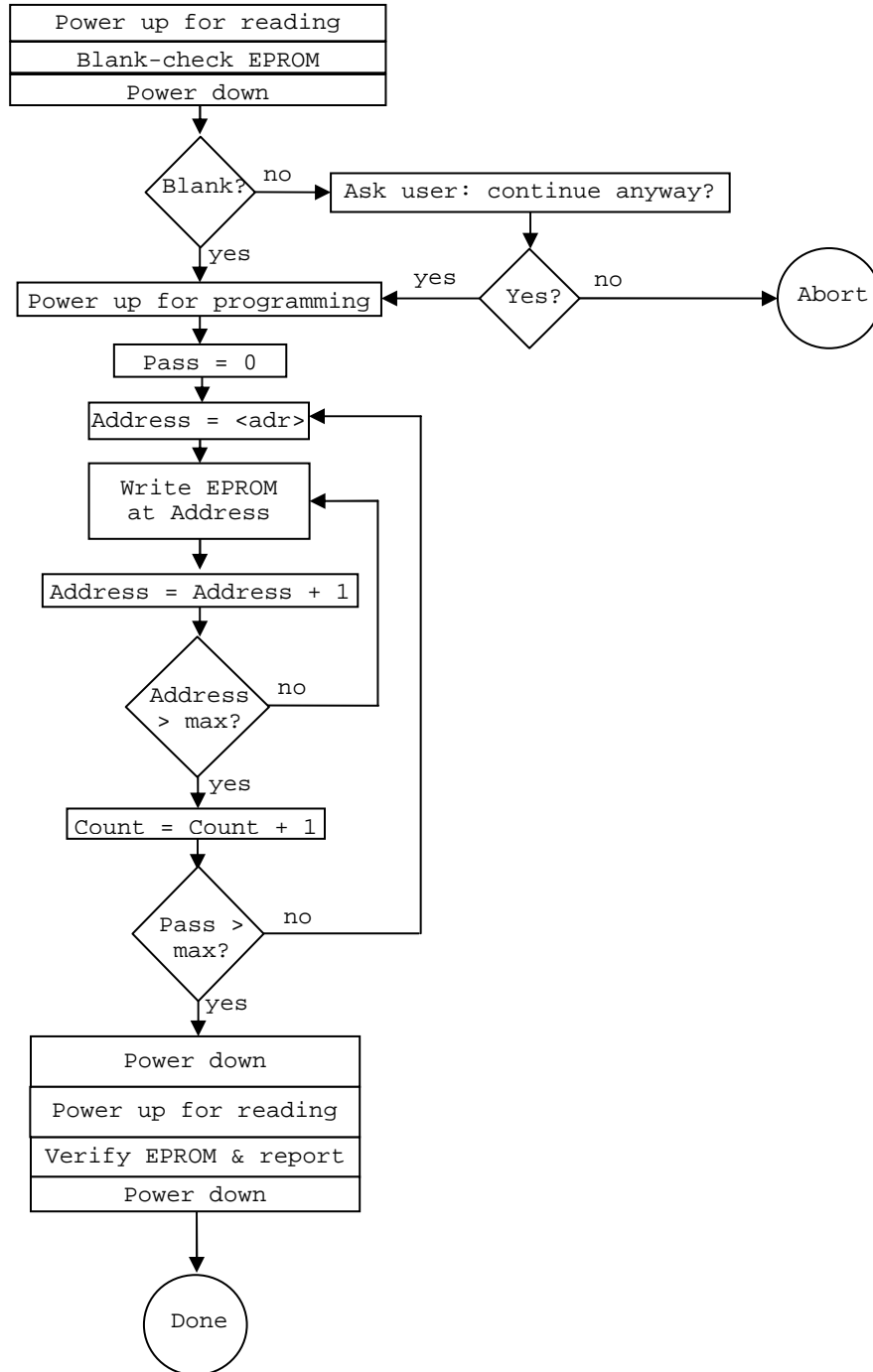
- The EPROM's pinout
- Any stuck-high pins
- The programming voltage, V_{pp}
- Special programming voltage requirement for V_{cc}
- Special programming voltage requirement Output Enable
- The programming pulse width
- On which signal the programming pulse is applied
- The minimum separation between programming pulses
- The programming algorithm (see below)
- Whether or not to blank-check the EPROM before programming
- Whether a device erases to 0x00 or 0xFF
- Whether or not to write 0xFF to each byte before programming it (as required for some EEPROMs)
- Whether or not to poll each byte for completion (as required for some EEPROMs)
- Whether or not the device requires EEPROM software unlocking prior to programming
- Whether or not to lock the EEPROM when done

During the entire programming cycle:

- The Output Enable pin is disabled, if it exists (except during verify)
- The Chip Enable pin is enabled, if it exists
- The V_{cc} pin is raised to an elevated voltage, if required
- The Output Enable pin is raised to an elevated voltage, if required
- The V_{pp} pin is driven to the specified voltage, unless it is pulsed per-byte

5.1 Simple Programming Algorithm

This algorithm sequentially writes every byte of the specified range of the EPROM, and then repeats the specified number of times.



5.2 Smart Programming Algorithms

The ME27000's Smart Programming is a flexible system, designed to allow implementation of all (or at least most) Smart/Fast/Quick/Express/Rapid/Turbo/Whatever programming algorithms specified by the various EPROM manufacturers. All of these Smart Programming algorithms have either one or two phases. Each phase may be defined independently, each comprising one pass through the specified address range of the EPROM.

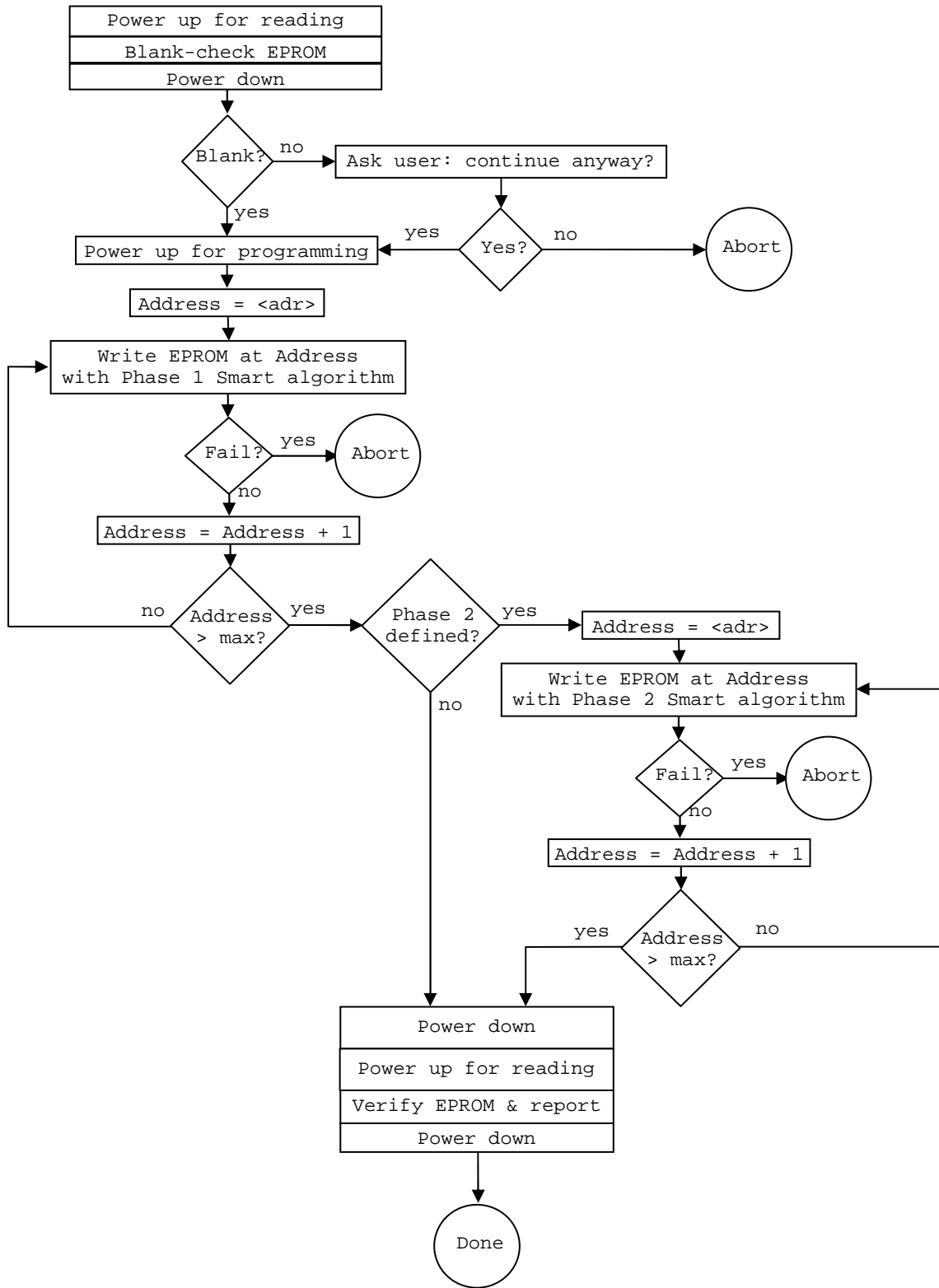
Each of the two Smart Programming phases is one of 6 possible types:

1. Program each byte until it matches, then program it an additional n times, where n is a value between 0 and 15
2. Program each byte until it matches, then program it with one additional pulse that is $n * \text{PPW}$ long, where n is a value between 0 and 15
3. Program each byte until it matches (P times), then program it an additional nP times, where n is a value between 0 and 15, and where the maximum P is specified (and is less than 256)
4. Program each byte until it matches (P times), then program it one additional time with a pulse that is $nP * \text{PPW}$ long, where n is a value between 0 and 15, and where the maximum P is specified (and is less than 256)
5. Program each byte n times, where n is a value between 0 and 255
6. Program each byte with one pulse that is $n * \text{PPW}$ long, where n is a value between 0 and 15

The first pass may also perform the following:

1. Write 0FFh to every byte, instead of writing buffer data. This is for erasing some EEPROMs, such as the Intel 2816A.
2. SuperFlash erase, which applies programming voltage to V_{pp} , +12V to ZIF socket pin 24, and then pulses the PGM pin for $n * \text{PPW}$, where n is between 0 and 15.
3. EEPROM unlock sequence. (See Section 5.3 for details.)

The following page shows the general Smart programming algorithm, where one of the above six byte-programming algorithms is used for each phase of the algorithm.

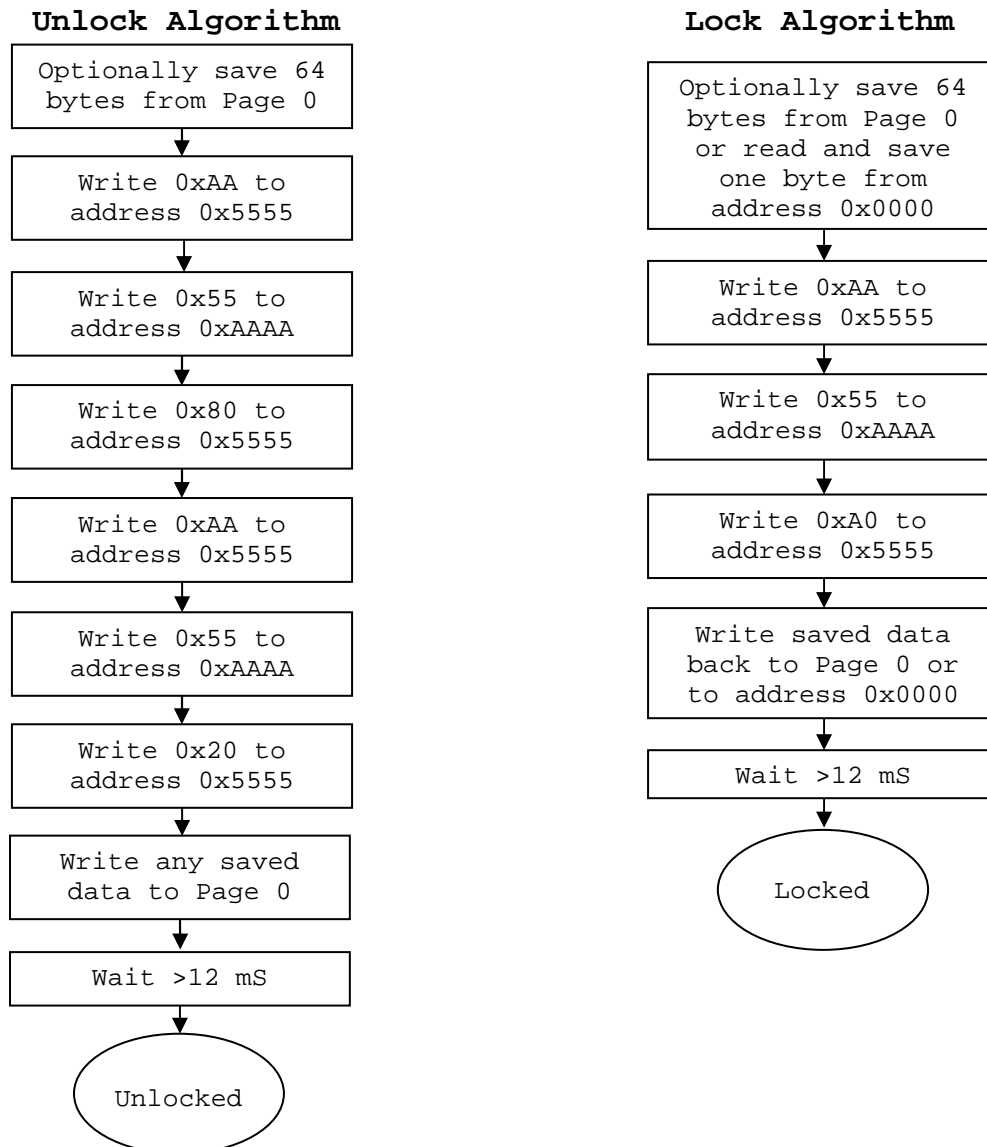


5.3 EEPROM Software Lock and Unlock Algorithms

EEPROMS that support software locking and unlocking all use similar algorithms. The ME27000's algorithms are designed to be compatible with all EEPROMS that support this feature. These algorithms require writing a specific sequence of data to a specific sequence of addresses, and require writing these sequences fast enough that the EEPROM recognizes the lock or unlock sequence, instead of actually writing to the EEPROM memory array. Typically, EEPROMS require that each byte of the sequence is written no more than 100 uS after the previous byte. The ME27000 requires less than 45 uS to write each byte.

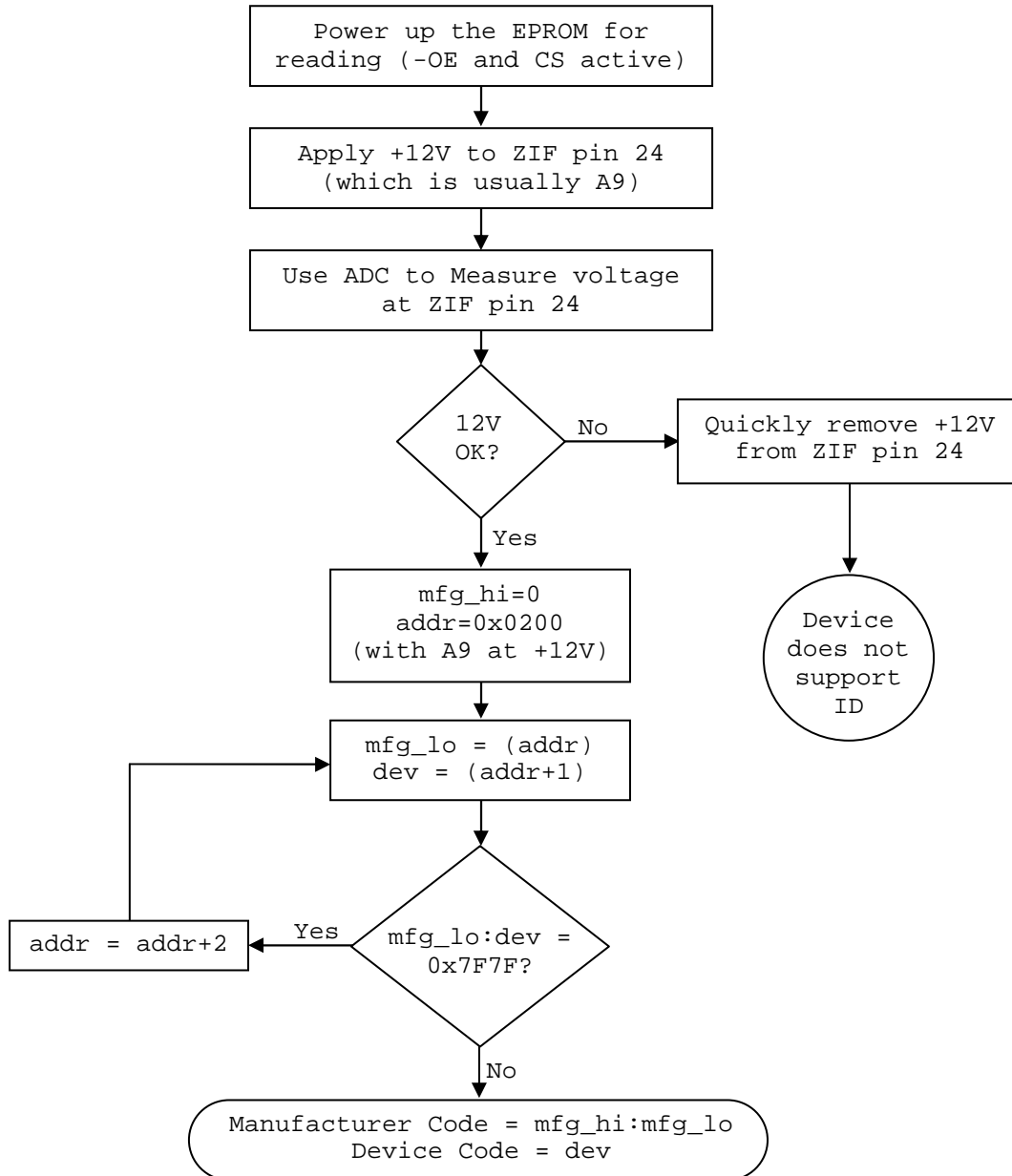
Some EEPROMS require an actual data-write or a 64-byte page write to complete the unlock or lock sequence. The ME27000 pre-read EEPROM data, so that it can write it back as needed at the end of the unlock or lock sequence.

Note that smaller EEPROMS don't have the upper address bits, so the addresses will be truncated as needed.



5.4 Device ID Algorithm

Later EPROMs include a mechanism for programmers to query the EPROM, to determine the manufacturer and the particular device form that manufacturer. This is the algorithm for reading the Device ID, comprising a 16-bit Manufacturer Code, and an 8-bit Device Code:



Early devices that do not support the Device ID function may be damaged by applying +12V to the signal on pin 24 of the ZIF socket. The ME27000 tries to minimize this possibility two ways: 1) current to this pin is limited to about 7 mA, and 2) the 12V signal is removed within about 38 μ s if the current exceeds about 3 mA (meaning the pin does not support application of +12V). Nonetheless, it would be better not to try to read the ID for a device that is known not to support this function.

EPROM specifications are inconsistent about how some of the high-address bits should be set (high or low) when reading the device ID. For this reason, the ME27000 goes through a sequence of trials to read the EPROM's ID. It will try valid address combinations until either a valid-looking ID is found, or all allowed possibilities have been tried.

Attempting to read the ID from a device that does not support the ID feature and that is not blank, may result in reading the EPROM's (non-blank) data. This data may look like a valid ID and fool the ME27000.

Some early EPROMs will give a Manufacturer Code = 0x0000 and a Device Code = 0x00. This means that the device allows the ID to be read without damage, but the actual ID is not supported.

The ME27000 firmware includes a fairly extensive list of Manufacturer and Device Codes, and in most cases can automatically set up for the correct Device Type based on the ID read, if the device divulges its ID. If the Device Code is unrecognized, the ME27000 can usually still recognize the manufacturer code, and will at least tell you who made the device. Note that sometimes the Manufacturer Code does not match the manufacturer that is printed on the package. This is usually because one manufacturer has bought the device masks from another manufacturer. (This is especially common with Microchip and ST.)

Section 6. ME27000 Theory of Operation

6.1 Architecture

The following is a very brief description of the ME27000 Programmer's circuit operation.

6.2 Microcontroller

At the heart of the ME27000 Programmer is a 40-pin PIC18F45K20 microcontroller, running at 16 MHz (meaning a 250 ns instruction cycle time), which includes the following functions:

- 32K of Flash program memory, which holds the ME27000 firmware. This Flash memory can be reprogrammed in place via the 6-pin PIC ICP connector and a Microchip PICkit III programming device.
- Internal SRAM, used for transmit and receive queues, variable storage, and general purpose registers.
- Internal EEPROM, used to store settings that will be retained when the power is off, as well as custom EPROM definitions.
- An internal UART that is connected to the serial port through a MAX232 RS232C transceiver. and to the USB port through an MCP2221A USB transceiver
- An external 64K-byte serial SRAM chip (which is used for the EPROM buffer), attached to the PIC's SPI bus
- An internal voltage reference, PWM circuit, and voltage comparator, combined together to become the Vpp voltage regulator.
- A multi-channel A/D converter, used to measure Vpp for foldback current limiting, and to check for overcurrent when reading the EPROM ID

The PIC's Flash memory has been programmed with two pieces of firmware: the Loader and the Programming Firmware.

6.3 Logic Supplies

V1 regulates the Wall adapter's +12V down to about +5.4V, used for some of the logic, as well as for powering the EPROM. (This voltage is a little high because of the voltage drop across protection diodes and switching transistors in the circuit that supplies power to the EPROM.) It is normal for this regulator to get warm during operation, particularly during programming. (This is the only component that should ever get warm.)

V2 regulates the +5.4V supply down to 3.3V for the PIC microcontroller, the serial SRAM, and the data buffer that sits between the microcontroller and the EPROM.

6.4 +6.35V Supply

This power supply is used to provide special Vcc voltage for some EPROMs during programming. The voltage drop across the protection diode and the transistor switch reduce this voltage by 0.35V by the time it reaches one of the Vcc pins of the EPROM.

The ME27000 can boost this voltage by 0.37V, so that the ME27000 can provide the following Vcc voltages to the EPROM: 5.00V, 6.00V, 6.37V, and 12V. The 6.37V option is within tolerance for EPROMs that require either 6.25V±5% or 6.50V±5%.

This supply is adjustable because a few EPROMs need 5.5V. When such EPROMs are to be programmed, the user can adjust this supply (with VR5) as needed.

Note: when measuring at TP7, adjust VR5 to be 0.25V higher than the target EPROM Vcc.

6.5 Negative 5V Supply

U11, together with C5 and C6, comprise a switched-capacitor voltage inverter that produces -5.4V from the +5.4V supply.

6.6 Microcontroller-Controlled High-Voltage Supply

The various Vpp voltages are produced by a switching power supply that is directly controlled by the PIC microcontroller. The PIC's internal PWM is set up to produce a 108 KHz pulse-train to the base of Q1, which in turn kicks current through inductor L1, to boost voltage at C3, using diode D2.

The PIC's internal voltage comparator compares the output from one of the four voltage dividers (R5-R12 and VR1-VR4) to its internal voltage reference. When the measured voltage is too high, the PIC's internal PWM circuit stops producing pulses to Q1. Once the measured voltage falls below the reference, the PIC's PWM again generates pulses. In this way, Vpp is produced and regulated by the PIC.

Firmware selects which of the four voltage dividers is used for this feedback, depending on what Vpp voltage is required. The resistor values of each divider have been chosen to produce the various Vpp values needed for the various EPROMs supported by the ME27000.

Resistor R3, transistor Q2, and diodes D3 and D4 serve as an approximately 100 mA current limiter for Vpp. Q2 is normally saturated, and the voltage drop across this circuit is 1.4V. If Vpp current exceeds about 100 mA, Q2 will come out of saturation, and the voltage drop across it will rise, causing Vpp to be reduced proportionally to Vpp current. While generating Vpp, the firmware connects another one of the voltage dividers to its A/D converter, to verify that the produced voltage is within ±15% of the intended voltage, and shuts the power supply (with an error message) down if not. Thus, when Vpp current is very much in excess of about 100 mA (which will occur only when something is wrong with the EPROM being programmed), the firmware performs a foldback current limiting function for Vpp.

Vpp can be directed to any one of the following ZIF socket pins: 1, 20, 22, 23. If Vpp is directed to ZIF pin 20, then it will be one diode drop (about 0.7V) higher than on the other pins. This is to support EPROMs like the 2708, which need about 26V for Vpp, compared to about 25V for other EPROM Device Types.

6.7 EPROM Digital Pin Interface

All signals to the EPROM are buffered.

The EPROM data pins are buffered with a 74LVC245 (the External Data Buffer). The EPROM address pins 0 through 7 are buffered with a 74LS374 8-bit latch (the Low Address Latch). The remaining EPROM address and control pins are buffered by logic that allow these signals to be assigned to one of several pins, and to be set to more than just digital signal voltages. Many of these signals are controlled by another 74LS374 8-bit latch, called Port F. The remaining digital signals are controlled by the PIC's internal ports.

The the External Data Buffer, Low Address Latch, and Port F are all driven by the same 8 data Port D pins of the PIC microcontroller. The latch signal for the Low Address Latch is also the signal that drives the BUSY light. The latch signal of Port F is also the signal that drives the ERROR light.

Section 7. Downloading Firmware via the Serial Port

You can load new ME27000 firmware via the serial port. Most likely, you will load a firmware update that I have emailed to you. However, if you have the programming skills and I am not providing some firmware feature that you need, then you can create your own firmware (probably by modifying mine), and download that to your ME27000.

The ME27000 firmware is divided into two components: the ME27000 Loader (which cannot be downloaded via the serial port) and the ME27000 Programming Firmware (the Programming Firmware). The primary function of the Loader is to load new Programming Firmware via the serial port, eliminating the need to use a PIC programming device, such as the PICkit 3.

This section describes how to load new Programming Firmware.

7.1 Firmware Download Instructions

Connect your ME27000 to the serial port of your computer (or a serial port dongle on the USB port of your computer) and start a terminal program, such as Hyperterm. Set up the terminal program this way: 9600 baud, 8 data bits, no parity, XOFF/XON handshaking enabled.

Install a shunt between pins 1 and 2 of J4, the ICP connector, and then power the programmer on. You will see the Loader message, instead of the ME27000 sign-on banner:

```
ME27000 Loader 1.0
```

When you see this message, the ME27000 is ready to receive a Programming Firmware file in Intel Hex format. The Loader expects to see an Intel Hex file exactly like that produced by Microchip's MPASM assembler.

Send the hex file (typically, ME27000.hex) to the ME27000. It will take a few minutes to download, and you should see the hex file as it downloads. If there are any errors, they will be flagged with a brief error message:

| Message | Meaning |
|---------|--|
| ?Csm | Checksum error in Intel Hex record |
| ?Hex | Illegal (non-hex) character received (Only '0'-'9' and 'A'-'F' are allowed) |

The Loader will first read the entire Intel Hex file into its RAM (and so does not require any serial port handshaking).

When the firmware load is complete, the Loader will print the total number of Intel Hex records loaded, as well as the number of errors detected. If the error count is anything but 0000, the firmware load failed, and the downloaded firmware is most likely corrupted. (The original programmer firmware is still intact and can vbe run by removing the shunt on J4 and power-cycling the programmer.)

If no errors were detected during the Hex load, the Loader will then write the file to its program flash memory, and the read it back to verify the write. Verify errors are reported to the console.

If the load is successful, you can jump to the new Programming Firmware by typing the ESC key several times. Or you can power-cycle the ME27000.

If the load fails for any reason, you can try again immediately after the failed load, when Loader message is printed. Or, you can power-cycle the ME27000 (with the shunt in place in J4) to invoke the Loader again.

7.2 Intel Hex File Format for Firmware Downloads

This section specifies the format of Intel Hex files that are accepted by the Loader, as well as the error messages that are printed by the Loader. This Intel hex format is exactly the format produced by Microchip's MPASM assembler. Note that this specification is slightly different than Intel Hex files accepted by the Programming Firmware.

An Intel Hex record is defined as follows:

```
:NNAAaaTTDDDDDD..DDDDCC
```

- A colon marks the beginning of an Intel Hex record. All characters are ignored until a colon has been received. This means that comment lines in the Intel Hex file (that contain no colons) will be ignored. This also means that any record where the initial colon has been corrupted will be ignored without being caught as an error.
- NN defines the number of Data bytes in the record.
- AAaa is the address field of the record. AA is the most significant address byte; aa is the least significant address byte.
- TT is the record type.
- DD is a data byte. Data bytes belong in memory at sequential addresses, starting at AAaa. The record should have NN data bytes.
- CC is the checksum of the record. The low byte of the sum of NN, AA, aa, TT, all the DDs, and CC should be 00.
- A carriage return (CR), a line feed (LF), or both, is optional.

Three Intel Hex record types are accepted; all other records are ignored.

- 1) **Type 00 records** are data records. Data records are written to FLASH only if a Type 04 record has already been received, with extended address = 0000. If no Type 04 record has been received yet, or if the last Type 04 record set the extended address to something other than 0000, then the data record will be ignored. This means (for example) that an Intel Hex file cannot write to the Config registers of the PIC.
- 2) **A Type 01 record** (with 0 bytes of data) is an End-Of-File record, and is required at the end of the file to force a write to FLASH of the last RAM buffer full of data. NOTE: a data record (Type 00) with 0 bytes of data is NOT treated as an End-of-file record.
- 3) **Type 04 records** set the extended address for the subsequent records. The "address" field of the Type 04 record (bytes 2 and 3) is ignored. The first 2 bytes of the "data" field (bytes 5 and 6) set the extended address. MPASM sets the extended address to 0000 for FLASH data, and to other values for the PIC Config registers, EEPROM, etc.

The hex file must end with a type 01 (End-Of-File) record.

Section 8. ME27000 Programmer Assembly

Assembly requires basic electronics skills, a decent soldering iron and solder, needle-nosed pliers, diagonal cutters, wire strippers, and a couple of screwdrivers.

Take your time to install all components in their correct locations, with the correct orientation. Install all components flush to the PC board, and with good, clean soldering. Inspect your work when you are done.

The silkscreen on the PC board is verbose, mainly to help you assemble it correctly and to aide in debugging. But the silkscreen may not be perfect. When in doubt, refer to these assembly instructions.

Be careful with diode type and orientation: the diode's stripe must align with the stripe on the silkscreen. The silkscreen has an abbreviation of the diode number, to aide in putting the correct diode in each location.

Also pay attention to the orientation of the electrolytic capacitors. Reversing these capacitors can cause some excitement.

There is logic to the order of assembly: the lower-profile components first, the higher ones later. Also, unusual components are installed first, so that bulk components will not be installed in the wrong places.

All unlabeled transistors are 2N3906. Most unlabeled resistors are 1K, 5%. All unlabeled diodes are 1N5917.

When installing IC sockets, check their orientation, and make sure they seat completely against the PC board with no pins bent under. I suggest soldering them in place with just two diagonally-opposite pins, then re-heating these solder connections while gently pressing the socket to the board, to get them nice and tight. Solder the rest of the pins once the socket is flush to the PC board.

This manual has check boxes next to every step, so you can check off each step when it is complete. Some of the check boxes are at the left margin; others are the left column of tables.

8.1 Printed Circuit Board Assembly

Follow these steps to assemble the PC board. The order of component installation has been chosen to ease assembly and to minimize mistakes. Low-profile components are inserted first, so that the PC board will lie flat on your workbench during soldering, and less-common components are installed before more-common components. Most component values are printed (or abbreviated) on the PC board silkscreen.

Step 1. Install the following diodes. **Be very careful** about orientation and also **be very careful** to put the correct diode in each location. It's a good idea to bend the leads such that the last 2 or 3 digits of the diode number will be readable when the diodes are soldered in place.

| √ | Qty | Locations | Component | Digikey Part Number |
|---|-----|----------------|-----------|---------------------|
| | 1 | D19 | BZX79C4V7 | BZX79C4V7-T50ACT-ND |
| | 10 | D6-D13,D15,D16 | BAT46 | BAT46CT-ND |

Step 2. Install the following 1/4 W, 1% resistors.

| √ | Qty | Locations | Value | Digikey Part Number |
|---|-----|------------|------------------|--------------------------|
| | 1 | R49 | 93.1 Ω 1% | 93.1XBK-ND |
| | 3 | R2,R22,R50 | 274 Ω 1% | 274XBK-ND |
| | 1 | R52 | 866 Ω 1% | 866XBK-ND |
| | 1 | R1 | 910 Ω 1% | 13-MFR-25FTE52-910RCT-ND |

Step 3. Install the following 1/4 W and 1/2 W, 5% resistors. Note that the silkscreen values for R7, R9, and R11 are incorrect.

| √ | Qty | Locations | Value | Digikey Part Number |
|---|-----|---------------------------------|---------------------|---------------------|
| | 1 | R3 | 6.8 Ω | 6.8QBK-ND |
| | 5 | R29,R35,R37,R43,R61 | 100 Ω | 100QBK-ND |
| | 6 | R23,R30,R40,R48,R57,R59 | 330 Ω | 330QBK-ND |
| | 4 | R24,R27,R31,R58 | 470 Ω | 470QBK-ND |
| | 7 | R17,R38,R41,R44,R45,R51,R53 | 680 Ω | 680QBK-ND |
| | 3 | R5,R7,R11 | 2 K Ω | 2.0KQBK-ND |
| | 5 | R25,R26,R28,R32,R62 | 2.2 K Ω 1/2W | S2.2KHTR-ND |
| | 1 | R9 | 2.4 K Ω | 2.4KQBK-ND |
| | 3 | R16,R18,R34 | 4.7 K Ω | 4.7KQBK-ND |
| | 2 | R15,R20 | 10 K Ω | 2019-CFS1/4C103J-ND |
| | 6 | R6,R8,R19,R21,R39,R47 | 22 K Ω | MFR-25FBF52-22K-ND |
| | 2 | R10,R12 | 43 K Ω | MFR-25FBF52-43K-ND |
| | 9 | R13,R14,R33,R36,R46,R54-R56,R60 | 1 K Ω | 1.00KXBK-ND |

Step 4. Install the following 1 W, 5% (small form-factor) resistors.

| √ | Qty | Locations | Value | Digikey Part No. |
|---|-----|-----------|--------------------------------|--------------------|
| | 1 | R4 | 1 K Ω | 738-RSMF1JB1K00-ND |
| | 1 | R42 | 2 K Ω or 2.2 K Ω | 2.0KW-1-ND |

Step 5. Install the following diodes. **Be very careful** about orientation

| √ | Qty | Locations | Component | Digikey Part Number |
|---|-----|-------------|-----------|---------------------|
| | 3 | D1,D17, D18 | 1N5817 | 1N5817GOS-ND |
| | 4 | D3-D5,D14 | 1N4004 | 1N4004-TPMSCT-ND |

Step 6. Install all of the DIP sockets flush to the PC board, paying attention to their orientation:

| √ | Qty | Locations | Component | Digikey Part Number |
|---|-----|-----------------|------------|---------------------|
| | 2 | U7,U11 | 8-pin DIP | 2057-ICS-308-T-ND |
| | 5 | U3,U8-U10,U16 | 14-pin DIP | ED3045-5-ND |
| | 5 | U1,U2,U5,U6,U15 | 16-pin DIP | 2057-ICS-316-T-ND |
| | 3 | U12-U14 | 20-pin DIP | 2057-ICS-320-T-ND |
| | 1 | U4 | 40-pin DIP | ED3048-5-ND |

Step 7. Install the resistor pack, being careful to orient it with pin 1 toward the top of the PC board:

| √ | Qty | Locations | Component | Digikey Part Number |
|---|-----|-----------|-----------------------------------|---------------------|
| | 1 | RP1 | 10K, 9-pin, 8-resistor SIP R-pack | 4609x-101-103LF-ND |

Step 8. Install the large power diode in D2. **Be very careful** about orientation. The leads need to be bent close to the diode body.

| √ | Qty | Locations | Component | Digikey Part Number |
|---|-----|-----------|---------------|---------------------|
| | 1 | D2 | Diode, 1N5821 | 1N5821RLGOSCT-ND |

Step 9. Install the following ceramic capacitors.

| √ | Qty | Locations | Value | Digikey Part Number |
|---|-----|----------------------------------|--------------------|---------------------|
| | 3 | C27-C29 | 0.047 μ F, 50V | 445-173599-1-ND |
| | 1 | C17 | 0.47 μ F 25V | 445-181293-ND |
| | 7 | C9-C12,C14-C16 | 1 μ F, 16V | 445-173583-1-ND |
| | 3 | C5,C6,C20 | 10 μ F, 10V | 445-181283-ND |
| | 12 | C4,C7,C8,C13,C18,C19, C21-C26 | 0.1 μ F, 100V | 399-4329-ND |

Step 10. Install the following TO-92 devices. Be very sure you put the right component in each location. Double-check their orientation. When installed, these components should stand straight, and have about 3/16 inch of lead between the PC board and their plastic bodies.

| √ | Qty | Locations | Component | Digikey Part Number |
|---|-----|-----------------------|------------------|---------------------|
| | 1 | V2 | MCP1700-3302E/TO | MCP1700-3302E/TO-ND |
| | 1 | V3 | LM317L | 497-1573-5-ND |
| | 1 | Q9 | 2N3904 | 2N3904FS-ND |
| | 1 | Q13 | 2N6520 | 2N6520TACT-ND |
| | 2 | Q3,Q22 | 2N7000 | 2N7000FS-ND |
| | 16 | Q4-Q8, Q10-Q12,Q14-21 | 2N3906 | 641-1946-ND |

Step 11. Install 5 trim-pots in the following locations, and set them to approximately the center of their ranges. Push them firmly into the holes in the PCB before soldering - they will snap into place.

| √ | Qty | Locations | Component | Digikey Part Number |
|---|-----|-----------|-----------------|---------------------|
| | 5 | VR1-VR5 | 500 ohm Trimpot | 3306P-501-ND |

Step 12. Install LEDs in the following locations, paying attention to their orientation. (The LEDs have a flat side that should match the flat side shown on the silkscreen LED outline and the shorter lead goes in the square pad, toward the bottom edge of the board.)

| √ | Qty | Locations | Component | Digikey Part Number |
|---|-----|-----------|-----------|----------------------|
| | 1 | LED1 | Blue LED | 67-1751-ND |
| | 1 | LED2 | White LED | 1516-QBL8IW60D-NW-ND |
| | 1 | LED3 | Amber LED | 754-1872-ND |
| | 1 | LED4 | Red LED | 1516-1359-ND |

Step 13. Install the four electrolytic capacitors. Be sure to install them with the correct orientation. The negative sign on each capacitor should be farthest from the + sign on the PC board. Note that C3 is incorrectly labeled "470 uF" on Rev C PC boards.

| √ | Qty | Locations | Value | Digikey Part Number |
|---|-----|-----------|---------------------------|------------------------|
| | 1 | C3 | 100 μ F, 50V, low ESR | 399-ESX107M050AH2AA-ND |
| | 2 | C1-C2 | 470 μ F, 35V, low ESR | 399-6086-ND |

Step 14. Install the inductor snugly against the PC board. Make sure the wires are pulled tight through their holes before soldering.

| √ | Qty | Locations | Value | Digikey Part Number |
|---|-----|-----------|-------------|---------------------|
| | 1 | L1 | 100 μ H | 732-1424-ND |

Step 15. Install the following power transistors, standing straight up from the board. Note that although the orientations are not the same, both transistors should be installed with their writing facing toward the left side of the PV board.

| √ | Qty | Location | Component | Digikey Part Number |
|---|-----|----------|-----------|---------------------|
| | 1 | Q1 | KSC2690 | KSC2690AYSFS-ND |
| | 1 | Q2 | 2SA2222SG | 2SA2222SGOS-ND |

Step 16. Install connectors in the following locations. Be sure they are installed completely flush to the board. Also be sure that the larger holes are completely filled with solder.

| √ | Qty | Locations | Component | Digikey Part Number |
|---|-----|-----------|-----------------------|----------------------------|
| | 1 | J1 | Barrel, 5.5mm x 2.0mm | CP-063AH-ND |
| | 1 | J2 | DC9F | AE10921-ND |
| | 1 | J3 | USB-B | 2057-USB-B-S-RA-WT-SPCC-ND |
| | 1 | J4 | Header, 6-pin | A31116-ND |

Step 17. Screw the TO-220 voltage regulator to its heatsink, and then solder this subassembly onto the PC board.

| √ | Qty | Location | Component | Digikey Part Number |
|---|-----|----------|-------------------|---------------------|
| | 1 | V1 | LM317 | 497-1575-5-ND |
| | 1 | V1 | Heat sink | 345-1023-ND |
| | 1 | V1 | 6-32 x 3/8" screw | H356-ND |
| | 1 | V1 | 6-32 nut | H220-ND |

Step 18. Install the ZIF socket. Install the socket with its handle toward the top edge of the PC board - the handle should be closest to the marked pin-1 pad. It is **very important** to open the socket (handle perpendicular to the PC board) before you solder it in place. Failure to open the socket before soldering will cause the socket to open incorrectly during use.

| √ | Qty | Locations | Component | Digikey Part Number |
|---|-----|-----------|-------------------|---------------------|
| | 1 | ZIF1 | 28-pin ZIF socket | |

Step 19. Install the power switch. Make sure it is flush to the board while soldering and that all holes are completely filled with solder.

| √ | Qty | Locations | Component | Digikey Part Number |
|---|-----|-----------|-------------|---------------------|
| | 1 | SW1 | SPDT switch | EG2365-ND |

Step 20. Install a 4-40 x 5/8" screw and a 4-40 nut as a support leg in each of the four corner holes. Install one more screw and nut in the hole near the lower left corner of the ZIF socket. (This 5th screw is to prevent board flexing when using the ZIF socket.)

| √ | Qty | Locations | Component | Digikey Part Number |
|---|-----|------------------------|-------------------|---------------------|
| | 5 | PCB corners & near Q13 | 4-40 x 5/8" screw | H348-ND |
| | 5 | PCB corners & near Q13 | 4-40 nut | 36-9600-ND |

Step 21. If you will use a MAX660 in location U11, then solder a jumper wire from pin 1 of u11 to pin 8 of U11. (Do not install this jumper if you are using an ADM660.)

Step 22. Inspect your work! Check for shorts, inadequate solder, component orientation, etc. This is a high-current circuit, and construction mistakes will probably damage components.

Note that the ICs are not yet installed on the PCBA. This will be done after some power supply checkout.

Section 9. Checkout and Adjustment

Basic checkout requires a voltmeter and either a computer terminal (such as the most excellent Wyse WY-30¹) or a PC with a serial port and a terminal emulation program. These tests are sequential - if you find a defect, do not move on until the defect has been corrected!

CAUTION: There are high voltages present on this board - not high enough to hurt you (unless you really try), but definitely high enough to damage components if you accidentally short a high-voltage signal to a digital signal. Be especially careful when probing the two 7407's, as these chips have both high-voltage signals on their pins, as well as digital signals from other chips, including from the PIC. One false move with your meter or scope probe and you will blow the output driver on some other chip. Voice of experience here...

9.1 Basic PCBA Checkout

These measurements are mainly made on the labeled test points along the top edge of the PC Board.

Step 1. At this point, no ICs should be installed. Turn the power switch off (toggle toward the board edge), and plug the AC Adapter into J1. Hook the ground lead of your voltmeter to the GND test point (TP1). Turn the power on. The white "POWER" LED should light.

Step 2. Measure the following voltages to confirm power supply operation:

| √ | Measure | Measurement | Meaning |
|---|---------|---------------------------------|---|
| | TP2 | 12.0V to 12.3V | Correct operation |
| | | >12.3V | Incorrect wall adapter? |
| | | <12V | Incorrect wall adapter? PC board short? Wrong component somewhere? |
| | TP4 | 5.38V to 5.49V | Correct operation |
| | | otherwise | R1 and R2 correct? PC board short? |
| | TP3 | 3.25V to 3.35V | Correct operation |
| | | otherwise | Correct component in V2? PC board short? |
| | TP7 | Adjust VR5 for 6.25V ± 0.03V | Correct operation |
| | | Can't adjust | Incorrect R46 or R47? PC board short? |
| | TP6 | 11.1V to 11.7V | Correct operation |
| | | Otherwise | Check switcher circuit: L1,D1,R3,Q1,D4, etc. |

Step 3. Turn the ME27000 off and install the ADM660 IC in U11. Turn the ME27000 back on, and measure the -5V supply.

| √ | Measure | Measurement | Meaning |
|---|---------|--------------|---|
| | TP5 | -5V to -5.3V | Correct operation |
| | | Otherwise | Incorrect C6 or C8? U11 installed correctly? PC board short? |

¹ The Wyse Technology WY-30 was the first product that I designed professionally.

Step 4. Turn off the ME27000 Programmer. Install ICs in the following locations, paying attention to orientation. Be careful not to bend any leads as you insert the ICs.

| √ | Qty | Locations | Component | Digikey Part # |
|---|-----|-------------|---|--|
| | 1 | U1 | MAX232 RS232 Transceiver | 296-26139-5-ND |
| | 1 | U3 | MCP2221A USB Transceiver | MCP2221A-I/P-ND |
| | 1 | U10 | 74ACT04 Hex inverter, CMOS outputs | 296-4351-5-ND |
| | 3 | U8, U9, U16 | 74LS07 Hex open collector driver | 296-14878-5-ND |
| | 3 | U5, U6, U15 | 74LS139 Dual 2:4 decoder, CMOS outputs | 296-1640-5-ND |
| | 1 | U2 | 74LS157 Quad 2:1 multiplexor | 296-1645-5-ND |
| | 2 | U12, U14 | 74LS374 8-bit flipflop, CMOS outputs | 296-1662-5-ND |
| | 1 | U13 | 74LVC245A 8-bit bi-directional driver, 3.3V outputs, 5V tolerant inputs | 296-8503-5-ND |
| | 1 | U7 | 23A512 65K-byte serial SRAM | 23LC512-I/P-ND |
| | 1 | U4 | PIC Microcontroller, pre-programmed with ME27000 firmware | PIC18F45K20-I/P-ND + ME27000 Firmware |

U4 is a PIC microcontroller with internal flash memory. You must use a PIC that has been pre-programmed with the ME27000 Loader Kernel 1.0, or program it in place yourself, using a PC, a Microchip PICkit-3 programming device, and my program file. (J4 is the PICkit-3 compatible in-circuit programming connector for this purpose.) The PIC must also be loaded with the ME27000 Programming Firmware, which can be loaded via the serial port - see section 7. If you are using the PIC that I supplied, then it has already been programmed with both the loader and the programming firmware.

Step 5. Turn the ME27000 back on, and re-check the voltages from steps 2 and 3 above.

9.2 Microcontroller Bring-Up

Step 2. Plug a terminal (or a PC with a terminal program) into the ME27000 Programmer's serial port connector, making sure to connect the transmit signal (TxD, pin 2) from the ME27000 to receive signal of the terminal and the receive signal (RxD, pin 3) from the ME27000 to the transmit signal of the terminal. (No hardware handshaking signals are required.) For a normal PC, you will need a straight-through DA-9S to DA-9P.

Step 3. Set up the terminal (or terminal program) this way:

| | |
|-----------|----------|
| Baud Rate | 9600 |
| Stop Bits | 1 |
| Parity | None |
| Handshake | XON/XOFF |

Step 4. Plug in the ME27000 Programmer and turn it on. On the terminal screen, you should see a sign-on banner and a prompt like this:

```

*=====*
*           ME27000           *
*=====*
* Orphan EPROM Programmer II *
*   By Martin Eberhard       *
*   Firmware Version 1.00    *
*=====*

Serial Number: C001
Current Device Type is 00: 2704
EPROM data invert: off
Type ? for command list
>

```

If you do not see this banner, check the following:

| √ | Check |
|---|--|
| | Is the terminal setup right? - baud rate, etc. as above |
| | If you are using a PC (maybe with an RS-232C - to - USB dongle), check that this is all working correctly. You can roughly test it with a loop-back from pin 2 to pin 3. |
| | TxD, RxD and GND wiring from the ME27000 Programmer to the terminal. Are TxD and RxD reversed? |
| | Are IC1 and IC2 inserted correctly? |
| | Is IC2 in fact programmed? (With the right code?) |

Step 5. Just the white "POWER" LED should now be lit. Debug if not.

| √ | LED | State | Meaning |
|---|---------------|-------|--|
| | LED2 Power | On | Correct |
| | | Off | PC Board short? IC inserted backwards? |
| | LED3 Busy | Off | Correct |
| | | On | Short on PC board? |
| | LED4 Error | Off | Correct |
| | | On | Short on PC board? |

Step 6. Type '?' to see a full help screen. You will try out all of the commands on this screen in the following sections.

9.3 Microcontroller-Assisted Checkout and Adjustment

NOTE: The following steps involve dialog with the ME27000's monitor. The monitor's prompt is '>'. You should type what is in **bold**, and the monitor will respond as indicated. If you turn off the power between steps, you may need to repeat the dialog up to the point where you are working, when you power back on. Most settings (such as the selected EPROM Device Type) are stored in EEPROM, and will be retained when the power is off.

All voltages are referenced to ground - reconnect the voltmeter ground lead to the PCBA GND pin (TP1). If the terminal and/or ME27000 Programmer are off, then turn them back on.

Step 1. Test and Adjust Vpp Supply

Connect the positive lead of your voltmeter to TP6 for the following tests.

```
>AVPP 1
Vpp set for 12.80V (Measure at TP6)
Note: Vpp will be about 0.7V higher on ZIF pin 20
>
```

Now, both LEDs should be lit:

| √ | LED | State | Meaning |
|---|---------------|-------|----------------------------------|
| | LED2 Power | On | Correct |
| | | Off | LED1 Orientation? |
| | LED3 Busy | On | Correct |
| | | Off | LED orientation? PC board short? |
| | LED4 Error | On | Short circuit in Vpp circuit? |
| | | Off | Correct |

Measure the voltage at TP6

| √ | Measurement | Meaning | Action |
|---|-------------|-------------------|---|
| | 11V to 16V | Correct operation | Adjust VR1 for 12.80V ± 0.05V |
| | 0V to 10.9V | Problem | <ul style="list-style-type: none"> Correct component in R5 and R6? Problem with Vpp Switcher circuit? |
| | >16V | Problem | <ul style="list-style-type: none"> Correct component in R5 and R6? |

```
>AVPP 2
Vpp set for 13.2V (Measure at TP6)
Note: Vpp will be about 0.7V higher on pin 20 of the ZIF socket
>
```

Measure the voltage at TP6

| √ | Measurement | Meaning | Action |
|---|-------------|-------------------|---|
| | 11V to 17V | Correct operation | Adjust VR2 for 13.20V ± 0.05V |
| | 0V to 10.9V | Problem | <ul style="list-style-type: none"> Correct component in R7 and R8? Problem with Vpp Switcher circuit? |
| | >17V | Problem | <ul style="list-style-type: none"> Correct component in R7 and R8? |



>AVPP 3

Vpp set for 21.1V (Measure at TP6)

Note: Vpp will be about 0.7V higher on pin 20 of the ZIF socket

>

Measure the voltage at TP6

| √ | Measurement | Meaning | Action |
|---|-------------|-------------------|--|
| | 16V to 25V | Correct operation | Adjust VR3 for 21.10V ± 0.05V |
|  | 0V to 15.9V | Problem | <ul style="list-style-type: none"> • Correct component in R9 and R10? • Problem with Vpp Switcher circuit? |
|  | >25V | Problem | <ul style="list-style-type: none"> • Correct component in R9 and R10? |



>AVPP 4

Vpp set for 25.2V (Measure at TP6)

Note: Vpp will be about 0.7V higher on pin 20 of the ZIF socket

>

Measure the voltage at TP6

| √ | Measurement | Meaning | Action |
|---|-------------|-------------------|---|
| | 19V to 32V | Correct operation | Adjust VR4 for 25.20V ± 0.05V |
|  | 0V to 18.9V | Problem | <ul style="list-style-type: none"> • Correct component in R11 and R12? • Problem with Vpp Switcher circuit? |
|  | >32V | Problem | <ul style="list-style-type: none"> • Correct component in R11 and R12? |

>TOFF {Turns off the Vpp supply and the Busy light.}

 Step 2. Test Data Outputs

>WD 55

>

The BUSY light should now be on. Use a voltmeter to measure the voltages on the ZIF socket data pins (11-13 and 15-19) to see 55h there. Logic low should be less than 0.2V, and logic high should be more than 3V. Try it again with the opposite polarities:

>WD AA

>

Resolve any problems with the data driver before continuing.

 Step 3. Test the Address Drivers

First, select a 64K EPROM, which will have 16 address lines:

>ET 4E

Current Device Type is 4E: 27512

>

Now write a pattern to the address lines:

>WA AAAA

>

The ETD command will show you a picture of the EPROM:

Type 4E: 27512, size: 65536 x 8

```

-----v-----
A15 -| 1      28  |- Vcc      Programming Vcc = 6.0V
A12 -| 2      27  |- A14
A7  -| 3      26  |- A13
A6  -| 4      25  |- A8
A5  -| 5      24  |- A9
A4  -| 6      23  |- A11
A3  -| 7      22  |- -OE/Vpp 13.15V
A2  -| 8      21  |- A10
A1  -| 9      20  |- -CS/-PGM  Supported Devices:
A0  -| 10     19  |- D7      AMD      AM27512
D0  -| 11     18  |- D6
D1  -| 12     17  |- D5
D2  -| 13     16  |- D4
GND -| 14     15  |- D3
-----

```

Vpp during read: 5.0V

Programming pulse on PGM pin

Programming pulse width: 1000 uS

Programming algorithm:

Pass 1: write each byte until it matches (P times)

Pass 2: write each byte until it matches (P times),
and then with one 2 * 1000 uS pulse

Maximum P=25

>

- Use this picture as a guide to measure the 16 address line voltages. Try it again with address bits at the opposite polarities:

>WA 5555

>

Resolve any problems with the address drivers before continuing. **Note that the absence of a load on the upper address lines may cause a false voltage reading.** If a signal is not a solid logic-low, test it again after inserting a 1K resistor in the ZIF socket, between ground (pin 14) and the pin you are testing. If the voltage levels are acceptable with this test resistor, then the driver is okay.

Step 4. Test the Data Receivers

>TVCC 1

Vcc pin 28 active state

(ZIF pin 28 should measure about 5.25V.)

>WD 0

>RD

Data Read: 00

>

- Use a 220 ohm resistor to pull each of the data pins high (to pin 28 of the ZIF socket) or low (to pin 12 of the ZIF socket), and test the result with the RD command. (Floating pins will have random results.)

Step 5. Test ZIF pin 28 modes

- Pin 28 should now measure about 5.2V.

>TVCC 2

Vcc pin 28 at programming level

>

Pin 28 should now measure 6.00V.
 >TVCC 0
 Vcc pin 28 inactive state
 >

Pin 28 should now measure less than 0.3V.

Step 6. Test Vcc = 6.37V on ZIF pin 28

Select an EPROM with programming Vcc=6.25V.

>ET 50
 Current Device Type is 50: 27C512-Intel

Turn on Vcc for programming.

>TVCC 2
 Vcc pin 28 at programming level
 >

Pin 28 should now measure 6.37V.
 >TVCC 0
 Vcc pin 28 inactive state
 >

Step 7. Test ZIF pin 22 modes

>ET 53
 Current Device Type is 53: 27C512AMC
 >TVPP 3
 Vpp pin 22 programming mode 13.15V
 >

Pin 22 should now measure 13.1V.
 >TVPP 1
 Vpp pin 22 read mode
 >

Pin 22 should now measure about 4.8V
 >TVPP 0
 Vpp pin 22 powered off
 >

Pin 22 should be less than 0.3V

Step 8. Test Vbb and Vdd

>ET 5
 Current Device Type is 05: 2708
 (The Busy LED should have turned off.)
 >TVBD 1
 -5V Vbb pin 21 and +12V Vdd pin 19 active state
 >

ZIF socket pin 23 should now measure about -5.1V, and ZIF socket pin 21 should now measure about +12V.
 >TVBD 0
 -5V Vbb pin 21 and +12V Vdd pin 19 inactive state
 >

ZIF socket pins 23 and 21 should now measure about 0V.

Step 9. Test -OE modes on ZIF pin 22

- ```

>TOE 2
-OE pin 20 at programming level
>
 ZIF pin 22 should now measure about +12V
>TOE 1
-OE pin 20 active state
>
 ZIF pin 22 should now measure about 0V
>TOE 0
-OE pin 20 inactive state
>
 ZIF pin 22 should now measure about 4.8V.

```

**Step 10. Test ZIF pin 20 positive voltage modes**

- ```

>TVPP 3
Vpp pin 18 programming mode 25.8V
>
 ZIF pin 20 should now measure about 25.8V.
>TVPP 0
Vpp pin 18 powered off
>
 ZIF pin 20 should now measure less than 0.4V.
>ET 25
Current Device Type is 25: 27HC641S
>WA 1000
>
 ZIF pin 20 should now measure more than 3.2V
>WA 0
>
 ZIF pin 20 should now measure less than 0.4V.

```

Step 11. Test ZIF pin 21 logic modes

- ```

>WA 800
>
 ZIF pin 21 should now measure about 4.5V.
>WA 000
>
 ZIF pin 21 should now measure less than 0.4V.

```

**Step 12. Test ZIF pin 23 positive modes**

- ```

>ET 6
Current Device Type is 06: TMS2758
>TVPP 3
Vpp pin 21 programming mode 25.2V
>
 ZIF pin 23 should now measure about 25.2V.
>TVPP 1
Vpp pin 21 read mode
>
 ZIF pin 23 should now measure about 4.5V

```

```
>TVPP 0
Vpp pin 21 powered off
>
```

- ZIF pin 23 should now measure less than 0.2V. (Less than 0.4V with no load)

Step 13. Test ZIF pin 1 modes

```
>ET 36
Current Device Type is 36: 27128-Intel
>TVPP 3
Vcc pin 24 at programming level
>
```

- ZIF pin 1 should now measure about 21V.

```
>TVPP 1
Vcc pin 24 read mode
>
```

- ZIF pin 1 should now measure about 4.5V.

```
>TVPP 0
Vcc pin 24 powered off
>
```

- ZIF pin 1 should now measure less than 0.39V.

Step 14. Test ZIF pin 26 modes

```
>ET E
Current Device Type is 0E: TMS2716
>TVCC 2
Vcc pin 24 at programming level
>
```

- ZIF pin 26 should now measure about 12V.

```
>TVCC 1
Vcc pin 24 on
>
```

- ZIF pin 26 should now measure about 5.2V.

```
>TVCC 0
Vcc pin 24 off
>
```

- ZIF pin 26 should now measure less than 0.2V.

```
>ET 10
Current Device Type is 10: 57C191C
>TVCC 2
Vcc pin 24 at programming level
>
```

- ZIF pin 26 should now measure about 6.37V.

```
>TVCC 1
Vcc pin 24 on
>
```

- ZIF pin 26 should now measure about 5.2V.

```
>TVCC 0
Vcc pin 24 off
>
```

- ZIF pin 26 should now measure less than 0.2V.

Step 15. Test EPROM ID function on ZIF pin 24

```
>TID 0
ZIF Pin 24 ADC result: A0      {The exact hex value may vary)
Press any key to end
>
```

- ZIF pin 24 should now measure about 12V.
Press any key.

Step 16. Reset When Done

```
>RESET
*=====*
*           ME27000           *
*=====*
* Orphan EPROM Programmer II *
*   By Martin Eberhard       *
*   Firmware Version 1.00    *
*=====*

Serial Number: C001
Current Device Type is 10: 57C191C
EPROM data invert: off
Type ? for command list
>
```

 Step 17. Test with some EPROMs

Use the BF command to fill the buffer with a pattern. Select the correct EPROM Device Type with the ET command, and then program a few EPROMs to verify basic functionality.

Section 10. Functional Testing

Power-off the ME27000 Programmer. If you were testing using a terminal, then connect it to a computer with a terminal program that can send and receive files. Set up the terminal program for 9600 baud, 1 stop bit, no parity. This program expects a display screen that is at least 24 rows of 80 columns, so adjust the display of your terminal appropriately.

Power-on the ME27000, and see that your terminal program can talk to it.

Note: The ME27000 Programmer uses a 'File Address Offset' when uploading and downloading files. The File Address Offset is set by the user (with the **FAO** command), and defines an 8-bit offset for the high address byte in the hex files. During uploads, this file address offset is added to the high address byte in the hex records. During downloads, the record data is only loaded into the buffer if the high address byte in the hex record minus the File Address Offset is 00 through 1F. If you issue the **FAO** command with no parameters, then you have selected automatic file address offset mode, where the file address offset is assumed to be the high-byte of the address in the first received hex record.

During downloads, the hex records are checked for valid record types, correct checksum, legitimate hexadecimal characters, correct record count (for Motorola S5 records). Any errors in these checks will generate a brief error message and bump the error count.

The record count, loaded record count (records with where the address high byte minus the File Address Offset was between 00 and 1F), and error count are displayed, and then reset whenever an end-of-file record is encountered.

Note that no command is required to start downloading to the ME27000 Programmer. The ME27000 simply detects a valid Intel Hex (any line that starts with ':') or Motorola record (any line that starts with 'S'). (Interestingly, you could mix and match S-records and Intel Hex records in the same download...)

10.1 Basic Buffer Operations and File Transfer

You can always pause ME27000 transmission using the space bar on your keyboard. Any key will restart transmission when paused.

Step 1. Display the Default Buffer Data

```
>BD
0000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

<etc.>

1FC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
1FD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
1FE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
1FF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
Buffer checksum: 00
>
```

Step 2. Fill the Buffer with a Constant

```

>BF 55
>Buffer filled with 55
>BD 40 100
040: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
050: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
060: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
070: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
080: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
090: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0A0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0B0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0C0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0D0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0E0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0F0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
100: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
110: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
120: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
130: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
Buffer range checksum: 00
>FB AA
>Buffer filled with AA
>BD 32 81
032: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  *****
040: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  *****
050: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  *****
060: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  *****
070: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  *****
080: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  *****
090: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  *****
0A0: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  *****
0B0: AA AA AA  *****
Buffer range checksum: AA
>

```

(Note that you can display portions of the buffer by specifying the start address and the number of bytes to display.)

Step 3. Edit the Buffer

```

>BE 110
110: AA 01 AA 02 AA 03 AA 04 AA 05 AA 06 AA 07 AA 08
118: AA 09 AA <control-C>

>BD 100 40
100: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  *****
110: 01 02 03 04 05 06 07 08 09 AA AA AA AA AA AA AA AA  .....*****
120: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  *****
130: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  *****
Buffer range checksum: B3
>

```

Step 4. Upload Buffer Contents to your Computer as an Intel Hex File

You will need to use your terminal program to capture the file in your computer. I suggest calling the file INTEST.TXT.

For this demonstration, I am randomly setting the page address to 0x68 - you will see the result in the file.

```

>FAO 68
>File Address Offset: 68

```

```

>UI 0 200      {start file capture before hitting Return}
:10680000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA8
:10681000AAAAAAAAAAAAAAAAAAAAAAAAAAAA9A8
:10682000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAC8
:10683000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAB8
:10684000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA8
:10685000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA98
:10686000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA88
:10687000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA78
:10688000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA68
:10689000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA58
:1068A000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA48
:1068B000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA38
:1068C000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA28
:1068D000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA18
:1068E000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA08
:1068F000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAF8
:10690000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAE7
:10691000010203040506070809AAAAAAAAAAAA4
:10692000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAC7
:10693000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAB7
:10694000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA7
:10695000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA97
:10696000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA87
:10697000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA77
:10698000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA67
:10699000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA57
:1069A000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA47
:1069B000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA37
:1069C000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA27
:1069D000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA17
:1069E000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA07
:1069F000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAF7
:0000001FF
>

```

{Stop capturing and close the file on your computer}
 Use a text editor to examine the file INTEST.TXT, to make sure it transferred correctly, and to delete the '>' at the end.

Step 5. Upload Buffer Contents to your Computer as a Motorola S-record File

You will need to use your terminal program to capture the file in your computer. I suggest calling the file STEST.TXT.

For this demonstration, I am randomly setting the page address to 0x31 - you will see the result in the file.

```

>FAO 31
>File Address Offset: 31
>US 0 200      {start file capture before hitting Return}

S1133100AAAAAAAAAAAAAAAAAAAAAAAAAAAA1B
S1133110AAAAAAAAAAAAAAAAAAAAAAAAAAAA0B
S1133120AAAAAAAAAAAAAAAAAAAAAAAAAAAAAFB
S1133130AAAAAAAAAAAAAAAAAAAAAAAAAAAAAEB
S1133140AAAAAAAAAAAAAAAAAAAAAAAAAAAAADB
S1133150AAAAAAAAAAAAAAAAAAAAAAAAAAAAACB
S1133160AAAAAAAAAAAAAAAAAAAAAAAAAAAAABB
S1133170AAAAAAAAAAAAAAAAAAAAAAAAAAAAAB
S1133180AAAAAAAAAAAAAAAAAAAAAAAAAAAA9B
S1133190AAAAAAAAAAAAAAAAAAAAAAAAAAAA8B
S11331A0AAAAAAAAAAAAAAAAAAAAAAAAAAAA7B
S11331B0AAAAAAAAAAAAAAAAAAAAAAAAAAAA6B
S11331C0AAAAAAAAAAAAAAAAAAAAAAAAAAAA5B

```

```
S11331D0AAAAAAAAAAAAAAAAAAAAAAAAAAAAA4B
S11331E0AAAAAAAAAAAAAAAAAAAAAAAAAAAAA3B
S11331F0AAAAAAAAAAAAAAAAAAAAAAAAAAAAA2B
S1133200AAAAAAAAAAAAAAAAAAAAAAAAAAAAA1A
S1133210010203040506070809AAAAAAAAAAD7
S1133220AAAAAAAAAAAAAAAAAAAAAAAAAAAAAFA
S1133230AAAAAAAAAAAAAAAAAAAAAAAAAAAAAEA
S1133240AAAAAAAAAAAAAAAAAAAAAAAAAAAAADA
S1133250AAAAAAAAAAAAAAAAAAAAAAAAAAAAACA
S1133260AAAAAAAAAAAAAAAAAAAAAAAAAAAAABA
S1133270AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
S1133280AAAAAAAAAAAAAAAAAAAAAAAAAAAA9A
S1133290AAAAAAAAAAAAAAAAAAAAAAAAAAAA8A
S11332A0AAAAAAAAAAAAAAAAAAAAAAAAAAAA7A
S11332B0AAAAAAAAAAAAAAAAAAAAAAAAAAAA6A
S11332C0AAAAAAAAAAAAAAAAAAAAAAAAAAAA5A
S11332D0AAAAAAAAAAAAAAAAAAAAAAAAAAAA4A
S11332E0AAAAAAAAAAAAAAAAAAAAAAAAAAAA3A
S11332F0AAAAAAAAAAAAAAAAAAAAAAAAAAAA2A
S9030000FC
>
```

{Stop capturing and close the file on your computer}

Use a text editor to examine the file STEST.TXT, to make sure it transferred correctly, and to delete the '>' at the end.

Step 6. Test downloading files to the ME27000 Programmer, using the two files we just created. First we will fill the buffer with something different, to be sure. (If you are paranoid, power-cycle the ME27000.) Note that we set the File Address Offset to match the base address in the hex file - otherwise nothing will get loaded into the buffer.

```
>BF 99
>Buffer filled with 99
>BD 25 44
25: 99 99 99 99 99 99 99 99 99 99 99 .....
30: 99 99 99 99 99 99 99 99 99 99 99 99 99 .....
40: 99 99 99 99 99 99 99 99 99 99 99 99 99 .....
50: 99 99 99 99 99 99 99 99 99 99 99 99 99 .....
60: 99 99 99 99 99 99 99 99 99 .....
Buffer range checksum: A4
>FAO 68
>File Address Offset: 68
```

{Now, start sending the file INTEST.TXT to the ME27000}

```
>:10680000AAAAAAAAAAAAAAAAAAAAAAAAAAAAE8
:10681000AAAAAAAAAAAAAAAAAAAAAAAAAAAA9A
:10682000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAC8
:10683000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAB8
:10684000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA8
:10685000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA98
:10686000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA88
:10687000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA78
:10688000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA68
:10689000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA58
:1068A000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA48
:1068B000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA38
:1068C000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA28
:1068D000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA18
:1068E000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA08
:1068F000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAF8
:10690000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAE7
:1069100010203040506070809AAAAAAAAAAAAA4
```

```

:10692000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAC7
:10693000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAB7
:10694000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA7
:10695000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA97
:10696000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA87
:10697000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA77
:10698000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA67
:10699000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA57
:1069A000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA47
:1069B000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA37
:1069C000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA27
:1069D000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA17
:1069E000AAAAAAAAAAAAAAAAAAAAAAAAAAAAA07
:1069F000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAF7
:0000001FF
Records: 21, Bad Records: 00
20 records loaded into buffer with Address File Offset: 68
>BD 100 100
100: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA *****
110: 01 02 03 04 05 06 07 08 09 AA AA AA AA AA AA AA .....*****
120: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA *****
130: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA *****
140: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA *****
150: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA *****
160: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA *****
170: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA *****
180: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA *****
190: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA *****
1A0: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA *****
1B0: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA *****
1C0: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA *****
1D0: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA *****
1E0: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA *****
1F0: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA *****
Buffer range checksum: 33
>

```

You can test with the file STEST.TXT the same way. Remember that its File Address Offset is 31.

10.2 EPROM Reading and Programming

Here, you need a few blank EPROMs - preferably several types. This section assumes 2732 EPROMs, but you can test with other types instead. Known-good EPROMs would be nice. You will also want an EPROM eraser, as you will be filling EPROMs with junk.

Step 1. Low-voltage Operations

```
>ET 16
```

```
Current Device Type is 16: 2732
```

Install a blank 2732 into the ZIF socket, with pin 1 closest to the ZIF socket handle.

```
>EB
```

```
EPROM is blank
```

```
>
```

```
{or...}
```

```
Error Address: XXXX EPROM: ZZ
```

```
{perhaps several errors}
```

```
Fail
```

```
>
```

Whether or not the EPROM is blank, you can read it back and see what

it contains:

```

>ER
EPROM read into buffer
EPROM checksum:00
>BD 0 100
0000: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
0010: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
0020: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
0030: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
0040: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
0050: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
0060: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
0070: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
0080: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
0090: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
00A0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
00B0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
00C0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
00D0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
00E0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
00F0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
Buffer range checksum: 00
>

```

(Obviously, if the EPROM was not blank, it would not read as all FF's and the checksum would be different.) You can now compare the buffer to the EPROM. Then, you can change the buffer data to force a failure.

```

>EC
EPROM matches buffer
>BE 85
0085: 00 77 00 <control-C>
>EC
Error Address: 0085 Buffer: 77 EPROM: 00
Fail
>

```

Step 2. Programming Operations

First, create some interesting data.

```

>BF 55
>Buffer filled with 55
>BE
0000: 55 1 55 2 55 4 55 8 55 10 55 20 55 40 55 80
0008: 55 AA 55 <control-C>
>BE 19A
019A: 55 12 55 34 55 56 55 78 55 9A 55 BC
01B0: 55 DE 55 F0 55 <control-C>
>BD 0 200
0000: 01 02 04 08 10 20 40 80 AA 55 55 55 55 55 55 ..... @.*UUUUUU
0010: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 .....
0020: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 .....
0030: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 .....
0040: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 .....
0050: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 .....
0060: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 .....
0070: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 .....
0080: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 .....
0090: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 .....
00A0: DE F0 55 55 55 55 55 55 55 55 55 55 55 55 55 .....
00B0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 .....
00C0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 .....

```

```

00D0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
00E0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
00F0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0100: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0110: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0120: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0130: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0140: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0150: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0160: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0170: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0180: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0190: 55 55 55 55 55 55 55 55 55 55 55 12 34 56 78 9A BC  UUUUUUUUUUU.4VX.<
01A0: DE F0 55 55 55 55 55 55 55 55 55 55 55 55 55 55  ^pUUUUUUUUUUUUUUUUUU
01B0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
01C0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
01D0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
01E0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
01F0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
Buffer range checksum: 3C
>

```

Now, program an EPROM. (The "-" character following the word "Programming" in the following example will rotate through the characters -\|/ to create the effect of a spinning propeller, while the EPROM is being programmed.)

```

>EB
EPROM is blank
>EP
Please be sure the EPROM is inserted correctly, with pin 1
closest to the socket handle. Ready to program (Y/N)? Y
Programming -
Verifying
EPROM matches buffer
>

```

If you get any error messages, try again with another EPROM, to determine if the problem is with the EPROM or the ME27000 Programmer.

Clear the buffer, and then calculate the EPROM's checksum. It should be the same as it was in the buffer:

```

>BF 0
Buffer filled with 00
>ES
EPROM checksum: 3C
>

```

Read the EPROM back into the buffer and have a look. If all goes well, it will go like this:

```

>ER
EPROM read into buffer
EPROM checksum: 3C
>BD 0 200
0000: 01 02 04 08 10 20 40 80 AA 55 55 55 55 55 55 55 55  .... @.*UUUUUUUU
0010: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0020: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0030: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0040: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0050: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0060: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0070: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0080: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU

```

```

0090: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
00A0: DE F0 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
00B0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
00C0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
00D0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
00E0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
00F0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0100: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0110: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0120: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0130: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0140: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0150: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0160: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0170: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0180: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
0190: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
01A0: DE F0 55 55 55 55 55 55 55 55 55 55 55 55 55 55  ^pUUUUUUUUUUUUUUUUUU
01B0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
01C0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
01D0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
01E0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU
01F0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUUUUU

```

Buffer checksum: 3C

>

Congratulations, your ME27000 EPROM Programmer appears to function correctly.

Section 11. Printed Circuit Board

11.1 Bill of Materials

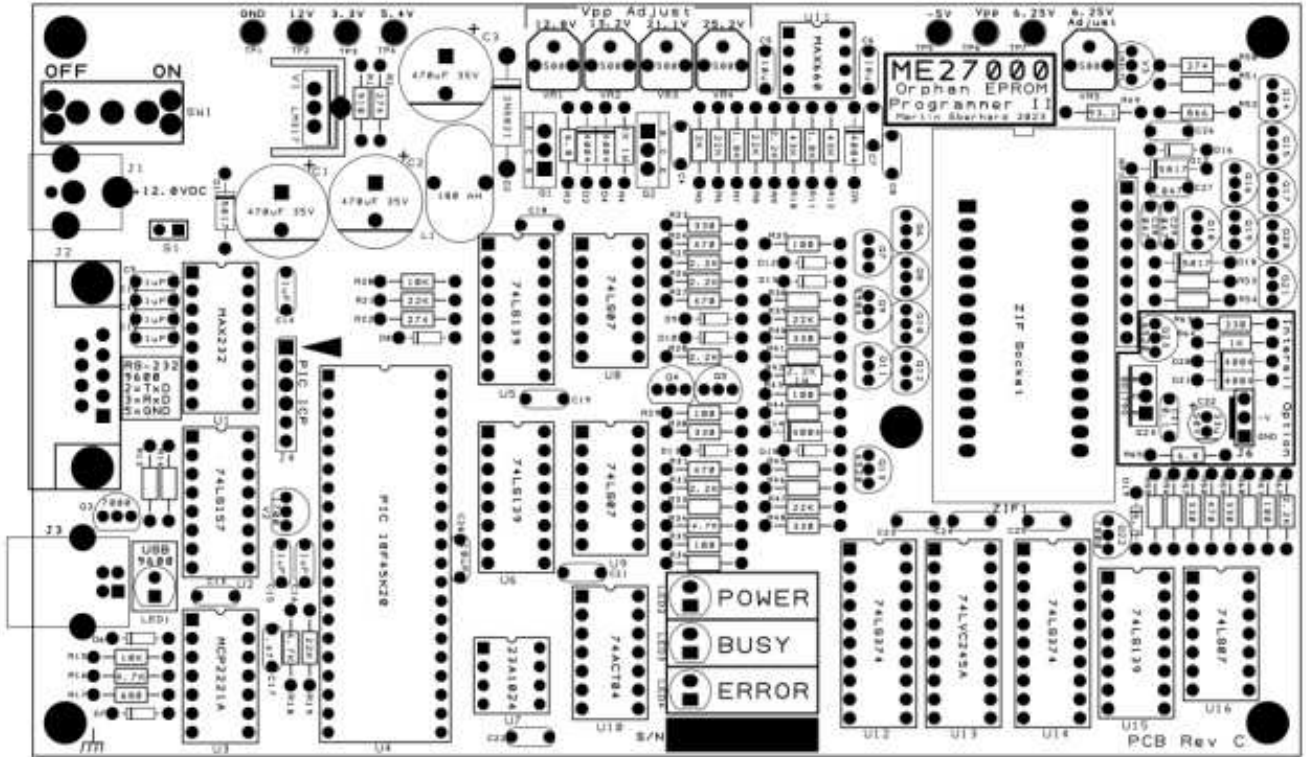
The following is the complete Bill of Materials for the the Rev C ME27000 PC Board, without the Intersil Option. (See page 4 for the Intersil Option components.) Digikey part numbers are current at the time this was written.

Save about \$10: buy the ZIF socket on eBay. Buy the type with wider pin slots for both the normal 0.6" wide EPROMs, and "Skinny DIP" 0.3" wide EPROMs.

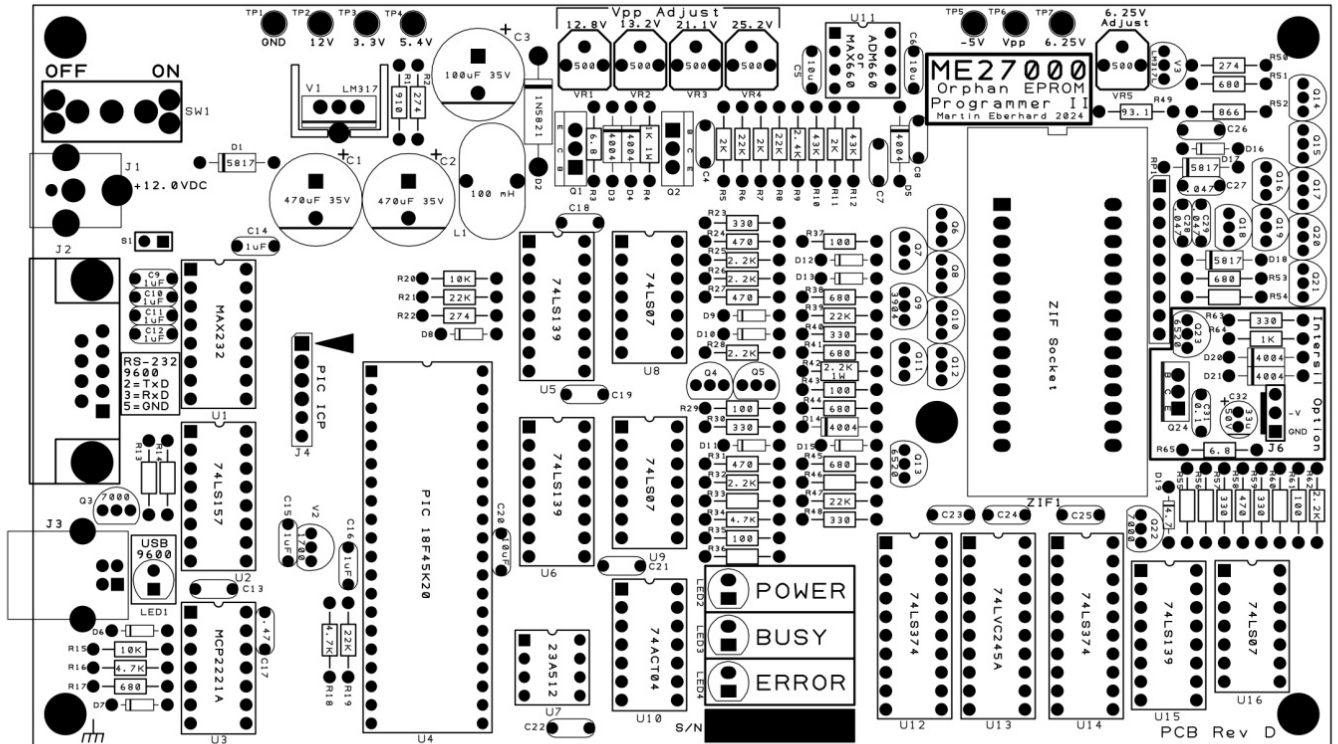
| Component | Value | Reference Name | Qty | Digikey Part # |
|-------------------|------------------|----------------------------------|-----|--------------------------|
| Zener Diode, 4.7V | BZX79C4V7 | D19 | 1 | BZX79C4V7-T50ACT-ND |
| Schottky Diode | BAT46 | D6-D13,D15,D16 | 10 | BAT46CT-ND |
| ¼ W Resistor | 93.1 Ω 1% | R49 | 1 | 93.1XBK-ND |
| ¼ W Resistor | 274 Ω 1% | R2,R22,R50 | 3 | 274XBK-ND |
| ¼ W Resistor | 866 Ω 1% | R52 | 1 | 866XBK-ND |
| ¼ W Resistor | 910 Ω 1% | R1 | 1 | 13-MFR-25FTE52-910RCT-ND |
| ¼ W Resistor | 6.8 Ω | R3 | 1 | 6.8QBK-ND |
| ¼ W Resistor | 100 Ω | R29,R35,R37,R43,R61 | 5 | 100QBK-ND |
| ¼ W Resistor | 330 Ω | R23,R30,R40,R48,R57,R59 | 6 | 330QBK-ND |
| ¼ W Resistor | 470 Ω | R24,R27,R31,R58 | 4 | 470QBK-ND |
| ¼ W Resistor | 680 Ω | R17,R38,R41,R44,R45,R51,R53 | 7 | 680QBK-ND |
| ¼ W Resistor | 2 KΩ | R5,R7,R11 | 3 | 2.0KQBK-ND |
| ½ W Resistor | 2.2 KΩ 1/2W | R25,R26,R28,R32,R62 | 5 | S2.2KHTR-ND |
| ¼ W Resistor | 2.4 KΩ | R9 | 1 | 2.4KQBK-ND |
| ¼ W Resistor | 4.7 KΩ | R16, R18, R34 | 3 | 4.7KQBK-ND |
| ¼ W Resistor | 10 KΩ | R15,R20 | 2 | 2019-CFS1/4C103J-ND |
| ¼ W Resistor | 22 KΩ | R6,R8,R19,R21,R39,R47 | 6 | MFR-25FBF52-22K-ND |
| ¼ W Resistor | 43 KΩ | R10,R12 | 2 | MFR-25FBF52-43K-ND |
| ¼ W Resistor | 1 KΩ | R13,R14,R33,R36 ,R46,R54-R56,R60 | 9 | 1.00KXBK-ND |
| 1W Resistor | 1 KΩ | R4 | 1 | 738-RSMF1JB1K00-ND |
| 1W Resistor | 2 KΩ or 2.2 KΩ | R42 | 1 | 2.0KW-1-ND |
| Schottky Diode | 1N5817 | D1,D17,D18 | 3 | 1N5817GOS-ND |
| Silicon Diode | 1N4004 | D3,D4,D5, D14 | 4 | 1N4004-TPMSCT-ND |
| 8-pin DIP socket | | U7,U11 | 2 | 2057-ICS-308-T-ND |
| 14-pin DIP socket | | U3,U8-U10,U16 | 5 | ED3045-5-ND |
| 16-pin DIP socket | | U1,U2,U5,U6,U15 | 5 | 2057-ICS-316-T-ND |
| 20-pin DIP socket | | U12-U14 | 3 | 2057-ICS-320-T-ND |
| 40-pin DIP socket | | U4 | 1 | ED3048-5-ND |
| Schottky Diode | 1N5821 (30V, 3A) | D2 | 1 | 1N5821RLGOSCT-ND |
| Ceramic Capacitor | 0.047 μF, 50V | C26-C28 | 3 | 445-173599-1-ND |
| Ceramic Capacitor | 0.47 μF 25V | C17 | 1 | 445-181293-ND |
| Ceramic Capacitor | 1μF, 16V | C9-C12,C14-C16 | 7 | 445-173583-1-ND |
| Ceramic Capacitor | 10 μF, 10V | C5,C6,C20 | 3 | 445-181283-ND |

| Component | Value | Reference Name | Qty | Digikey Part # |
|-----------------------------------|-------------------|------------------------------|-----|----------------------------|
| Ceramic Capacitor | 0.1 μ F, 100V | C4,C7,C8,C13,C18,C19,C21-C26 | 12 | 399-4329-ND |
| 8-resistor network | 10K | RP1 | 1 | 4609x-101-103LF-ND |
| 3.3V Regulator | MCP1700-3302E/TO | V2 | 1 | MCP1700-3302E/TO-ND |
| Adj. Regulator | LM317L | V3 | 1 | 497-1573-5-ND |
| NPN Transistor | 2N3904 | Q9 | 1 | 2N3904FS-ND |
| N-channel MOSFET | 2N7000 | Q3,Q22 | 2 | 2N7000FS-ND |
| PNP Transistor | 2N6520 | Q13 | 1 | 2N6520TACT-ND |
| PNP Transistor | 2N3906 | Q4-Q8, Q10-Q12,Q14-21 | 16 | 641-1946-ND |
| Electrolytic Cap. | 100 μ F, 50V | C3 | 1 | 399-ESX107M050AH2AA-ND |
| Electrolytic Cap. | 470 μ F, 35V | C1-C2 | 2 | 399-6086-ND |
| Trimpot Bournes | 500 Ω | VR1-VR5 | 5 | 3306P-501-ND |
| NPN Transistor | KSC2690 | Q1 | 1 | KSC2690AYSFS-ND |
| PNP Transistor | 2SA2222SG | Q2 | 1 | 2SA2222SGOS-ND |
| Barrel Connector | 5.5mm x 2.0mm | J1 | 1 | CP-063AH-ND |
| 9-pin Connector | DC9F | J2 | 1 | AE10921-ND |
| 6-pin connector | USB-B | J3 | 1 | 2057-USB-B-S-RA-WT-SPCC-ND |
| 6-pin header | 0.1" Spacing | J4 | 1 | A31116-ND |
| Torroidal Inductor | 100 μ H, 2A | L1 | 1 | 732-1424-ND |
| Adj. Regulator | LM317 | V1 | 1 | 497-1575-5-ND |
| TO-220 Heat Sink | 577102B04000G | V1 | 1 | 345-1023-ND |
| 6-32 x 5/8" screw | | V1 | 1 | H356-ND |
| 6-32 nut | | V1 | 1 | H220-ND |
| 4-40 x 5/8" screw | | PCB corners & near Q13 | 5 | H348-ND |
| 4-40 nut | | PCB corners & near Q13 | 5 | 36-9600-ND |
| 28-pin ZIF socket | Textool | ZIF1 | 1 | A302-ND |
| SPDT Switch | 5A, 120V | SW1 | 1 | EG2365-ND |
| Blue LED | 5 mm | LED1 | 1 | 67-1751-ND |
| White LED | 5 mm | LED2 | 1 | 1516-QBL8IW60D-NW-ND |
| Amber LED | 5 mm | LED3 | 1 | 754-1872-ND |
| Red LED | 5 mm | LED4 | 1 | 1516-1359-ND |
| Switched Cap. Inverting Regulator | ADM660 or MAX660 | U11 | 1 | ADM660ANZ-ND |
| RS232 Transceiver | MAX232 | U1 | 1 | 296-26139-5-ND |
| USB Transceiver | MCP2221A | U3 | 1 | MCP2221A-I/P-ND |
| Hex Inverter | 74ACT04 | U10 | 1 | 296-4351-5-ND |
| Hex O.C. Driver | 74LS07 | U8, U9,U16 | 3 | 296-14878-5-ND |
| Dual Data Selector | 74LS139 | U5,U6,U15 | 3 | 296-1640-5-ND |
| Quad 2:1 Mux | 74LS157 | U2 | 1 | 296-1645-5-ND |
| 8-Bit D-Flip-Flop | 74LS374 | U12,U14 | 2 | 296-1662-5-ND |
| 8-Bit Bi-dir. Buffer | 74LVC245A | U13 | 1 | 296-8503-5-ND |
| 512 kb Serial RAM | 23A512 | U7 | 1 | 23LC512-I/P-ND |

11.2 Component Placement



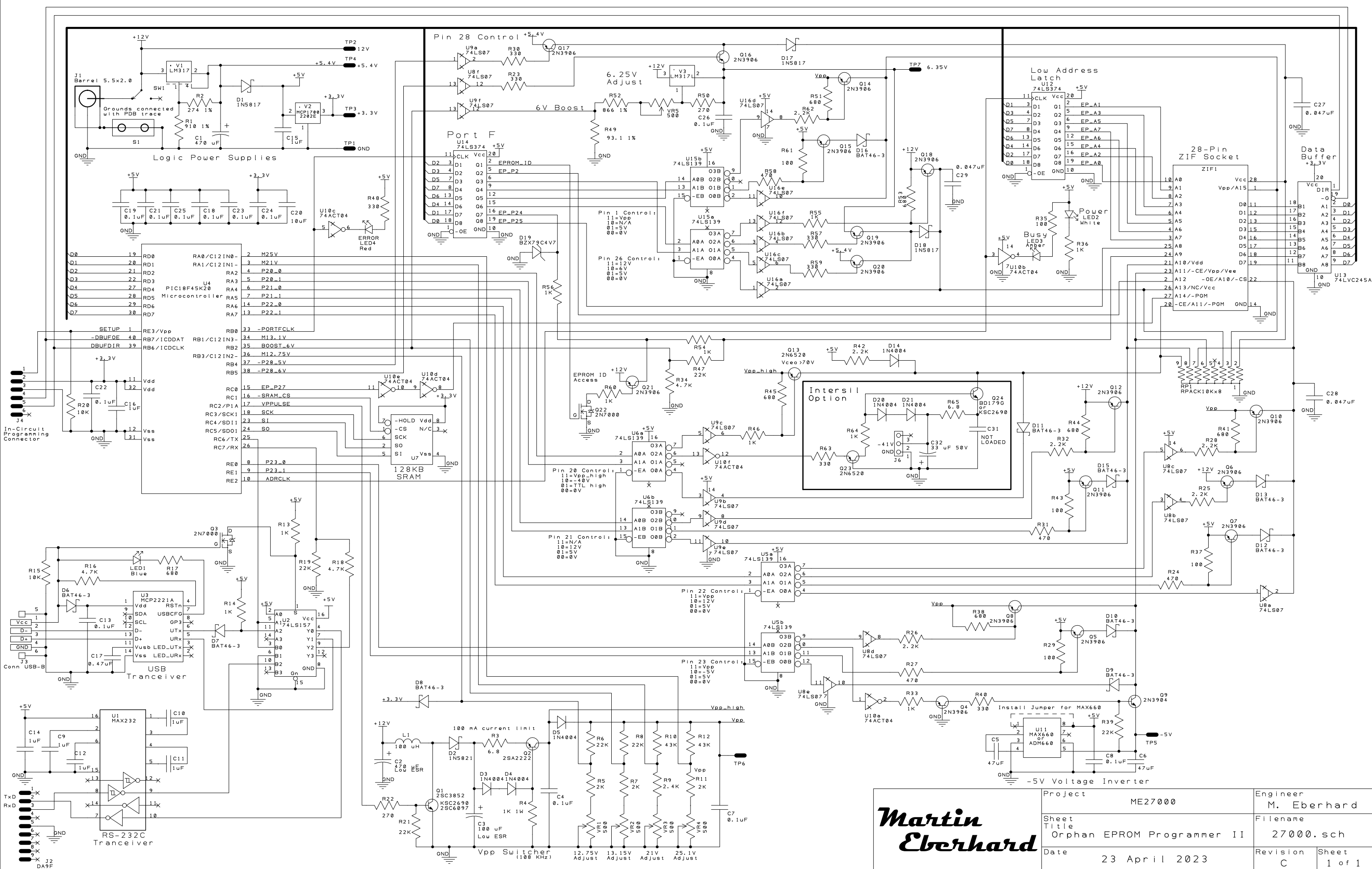
Rev C PC Board



Rev D PC Board

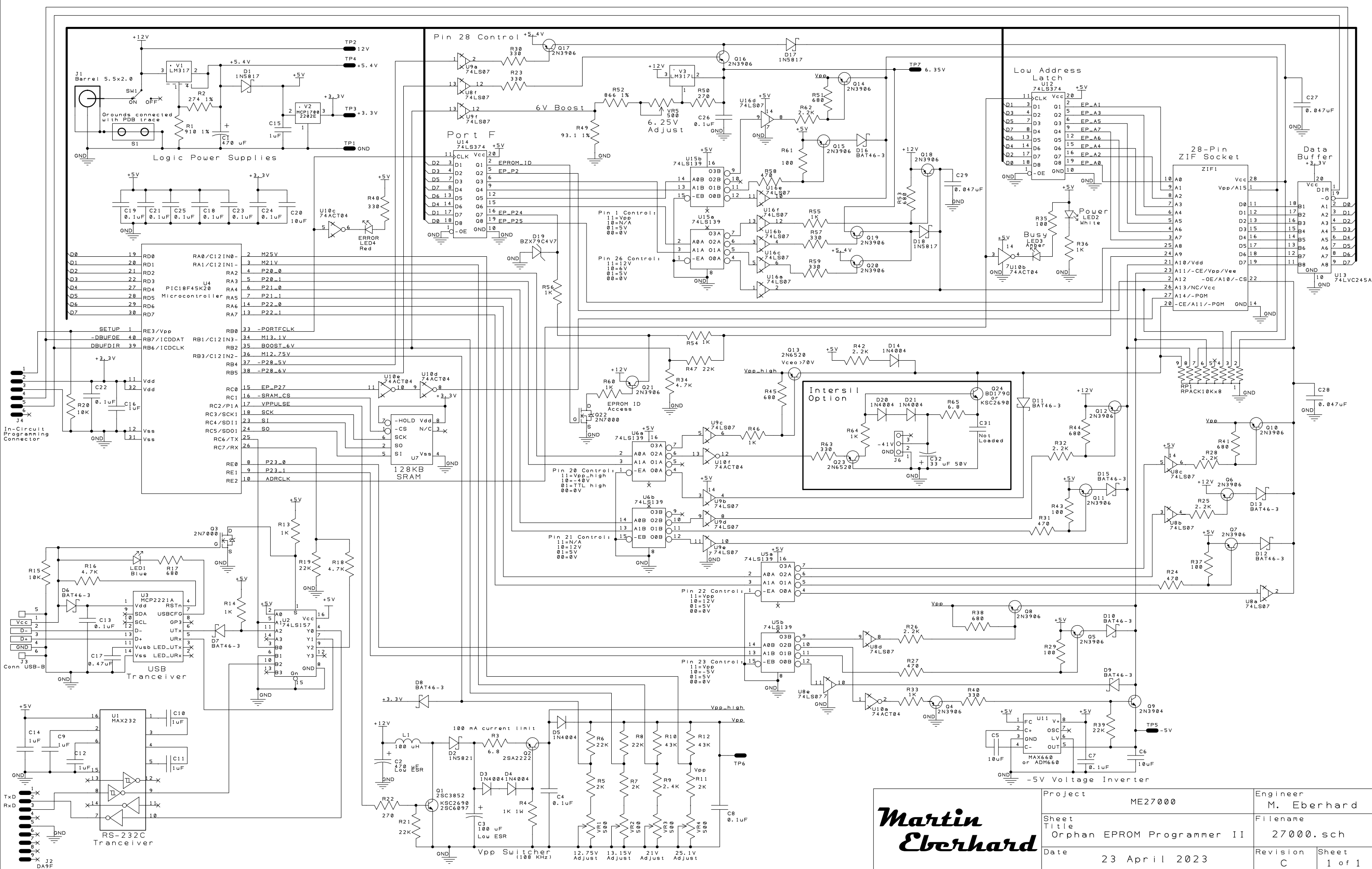
11.3 PCBA Schematic

The following page is the schematic for the rev C ME27000 Programmer's printed circuit board assembly.



Martin Eberhard

| | | | |
|-------------|----------------------------|----------|-------------|
| Project | ME27000 | Engineer | M. Eberhard |
| Sheet Title | Orphan EPROM Programmer II | Filename | 27000.sch |
| Date | 23 April 2023 | Revision | C |
| | | Sheet | 1 of 1 |



Martin Eberhard

| | | | |
|-------------|----------------------------|----------|-------------|
| Project | ME27000 | Engineer | M. Eberhard |
| Sheet Title | Orphan EPROM Programmer II | Filename | 27000.sch |
| Date | 23 April 2023 | Revision | C |
| | | Sheet | 1 of 1 |

Appendix A. Supported Devices

The following is a list of devices supported by the ME27000 firmware. Although generally in order of size, the order is arbitrary. As more devices are supported in future firmware, new Device Types will be added to the end of the list. (So these ME27000 Device Type numbers will not change².)

Underlined devices have been tested on the ME27000. The programming algorithm for devices in [brackets] are just guesses, as I could not find programming specifications for these devices.

Although some of these EPROM Device Types may have the same size and programming voltages, they are separate types because of differences in pinout, Vcc during programming, and/or programming algorithm.

Supported Devices Sorted by Device Type

- Type 00: 2704** **512x8 EPROM**
Intel 2704, National Semiconductor MM2704, Signetics 2704
- Type 01: 2804A** **512x8 EEPROM** (time delay for write completion)
Exel X12804A, Seeq 2804A, Xicor X2804A
- Type 02: 28C04** **512x8 EEPROM** (polled write completion)
Atmel AT28C04, General Instruments 28C04, Microchip 28C04A, NEC 28C04
- Type 03: 28C04N** **512x8 EEPROM, not self-timed**
(No devices tested)
- Type 04: IM6654** **512x8 EPROM** (Requires Intersil Option)
Intersil IM6654
- Type 05: 2708** **1024x8 EPROM** (Vcc=+5V, Vbb=-5V, Vdd=+12V)
AMD AM2708, Electronic Arrays EA2708, Fairchild F2708, Intel 2708, Intel D2708L, MME U555C, Motorola MCM2708, Motorola MCM68708, National Semiconductor MM2708, NTE NTE2708, Oki MSM2708AS, Signetics 2708, Tesla MHB8708C, Texas Instruments TMS2708, Toshiba TMM322
- Type 06: 2758** **1024x8 EPROM** (Vcc=+5V, pin 19 low)
Harris HM-6758, Intel 2758, National Semiconductor MM2758Q-A, Oki 2758, Texas Instruments TMS2508, Texas Instruments TMS2758-JL0
- Type 07: IM6658** **1024x8 EEPROM**, (Requires Intersil Option)
Intersil IM6658
- Type 08: 2716** **2048x8 EPROM** (Vcc=+5V, Vpp=25V)
AMD AM2716, Eurotechnique ET2716Q, Fujutsi MBM2716, Hitachi HN462716, Intel 2716, Mitsubishi M5L2716, MME U2716C, Motorola MCM2716, National Semiconductor MM2716, National Semiconductor MM27C16, NEC uPD2716, NTE NTE2716, Oki MSM2716AS, SGS-Thomson M2716, Signetics 2716, Soviet 573RF2, Tesla MHB2716C, Texas Instruments TMS2516, Thomson-Mostek ET2716Q, Toshiba TMM323D, Toshiba TMM323DI
- Type 09: 2716A** **2048x8 EPROM** (Vcc=+5V, Vpp=21V)
(No datasheets found)
- Type 0A: 2716-fast** **2048x8 EPROM** (Vcc=+5V, Vpp=21V)
ST M2716-fast
- Type 0B: 2716B** **2048x8 EPROM** (Vcc=+5V, Vpp=12.7V)
AMD AM2716B

² These Device Type numbers are not the same as those for the original Orphan Eprom Programmer, the ME2700.

Type 0C: 27C16H **2048x8 EPROM** (Vcc=+5V, Vpp=25V)
 Fairchild NMC27C16H, National Semiconductor NMC27C16H

Type 0D: 27C16B **2048x8 EPROM** (Vcc=+5V, Vpp=12.7V)
 Fairchild NMC27C16B, National Semiconductor NMC27C16B

Type 0E: TMS2716 **2048x8 EPROM** (Vcc=+5V, Vbb=-5V, Vdd=+12V)
 Motorola TMS2716, Texas Instruments TMS2716

Type 0F: 57C191 **2048x8 EPROM**
 Waferscale Integration WS57C191, Waferscale Integration WS57C191B,
 Waferscale Integration WS57C291, Waferscale Integration WS57C291B

Type 10: 57C191C **2048x8 EPROM**
Waferscale Integration WS57C191C, Waferscale Integration WS57C291C

Type 11: LH57191 **2048x8 EEPROM**
 Sharp LH57191

Type 12: 2816A **2048x8 EEPROM** (time delay for write completion)
 Samsung KM2816A, Seeq 2816A, Seeq 5516A

Type 13: 28C16 **2048x8 EEPROM** (polled write completion)
 Atmel AT28C16, Atmel 28C16E, Catalyst CAT28C16A, Exel XLS2816A, Exel XLS28C16A, Microchip 28C16A, On Semiconductor CAT28C16A, Xicor X2816B

Type 14: 2816Ai **2048x8 EEPROM** (erase before write, >10 mS write pulse)
 Intel 2816A, Seeq 52B13

Type 15: 52B13H **2048x8 EEPROM, 1.2 mS write pulse**
Seeq 52B13H

Type 16: 2732 **4096x8 EPROM** (Vpp=25V)
 AMD AM2732, Electronic Arrays EA2732Q, Eurotech. ETC2732, Fairchild F2732,
Fujitsu MBM2732, Hitachi HN472732G, Intel 2732, Mitsubishi M5L2732, MME
U2732, Motorola MCM2732, NEC uPD2732, Toshiba TMM2732D, Toshiba TMM2732DI

Type 17: 2732A **4096x8 EPROM** (Vpp=21V)
 AMD AM2732A, Fujitsu MBM2732A, Hitachi HN482732AG, Intel 2732A, NEC
 uPD2732A, Rockwell R87C32, SGS M2732A

Type 18: 2732A-fast **4096x8 EPROM** (Vpp=21V)
 SGS-Thomson M2732A-fast

Type 19: 2732B **4096x8 EPROM** (Vpp=12.7V)
AMD AM2732B

Type 1A: 27C32H **4096x8 EPROM** (Vpp=12.7V)
 Fairchild NMC27C32H, National Semiconductor NMC27C32H

Type 1B: 27C32B **4096x8 EPROM** (Vpp=12.7V)
 Fairchild NMC27C32B, National Semiconductor NMC27C32B

Type 1C: TMS2532 **4096x8 EPROM** (TI-unique pinout, Vpp=25V)
 Hitachi HN462532, Motorola MCM2532, SGS M2532, Texas Instruments TMS2532

Type 1D: TMS2532A **4096x8 EPROM** (TI-unique pinout, Vpp=21V)
Texas Instruments TMS2532A

Type 1E: TMS2732A **4096x8 EPROM** (TI-unique programming, Vpp=21V)
Texas Instruments TMS2732A

Type 1F: 57C43 **4096x8 EPROM**
Waferscale Integration WS57C43, Waferscale Integration WS57C43B

Type 20: 57C43C **4096x8 EPROM**
[Waferscale Integration WS57C43C]

Type 21: LH5749 **8192x8 EPROM**
 Sharp LH5749

Type 22: 57C49 8192x8 EPROM
Waferscale Integration WS57C49, Waferscale Integration WS57C49B

Type 23: 57C49C 8192x8 EPROM
[Waferscale Integration WS57C49C]

Type 24: 27HC641 8192x8 EPROM
Atmel AT27HC641, Atmel AT27HC642, Microchip 27HC641

Type 25: 27HC641s 8192x8 EPROM (no blank-check)
Signetics 27HC641

Type 26: 27HC641R 8192x8 EPROM
Atmel AT27HC641R, Atmel AT27HC642R, Microchip 27HC641

Type 27: 27HC65 8192x8 EPROM
NEC uPD27HC65

Type 28: 68764 8192x8 EPROM
Motorola MCM68764, Motorola MCM68766

Type 29: 2764 8192x8 EPROM
AMD 2764, Intel D2764, NTE NTE2764, Rockwell R2764

Type 2A: 2764-Intel 8192x8 EPROM
Epson SPM27C64, Epson SPM27C64H, Fujitsu 2764, Fujitsu 27C64, Hitachi 482764AG, Intel D2764, Mitsubishi M5L2764, Seeq 2764

Type 2B: 2764A 8192x8 EPROM
Atmel AT27HC64, Intel D2764A, NTE NTE27C64, Sharp LH5762J, Sharp LH5763J, Sharp LH5764J, Signetics 2764A, ST/Eurotech. 27C64A, St.CGS 27C64A, Texas Instruments 27C64, Toshiba TMM2764A

Type 2C: 2764A-AMD 8192x8 EPROM
AMD 2764A, Hyundai HY27C64

Type 2D: 27C64 8192x8 EPROM
Fairchild NM27C64, National Semiconductor NM27C64

Type 2E: 27C64-EXP 8192x8 EPROM
Microchip 27C64

Type 2F: 27C64-Intel 8192x8 EPROM
AMD 27C64, Hitachi HN27C64, Intel D27C64, Oki MSM2764A

Type 30: 27C64F 8192x8 EPROM
Waferscale 27C64F, Waferscale 57C64F

Type 31: TMS2564 8192x8 EPROM
Texas Instruments TMS2564

Type 32: 28C64 8192x8 EEPROM, no software lock feature
Atmel AT28C64, Atmel AT28C64H, Catalyst 28C64A, Exel 28C64, Microchip 28C64A, NEC 28C64, Pyramid PYA28C64, Samsung KY2864A, Seeq 28C64, Seeq 85B01

Type 33: 28C64-Lock 8192x8 EEPROM, with software lock feature
Atmel AT28C64B, Catalyst 28C64B, Exel 28C64B, Microchip 28C64B, Pyramid PYA28C64B, Pyramid PYX28C64, ST 28C64, Turbo 28C64A, Xicor 28C64

Type 34: 27128 16384x8 EPROM
Toshiba TMM27128

Type 35: 27128-Fujitsu 16384x8 EPROM
Fujitsu 27128, Fujitsu 27C128, Toshiba TMM27128

Type 36: 27128-Intel 16384x8 EPROM
Hitachi 4827128, Intel D27128, Mitsubishi M5L27128, Mitsubishi M5M27128, NEC 27128, Oki 27C128A, Seeq 27128, Texas Instruments 27128

Type 37: 27128A 16384x8 EPROM
Hitachi 4827128A, Intel 27128A, Intel 27C128, Microchip 27C128, NTE 21128, Sharp LH57127, Sharp LH57128, ST/SGS 27128A, Texas Instruments 27C128, Toshiba TMM27128A

Type 38: 27128A-AMD 16384x8 EPROM
AMD 27128A

Type 39: 27C128 16384x8 EPROM
Fairchild NM27C128, National Semiconductor NM27C128

Type 3A: 27C128-Fast 16384x8 EPROM
AMD 27C128, National Semiconductor NM27C128, Oki M5M27128A

Type 3B: 27C128-Cyp 16384x8 EPROM
[Cypress CY27C128]

Type 3C: 27C128F 16384x8 EPROM
Waferscale 27C128F, Waferscale 57C128F

Type 3D: 57C51 16384x8 EPROM
Waferscale 57C51, Waferscale 57C51B

Type 3E: 57C51C 16384x8 EPROM
[Waferscale 57C51C]

Type 3F: 27256 32768x8 EPROM
Hitachi 27256, NEC uPD27256, NEC uPD27C256, Toshiba TC57256

Type 40: 27256-Intel 32768x8 EPROM
AMD 27256, Atmel AT27HC256, Cypress 27256, Epson SPM27C256, Epson SPM27C256H, Intel 27256, NEC uPD27C256A

Type 41: 27256-Fast 32768x8 EPROM
AMD 27C256, AMD 27C256H, Intel 27256, ISSI 27C256, Macronix MX27C256, National Semiconductor NM7C256, Oki 27256, Oki 27C256, Sharp LH57256J, Sony 27C256, ST 27C256B, ST M87C257, Toshiba TC57256A, NTE NTE27C256

Type 42: 27C256 32768x8 EPROM
Atmel AT27C256, Eurotechnique 27C256, Fujitsu 27256, Fujitsu 27C256, Hitachi 27C256, Hitachi 27C256A, Intel 27C256, Mitsubishi M5M27256, Mitsubishi M5M27C256, Mitsubishi M5M27L256, Seeq 27C256, Sharp LH57254J, Signetics 27C256, ST 27256, Texas Instruments 27C256, Toshiba TC57H256, Toshiba TMM27256A

Type 43: 27256-AMD 32768x8 EPROM
AMD 27256

Type 44: 27C256-Rapid 32768x8 EPROM
Atmel 27BV256, Atmel 27C256R, Atmel 27LV256A, GI 27C256, Microchip 27C256, Microchip 27HC256, Microchip 27LV256

Type 45: 27C256-Turbo 32768x8 EPROM
Fairchild FM27C256, National Semiconductor NM27C256

Type 46: 27C256F 32768x8 EPROM
Waferscale 27C256F, Waferscaler 57C256F, Microchip 27C256F, Microchip 57C256F

Type 47: 27C256L 32768x8 EPROM
Waferscale 27C256L

Type 48: 27SC256-Cypress 32768x8 EPROM
[Cypress CY27C256]

Type 49: 27SF256 32768x8 EPROM
SST 27SF256

Type 4A: 57C71C 32768x8 EPROM
 [Waferscale WS57C71C]

Type 4B: 28C256 32768x8 EEPROM (without software lock)
 Hitachi HN58C256, Maxwell 28C256T, Samsung 28C256, SEEQ 28C256

Type 4C: 28C256-Lock 32768x8 EEPROM (with software lock and 64-byte pages)
 Atmel AT28C256, Catalyst CY28C256, Exel 28C256, Hitachi HN58C256A, NEC uPD28C256, Pyramid PYA28C256, Samsung KM28C256A, Seeq 28C256A, Turbo IC 28C256A, Xicor 28C256

Type 4D: 29F256 32768x8 Flash (with software lock and 64-byte pages)
 Atmel AT29F256

Type 4E: 27512 65536x8 EPROM
AMD 27512

Type 4F: 27C512 65536x8 EPROM
 Atmel AT27C512, Cypress CY27C512, Fujitsu 27C512, Hitachi 27512, Hitachi 27C512, Intel D27512, Mitsubishi M5L27512, Mitsubishi M5M27C512A, NEC uPD27512, Signetics 27C512, ST 27512, Texas Instruments 27C512, Toshiba TMM27512A, Toshiba TMM57C512

Type 50: 27C512-Intel 65536x8 EPROM
 AMD 27C512, Intel D27C512, ISSI 27C512, National Semiconductor NM27C512, National Semiconductor NM27C512A, NTE NTE27C512, Oki 27C512, Sony 27C512, ST 27C512, Toshiba 27C512A, Toshiba TMM27512A

Type 51: 27C512-Rapid 65536x8 EPROM
 Atmel AT27BV512, Atmel AT27C512R, Atmel 27LV512

Type 52: 27C512-Turbo 65536x8 EPROM
 Fairchild FM27C512, National Semiconductor NM27C512, National Semiconductor NM27C512A

Type 53: 27C512-MC 65536x8 EPROM
 GI 27C512A, Microchip 27C512, Microchip 27LV512

Type 54: 27C512-MX 65536x8 EPROM
Macronix MX27C512, Macrinix MX27L512

Type 55: 27C512F 65536x8 EPROM
 Waferscale WS27C512F, Waferscale WS57C512F

Type 56: 27C512L 65536x8 EPROM
 Waferscale WS27C512L

Type 57: 27C512-EON 65536x8 EPROM
 Eon EN27C512

Type 58: 27C512-Cypress 65536x8 EPROM
 [Cypress CY27C512]

Type 59: 57512 65536x8 EPROM
Sharp LH57512J

Type 5A: 27SF512 65536x8 SuperFlash EPROM
SST 27SF512

Supported Devices Sorted by Manufacturer

Numbers in parenthesis are the ME27000 Device Type. As above, chips that have been tested on the ME27000 are underlined.

AMD

AM2708 (05), AM2716 (08), AM2716B (0B), AM2732 (16), AM2732A (17), AM2732B (19), AM2764 (29), AM2764A (2C), AM27C64 (2F), AM27128A (3F), AM27C128 (3A), AM27256 (43), AM27C256 (41), AM27H256 (41), AM27512 (4E), AM27C512 (50)

Atmel

AT28C04 (02), AT28C16 (13), 28C16E (13), AT27HC641 (24), AT27HC642 (24), AT27HC641R (26), AT27HC642R (26), AT28C64 (32), Atmel AT28C64H (32), AT27HC256 (40), AT27C256 (42), 27BV256 (44), 27C256R (44), 27LV256A (44), AT28C256 (4C), AT29F256 (4D), AT27C512 (4F), AT27BV512 (51), AT27C512R (51), 27LV512 (51)

Catalyst

CAT28C16A (13), CAT28C64A (32), CAT28C64B (33), CAT28C256 (4C)

Cypress

CY27C128 (3B), CY27C256 (48), CY27C512 (58)

Electronic Arrays

EA2708 (05)

Eon

EN27C512 (57)

Epson

SPM27C64 (2A), SPM27C64H (2A), SPM27C256 (40), SPM27C256H (40)

Eurotechnique

ET2716Q (08) ETC2732Q (16), 27C64A (2B), 27C256 (42)

Exel

X12804A (01), XLS2816A (13), XLS28C16A (13)

Fairchild

F2708 (05), NM2716H (0C), NM27C16B (0D), F2732 (16), NMC27C32B (1B), NMC27C32H (1C), NMC27C64 (2D), NM27C128 (39), FM27C256 (45), FM27C512 (52)

Fujitsu

MBM2716 (08), MBM2732 (16), MBM2732A (17), MBM2764 (2A), MBM27C64 (2B), MBM27128 (35), MBM27256 (42), MBM27C256 (42)

General Instruments

28C04 (02), 27C256 (44), 27C512 (53)

Harris

HM-6758 (06)

Hitachi

HN462716 (08), HN462532 (16), HN472732AG (17), HN462532 (1C), 482764AG (2A), HN27C64 (2F), HN4827128 (36), HN27128A (37), HN27256 (3F), HN27C256 (42), HN27C256A (42), HN58C256 (4B), HN58C256A (4C), HN27512 (4F),

Hyundai

HY27C64 (2C)

Intel

D2704 (00), D2708 (05), D2708L (05), D2758 (06), D2716 (08), D2816A (14), D2732 (16), D2732A (17), D2764 (29 or 2A), D2764A (2B), D27C64 (2F), D27128 (36), D27128A (37), D27C128 (37), D27256 (40 or 41), D27C256 (42), D27512 (4F)

Intersil

IM6654 (04), IM6658 (07)

ISSI

27C256 (41), 27C512 (50)

Macronix

MX27C256 (41), MX27C512 (55), MX27L512 (55)

Microchip

28C04A (02), 28C16A (22), 27HC641 (23 or 25), 27C64 (2E), 28C64A (32), 28C64B (33), 27C128 (37), 27C256 (44), 27HC256 (44), 27LV256 (44), 57C256F (46), 27C512 (54), 27C512A (54), 27LV512 (54)

Mitsubishi

M5L2716K (08), M5L2732 (16), M5L2764 (2A), M5L27128 (36), M5M27128 (36), M5M27256 (42), M5M27C256 (42), M5M27L256(42), M5L27512 (50), M5M27C512A (50)

MME

U555C (05), U2716C (08), U2732 (16)

Motorola

MCM2708 (05), MCM68708 (05), TMS2716 (0E), MCM2532 (16), MCM2732 (1C), MCM68764 (28), MCM68766 (28)

National Semiconductor

MM2704 (00), MM2708 (05), MM2758A (06), MM2716E (08), NMC27C16 (08), NM2716H (0C), NM27C16B (0D), NM2732H (1A), NM2732B (1B), NM27C64 (2D), NM27C128 (39 or 3A), NM27C256 (41 or 45), NM27C512 (4F or 51), NM27C512A (4F or 51)

NEC

uPD28C04 (02), uPD2716 (09), uPD2732 (16), uPD2732A (17), uPD27HC65 (27), uPD28C64 (32), uPD27128 (36), uPD27256 (3F), uPD27C256 (3F), uPD27C256A (40), uPD28C256 (4C), uPD27C512 (4F)

NTE

NTE2708 (05), NTE2716 (09), NTE2732A (17), NTE2532 (1C), NTE2764 (29), NTE27C64 (2B), NTE21128 (37), NTE27C256 (41), NTE27C512 (50)

On Semiconductor

(See Catalyst)

Oki

MSM2708AS (05), MSM2758 (06), MSM2716A (08), MSM2764A (2F), 27C128A (36), M5M27128A (3A), MSM27256 (41), MSM27512 (50)

Pyramid

PYA28C64 (32), PYX28C64 (33), PYA28C64B (33), PYA28C256 (4C)

Rockwell

R87C32 (17)

Samsung

KM2816A (12), KM2864A (32), KM28C64A (33), KM28C256 (4A), KM28C256A (4B)

Seeq

2804A (01), 2816A (12), 5516A (12), 52B13 (14), 52B13H (15), Seeq 2764 (2A), 28C64 (32), 85B01 (32) 27128 (36), 27C256 (42), 28C256 (4A), 28C256A (4C)

SGS-Thomson (ST)

M2716 (08), M2716-fast (0A), SGS M2532 (19), M2732A (17), M2732A-fast (18) M2532 (1C), M2764A (2B), M27128A (37)

Sharp

LH57191 (11), LH5749 (21), LH5762 (2B), LH5763 (2B), LH5764 (2B), LH57126(37),
LH57127(37), LH57128(37), LH57256 (41), LH57254 (42), LH57512 (59)

Signetics

2704 (00), 2708 (05), 2716 (08), 27HC641 (25), 2764A (2B), 27C256 (42), 27C512
(4F)

Sony

27C256 (41), 27C512 (50)

Soviet

573RF2 (08)

SST

27SF256 (48), 27SF512 (5A)

Tesla

MHB8708C (05), MHB2716C (08)

Texas Instruments

TMS2708 (05), TMS2508 (06) , TMS2758-JL0 (06), TMS2516 (08) , TMS2716 (0E) ,
TMS2532 (1C), TMS2532A (1D), TMS2732A (1E), TMS27C64 (2B), TMS2564 (31),
TMS27128 (36), TMS27C128 (37), TMS27C256 (42), TMS27C512 (4F)

Thomson-Mostek

ET2716Q (08)

Toshiba

TMM322 (05), TMM323D (08), TMM323DI (08), TMM2732D (16), TMM2732DI (16),
TMM2764A (2B), TMM27128 (35), TMM27128A (37), TC57256 (3F), TC57256A (41),
TMM27256A (42), TC57H256 (42), TMM27512A (4F), TMM27512A (50)

Turbo IC

28C64A (33), 28C256A (4C)

Waferscale Integration

WS57C191 (0F), WS57C191B (0F), WS57C191C (10), WS57C291 (0F), WS57C291B (0F),
WS57C291C (10), WS57C43 (1F), WS57C43B (1F), WS57C43C (20), WS57C49 (22),
WS57C49B (22), WS57C49C (23), WS27C64F (30), WS57C64F (30), WS27C128F (3C),
WS57C128F (3C), WS57C51 (3D), WS57C51B (3D), WS57C51C (3E), WS27C256F (46), WS27C256L
(47), WS57C71C (4A), WS27C512F (55), WS57C512F (56)

Xicor

X2804A (01), X2816B (13), 28C64 (33), 28C256 (4C)

Device Manufacturer Codes

These are the JEDEC-assigned device manufacturer codes for most (all?) EPROM manufacturers. The high byte of each code is the "continuation code" for the manufacturer.

| Manufacturer | Code | Manufacturer | Code |
|--------------------|--------|--------------|--------|
| AMD | 0x0001 | Eurotech | 0x009B |
| Fujitsu | 0x0004 | Exel | 0x009E |
| Hitachi | 0x0007 | Hyundai | 0x00A7 |
| Intersil | 0x000B | Oki | 0x00AE |
| Motorola | 0x000E | Sharp | 0x00B0 |
| NEC | 0x0010 | SST | 0x00BF |
| Signetics | 0x0015 | Macronix | 0x00C2 |
| Xicor | 0x0019 | Samsung | 0x00CE |
| Mitsubishi | 0x001C | ISSI | 0x00D5 |
| Atmel | 0x001E | Winbond | 0x00DA |
| Atmel | 0x001F | | |
| ST/SGS | 0x0020 | | |
| Waferscale | 0x0023 | Eon | 0x011C |
| Microchip/GI | 0x0029 | Seiko-Epson | 0x013E |
| Catalyst/Xicor | 0x0031 | | |
| Cypress | 0x0034 | | |
| Harris | 0x0086 | | |
| Intel | 0x0089 | | |
| Fairchild/National | 0x008F | | |
| Seeq | 0x0094 | | |
| Texas Instruments | 0x0097 | | |
| Toshiba | 0x0098 | | |

Device Codes

This is a (probably incomplete) list of 3-byte device codes for EPROMs. The high 2 bytes are the manufacturer code, from the above table. These codes are read from the EPROM with the ID command.

| Code | Device | Code | Device | Code | Device |
|-------------------|--------------|-------------------|--------------|---------------------------|-----------|
| AMD | | Atmel | | Cypress | |
| 0x000104 | 27256 | 0x001E0D | AT27C512R | 0x00341A | CY27C128 |
| 0x000107 | 2732B | and | AT27BV512 | 0x003421 | CY27C256 |
| 0x000108 | 2764A | and | AT27LV512A | 0x00341F | CY27C512 |
| 0x000110 | 27C256 | 0x001E10 | AT27HC641R | | |
| 0x000115 | 27C64 | and | AT27HC642R | Intel | |
| 0x000116 | 27C128 | 0x001E1F | AT27HC641 | 0x008901 | 2732A |
| 0x000185 | 27512 | and | AT27HC642 | 0x008902 | 2764 |
| 0x000186 | 2716B | 0x001E8C | AT27C256R | 0x008904 | 27256 |
| 0x000189 | 27128A | and | AT27BV256 | 0x008907 | 27C64 |
| 0x000191 | 27C512 | and | AT27LV256 | 0x008908 | 2764A |
| | | 0x001F0D | AT27C512 | 0x00890D | 27512 |
| | | 0x001F10 | AT27HC641 | 0x008983 | 27128 |
| Fujitsu | | and | AT27HC642 | 0x008988 | 27256 |
| 0x000402 | MBM27256 | 0x001F91 | AT27HC64 | 0x008989 | M27128A |
| 0x000462 | MBM27C256A | 0x001F94 | AT27HC256 | 0x00898C | 27256 |
| 0x000472 | MBM27C256A | | | 0x00898D | 27C256 |
| 0x0004A1 | MBM27C128 | | | 0x0089FC | 27C128 |
| 0x0004E3 | MBM27C512 | ST/SGS | | 0x0089FD | 27C512 |
| | | 0x002000 | M27128A | | |
| | | 0x002004 | M27256 | | |
| Hitachi | | 0x002008 | M27C64A | Fairchild/National | |
| 0x00070D | HN27128A | 0x00200D | M27512 | 0x008F01 | 27C32B |
| 0x000710 | HN27256 | 0x002013 | M2732A-Fast | 0x008F04 | 27C256 |
| 0x000731 | HN27C256A | 0x00203D | M27C512 | and | 27C256B |
| and | HN27C256H | 0x002080 | M87C257 | and | NM87C257 |
| 0x000737 | HN27C64 | 0x002089 | M27128A | 0x008F80 | 27C16B |
| 0x000794 | HN27512 | 0x00208D | M27C256B | 0x008F83 | 27C128 |
| 0x0007B0 | HN27C256 | | | 0x008F85 | 27C512 |
| | | | | and | 27C512A |
| NEC | | Waferscale | | | |
| 0x001004 | uPD27256 | 0x0023A6 | WS57C128F | 0x008FC2 | 27C64 |
| 0x001025 | uPD27C512 | 0x0023A8 | WS27C128F | 0x008FC4 | 27C256 |
| 0x001064 | uPD27C256A | and | WS27C64F (!) | 0x008FC5 | 27C512 |
| 0x0010A4 | uPD27C256 | 0x0023AA | WS27C512F | | |
| 0x0010C4 | uPD27256A | and | WS57C512F | Seeq | |
| | | 0x0023C0 | WS27C256L | 0x009440 | M2764 |
| | | 0x0023C0 | WS27C256F | 0x0094C1 | M27128 |
| Signetics | | Microchip | | 0x0094C2 | M27C256 |
| 0x00150B | 27C64A | 0x002902 | 27C64 | | |
| 0x00151D | 27C512 | 0x002904 | 27256 | TI | |
| 0x00158C | 27C256 | 0x00290D | 27C512A and | 0x009704 | TMS27256 |
| | | | 27LV512 | and | TMS27C256 |
| Mitsubishi | | | | and | SMJ27C256 |
| 0x001C01 | M5M27C256 | 0x002983 | 27C128 | | |
| 0x001C04 | M5L27256 and | 0x00298C | 27C256 and | 0x009707 | TMS27C64 |
| | M5M27256 | | 27LV256 | 0x009783 | TMS27C128 |
| 0x001C07 | M5M27C512A | 0x002994 | 27HC256 | 0x009785 | TMS27C512 |
| 0x001C0D | M5L27512 | | | | |
| | | | | | |

| Code | Device | Code | Device | Code | Device |
|--------------------|-----------|-----------------|--------------|-------------|---------------|
| Toshiba | | Sharp | | ISSI | |
| 0x009804 | TC57256 | 0x00B0C0 | 27C256L | 0x00D510 | IS27C256 and |
| 0x009813 | TMM27128 | 0x00B0C2 | LH57512 | | IS27DV256 and |
| 0x009815 | TMM27512 | | | | IS27HC256 and |
| 0x009845 | TC57H256 | SST | | | IS27LV256 |
| 0x009852 | TMM2764A | 0x00BFA3 | 27SF256 | 0x00D591 | IS27C512 and |
| 0x009854 | TMM25256A | 0x00BFA4 | 27SF512 | | IS27HC512 |
| and | TMM27256A | | | | |
| 0x009885 | TC57512A | Macronix | | Eon | |
| 0x0098C4 | TC57256A | 0x00C210 | MX27C256 and | 0x011C512 | EN27C512 |
| 0x0098D3 | TMM27128A | | MX27L256 | | |
| | | 0x00C291 | MX27C512 and | | |
| Hyundai | | | MX27L512 | | |
| 0x00A708 | HY27C64 | | | | |
| | | | | | |
| ST/Eurotech | | | | | |
| 0x009B04 | ST27C256 | | | | |
| 0x009B08 | 27C64A | | | | |
| | | | | | |