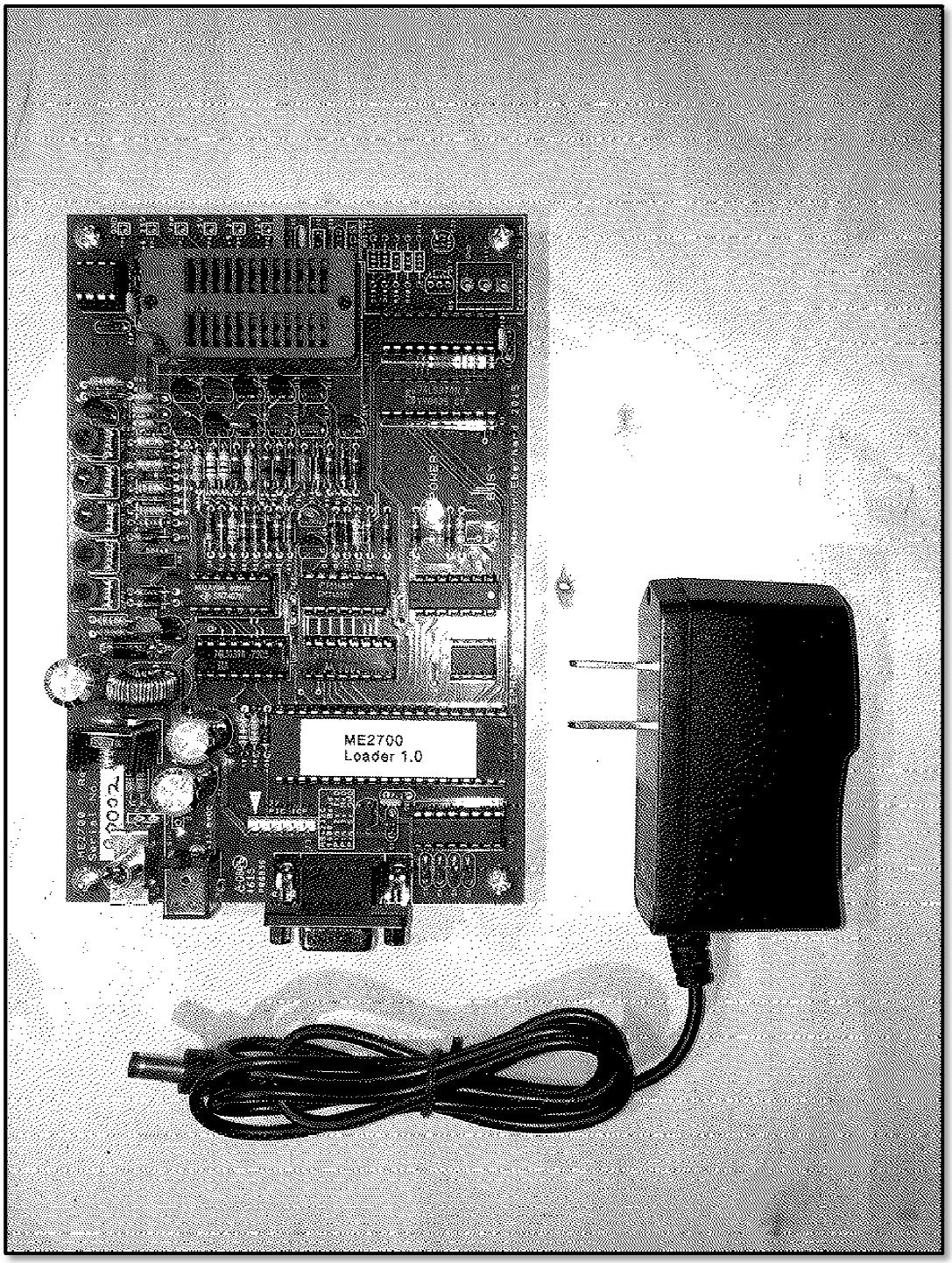
ME2700

Martin Eberhard's Orphan EPROM Programmer

User's Manual



ME2700

Martin Eberhard's Orphan EPROM Programmer

Rev B and C PC Boards, Rev 1.02 & 1.03 Firmware

The ME2700 is designed to program many of the "orphan" 24-pin 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 "universal" EPROM programmers. (The ME2700 can also program all of the "standard" 24-pin EPROMs.) In particular, the ME2700 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 Vpp=-40V.

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 ME2700.

The ME2700 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 ME2700 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 ME2700 is the Custom EPROM Editor, which lets you define and save (in onboard EEPROM) up to four cutom EPROM specifications. You can specify the functions for pins 18 through 22, 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 ME2700.

The ME2700 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 ME2700.

One thing the ME2700 does *not* do is attempt to automatically identify the EPROM that is inserted. Some 24-pin EPROMs will divulge their ID if you apply +12V to address pin A9. A few others have different (non-standard) ID methods. But most EPROMs do not include any provision for reading their ID, and applying +12V to their address pin A9 will actually damage the EPROM.

The ME2700's universal 12V AC adapter 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.

The only differences between the Rev B and Rev C PC boards are the replacement of transistor Q1 (which had become obsolete) and some minor tweeks to the layout (just because I was revising the PC board anyway). As part of these tweeks, the gates within U8 were swapped a bit, and a few components moved.

-Martin Eberhard
21 October 2018

ME2700 Revision History

PCB	Firmware	Manual	Date	Change Notes
B	1.00	Prelim	2 Jan 2016	First complete version
B	1.01	1.01	20 Jan 2016	First released version
B	1.02	1.02	2 Feb 2016	Typo fixes in manual. Add ASCII to BD output. Default to EPROM size for UI & US commands.
B	1.02 & 1.03	1.02A	1 Jun 2016	Typo corrections
B	1.02 & 1.03	1.02B	1 Jun 2016	Typo correction in BOM
B	1.02 & 1.03	1.03	15 Feb 2018	Support TTC004B as a substitute for 2SC6043 in Q1
B&C	1.02 & 1.03	1.03A	21 Oct 2018	Support Rev C PC Board. Add rework for pull-down resistor on the base of Q9 (for both rev B and rev C boards), to tolerate a somewhat leaky 2N3904 transistor in Q9.

Contents

Section 1. Getting Started 1
Section 2. ME2700 Programmer Usage 3
2.1 Worldwide Operation 3
2.2 Serial Port Connector Pinout 3
2.3 LEDs 3
2.4 Manual Voltage Adjustment 3
2.5 Intersil Option 4
2.6 Power Supply Voltage Checking 5
2.7 Programming an EPROM from a File 5
2.8 Reading an EPROM into a File 5
2.9 Copying an EPROM 5
2.10 File Address Offset 6
2.11 Buffer Address Offset 7
2.12 Data Invert 8
Section 3. ME2700 Commands 9
3.1 EPROM Commands 9
3.2 File Transfer Commands 10
3.3 Buffer Commands 12
3.4 Miscellaneous Commands 13
3.5 Diagnostic Commands 13
Section 4. Custom EPROM Editor 16
4.1 CEE General Commands 16
4.2 CEE Pin Assignment Commands 17
4.3 CEE Programming Parameter Commands 18
Section 5. Programming Algorithms 21
5.1 Simple Programming Algorithm 22
5.2 Fast Programming Algorithms 23
Section 6. ME2700 Theory of Operation 25
6.1 Architecture 25
6.2 Microcontroller 25
6.3 Logic Supplies 25
6.4 +6.2V Supply 25
6.5 -5V Supply 25
6.6 Microcontroller-Controlled High-Voltage Supply 26
6.7 EPROM Digital Pin Interface 26
Section 7. Downloading Firmware via the Serial Port 27
7.1 Firmware Download Instructions 27
7.2 Intel Hex File Format for Firmware Downloads 28

Section 8. ME2700 Programmer Assembly 30
8.1 Printed Circuit Board Assembly 31
Section 9. Checkout and Adjustment 35
9.1 Basic PCBA Checkout 35
9.2 Microcontroller Bring-Up 37
9.3 Microcontroller-Assisted Checkout and Adjustment 38
Section 10. Functional Testing 43
10.1 Basic Buffer Operations and File Transfer 43
10.2 EPROM Reading and Programming 47
Section 11. Printed Circuit Board 51
11.1 Bill of Materials 51
11.2 Rev B and C PCBA Component Placement 53
11.3 Rev B and C PCBA Schematics 55
Appendix A. Supported EPROMs and EEPROMs 59
EPROMs and EEPROMs Sorted by Device Type 59
EPROMs and EEPROMs Sorted by Manufacturer 61

Section 1. Getting Started

This section is a quick overview of basic ME2700 operation. Follow these steps to load a hex file into the ME2700's buffer, and then program the buffer contents into an EPROM.

1. Set up your computer's RS232 port and terminal program for 9600 baud, no handshaking. (Use a USB-to-RS232 dongle if your computer does not have an RS-232 port.)

2. Connect the ME2700 to your computer's 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:

```
*====================================*
*               ME2700               *
*====================================*
*       Orphan EPROM Programmer      *
*         By Martin Eberhard         *
*       Firmware Version  1.03       *
*====================================*

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

   Note that the "Current Device Type" will be the same as it was the last time you used the ME2700.

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 18
Current Device Type is 18: TMS2532
>
```

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

```
>ED
Type 18: TMS2532, size: 4096 x 8
       -----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 |-  -CS/-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
       -----------
Vpp during read: 5V
Programming pulse width: 50 mS
Programming cycles: 1
>
```

4. Send an Intel hex file of the EPROM image from your terminal program to the ME2700. The ME2700 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 ME2700 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.

6. Program the EPROM with the “EP” command.

```
>EP
Make sure the correct Device Type is selected, and that the EPROM
is inserted correctly, with pin 1 closest to the socket handle.\r"
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 ME2700 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 ME2700 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. ME2700 Programmer Usage

The previous section walked you through the most basic ME2700 Programmer operation. Here are some specifications, and the procedures for some more common EPROM operations.

2.1 Worldwide Operation

The ME2700's AC adaptor is a world-wide adapter, suited for 50 Hz or 60 Hz, 90VAC to 260V. The prongs are USA-type prongs, but simple adapters to other prong types are readily available.

2.2 Serial Port Connector Pinout

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

Female DA-9 Pin	Signal
2	Data Out (out of the ME2700)
3	Data In (in to the ME2700)
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 ME2700)	2
3	Data In (in to the ME2700)	3
5	Ground	5

2.3 LEDs

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

The blue 'POWER' LED, and indicates that the AC adapter is energized and the power switch is on.

The red LED indicates that the ME2700 is 'BUSY'. When lit, the EPROM socket is energized, and an EPROM should not be removed or inserted.

2.4 Manual Voltage Adjustment

The five voltage adjustments on the ME2700 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 TP3. Type "AVPP 1", and then adjust VR1 until TP3 measures 12.7V. Then do the same for the other three adjustments.

AVPP	Adjustment	Nominal TP3 Voltage
0	-	About 11.5V
1	VR1	12.70V ± 0.05V
2	VR2	13.15V ± 0.05V
3	VR3	21.00V ± 0.05V
4	VR4	25.20V ± 0.05V

Adjust the programming Vcc by measuring the voltage at TP2 and adjusting VR5. The nominal voltage for this adjustment is 6.20V ± 0.02V.

A few of the more obscure EPROMs supported by the ME2700 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.)

If the programming Vcc requires adjustment, measure the voltage at TP2, and adjust the voltage with VR5.

2.5 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 ME2700 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 ME2700. (So long as the shunt is removed from J4, the inclusion of these components will not affect programming other EPROM Device Types.)

√	Qty	Location	Value	Digikey Part Number
	1	R39	6.8 1/4W Resistor	S6.8HCT-ND
	1	R41	330 ohm 1/4W Resistor	330QBK-ND
	1	R40	1K ohm 1/4W Resistor	CF14JT1K00CT-ND
	2	D13, D14	1N4004 Diode	1N4004-TPMSCT-ND
	1	C22	0.1 uF 50V Chip Capacitor	478-4855-ND
	1	C23	33 uF 50V Electrolytic Capacitor	P5180-ND
	1	Q17	2N6520 Transistor	2N6520TACT-ND
	1	Q16	BD179G Transistor	BD179GOS-ND
	1	J4	2-pin 0.1” Header	A19423-ND
	1	J4	2-pin shunt	3M9580-ND
	1	J5	3-pin male 0.156” connector	WM4621-ND

When programming an Intersil IM6654 or IM6658 EPROM, short the pins of J4 with a shunt, and connect an external power supply to J5:

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

Select the EPROM Device Type (03 for the IM6654 or Type 25 for the IM6658), and program the EPROM.

Important: Remove the shunt from J4 when programming any other type of EPROM.

2.6 Power Supply Voltage Checking

The ME2700 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 - either because it is defective or if the wrong EPROM Device Type is selected. Although the ME2700 will shut down quickly when overcurrent is detected this way, the EPROM may still be damaged, and should probably be discarded.

Vpp will be too high only due to some fault with the ME2700 itself.

2.7 Programming an EPROM from a File

Here is how you program an EPROM from a file, using the (default) automatic file address offset mode.

```
{Power-on}
>ET {correct EPROM Device Type}
{insert blank EPROM in the socket}
>   {send S-Record or Intel Hex file to the ME2700}
>EP
Make sure the correct Device Type is selected, and that the EPROM
is inserted correctly, with pin 1 closest to the socket handle.\r"
Ready to program (Y/N)? Y
Programming -
Verifying
EPROM matches buffer
>
```

2.8 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.

```
{Power-on}
>ET {correct EPROM Device Type}
{insert programmed EPROM in the socket}
>ER
Success
>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.

2.9 Copying an EPROM

You can read an EPROM, and then write it into any number of blank EPROMs.

```
{Power-on}
>ET {correct EPROM Device Type}
{insert source EPROM in the socket}
>ER
Success
{insert blank EPROM in the socket}
>EP
Make sure the correct Device Type is selected, and that the EPROM
is inserted correctly, with pin 1 closest to the socket handle.\r"
Ready to program (Y/N)? Y
Programming -
Verifying
```

```
EPROM matches buffer
{insert another blank EPROM in the socket}
>EP
Make sure the correct Device Type is selected, and that the EPROM
is inserted correctly, with pin 1 closest to the socket handle.\r"
Ready to program (Y/N)? Y
Programming -
Verifying
EPROM matches buffer
>
```

2.10 File Address Offset

Simple Intel Hex files and Motorola S-record files have 2-byte (4 hex digit) addresses. Thus, the address in a hex file can be thought of as having 2 components: without any file address offset, the high byte (first 2 hex digits) can be 00 through 1F (depending on EPROM size), and the low byte (second 2 hex digits) represents the address within one page of the EPROM.

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 be 00-1F. Instead, these bytes will be the high byte of the intended target address.

You can tell the ME2700 what the file address offset is, using the F**AO** (File Address Offset) command. If you specify automatic File Address Offset mode (by typing the F**AO** 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 ME2700, the File Address Offset is subtracted from the high address bytes for the data in the file. Any data whose address high byte minus the File Address Offset is not between 00 and 1F will not be loaded into the buffer.

Note that it is possible for some of the data within a single record will get loaded into the buffer, and some of it will not. (This will happen if the record spans an EPROM boundary.) Such records are not counted as loaded records in the reporting at the end of the file, though the data that fit into the buffer does get written into the buffer.

If you have a hex file that spans several EPROMs, you can program each EPROM sequentially by first setting the File Address Offset for one of the EPROMs, then clearing the buffer, and then sending the whole hex file, and then programming the EPROM. Repeat this procedure for each EPROM - setting the File Address Offset appropriately. Each time, only data whose address high byte minus the File Address Offset equals 00-1F will get loaded into the buffer - the others will be ignored.

Similarly, 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 F**AO** 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.

2.11 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 (as large as 8K 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 like this:

```
>ET 4
Current Device Type is 04: 2708
>BAO 0
Buffer Address Offset: 00h
{insert first 2708 EPROM}
>EP
Make sure the correct Device Type is selected, and that the EPROM
is inserted correctly, with pin 1 closest to the socket handle.\r"
Ready to program (Y/N)? Y
Programming -
Verifying
EPROM matches buffer
>BAO 4
Buffer Address Offset: 04h
{insert second 2708 EPROM}
>EP
Make sure the correct Device Type is selected, and that the EPROM
is inserted correctly, with pin 1 closest to the socket handle.\r"
Ready to program (Y/N)? Y
Programming -
Verifying
EPROM matches buffer
>BAO 8
Buffer Address Offset: 08h
{insert third 2708 EPROM}
>EP
Make sure the correct Device Type is selected, and that the EPROM
is inserted correctly, with pin 1 closest to the socket handle.\r"
Ready to program (Y/N)? Y
Programming -
Verifying
EPROM matches buffer
>BAO C
Buffer Address Offset: 0Ch
{insert fourth 2708 EPROM}
>EP
Make sure the correct Device Type is selected, and that the EPROM
is inserted correctly, with pin 1 closest to the socket handle.\r"
Ready to program (Y/N)? Y
Programming -
Verifying
EPROM matches buffer
```

Similarly, you could read a larger EPROM and program its data into several smaller EPROMs.

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 18
Current Device Type is 18: TMS2532
{insert blank TMS2532 EPROM}
>BAO 0
Buffer Address Offset: 00h
>EP
Make sure the correct Device Type is selected, and that the EPROM
is inserted correctly, with pin 1 closest to the socket handle.\r"
Ready to program (Y/N)? Y
Programming -
Verifying
EPROM matches buffer
```

2.12 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.

Section 3. ME2700 Commands

3.1 EPROM Commands

The EPROM commands generally deal with the EPROM, and the 8K-byte buffer in the ME2700. 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>=2000 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 **ME2700 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, such as the Memory Merchant MM65K16S S-100 memory board.) 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 (data=FF). 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".

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

ED 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.

Note that the list of supported EPROMs is representative. If your EPROM is not listed here, then compare its specifications to those displayed with the ED command, to verify compatibility.

>**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, and their indices, for use with the ED, EE, and ET commands.

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

EP programs <cnt> bytes of the EPROM with data from the buffer, 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 ME2700 (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.

?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 ME2700, the File Address Offset will be taken from the high address byte of the first received record. When uploading a file from the ME2700, the File Address Offset will be 00.

>F**AO <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 section 2.10, **File Address Offset**, 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 an Intel Hex 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 1FFF will be loaded into the buffer. Any data whose target address is outside this range will not be loaded.

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

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 1FFF will be loaded into the buffer. Any data whose target address is outside this range will not be loaded.

An S5 (record count) record will compare the ME2700'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 checksum computed by the ME2700, 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**

FB 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 ME2700.

>**ECHO {0/1} Set Terminal Echo**

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

>**RESET Reset ME2700 Programmer**

RESET is just like power-cycling the ME2700.

>**? Help**

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

>**?N General notes on ME2700**

This command prints a page of general notes about the ME2700.

>**?L View Firmware Loader Notes**

?L displays notes on loading new firmware into the ME2700 via the serial port. Firmware loading is discussed in a later section.

^S Pause Serial Port Output

Control-S (XOFF) tells the ME2700 to stop sending data. Any subsequent character (including XON, which is control-Q) will re-enable the ME2700 output, and that character will be discarded by the ME2700.

3.5 Diagnostic Commands

These commands are intended only for diagnosing the ME2700, 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 four Vpp voltages, using a voltmeter at TP3. 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 TP3 Voltage
0	-	About 11.5V
1	VR1	12.70V ±0.05V
2	VR2	13.15V ±0.05V
3	VR3	21.00V ±0.05V
4	VR4	25.20V ±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).

>**TCS {0/1} Test CS Pin**

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

>**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.

>**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.

>**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 ME2700 (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 “Fast” 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 ME2700 (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**

If the selected EPROM Device Type has a VPP pin, then this command tests it. 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 four EPROMs displayed by the EP command are custom EPROMs. If any of these EPROMs has not been defined, 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 EE <n> at the main prompt, where <n> is the EPROM Device Type number for one of the four 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 QU prompt. 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>ED <Device Type> Display EPROM Specs

ED 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>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 “Fast” 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

Returns to the main command prompt.

4.2 CEE Pin Assignment Commands

Pins 18 through 22 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 Select
- PGM may be assigned to the same pin as either Output Enable or Chip Select

EEnn>A9 <pp> Assign A9 Signal to Pin

Signal A9 is assigned to the specified pin, where <pp> may be 0 or between 18 and 22. 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 or between 18 and 22. 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 or between 18 and 22. 0 means unassign A11. Note that if A11 is assigned to a pin then any stuck-high pin assignment will become unassigned.

EEnn>A12 <pp> Assign A12 Signal to Pin

Signal A12 is assigned to the specified pin, where <pp> may be 0 or between 18 and 22. 0 means unassign A12. Note that if A12 is assigned to a pin then any Address Strobe pin assignment will become unassigned.

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

The (active-low) Address Strobe Signal is assigned to the specified pin, where <pp> may be 0 or between 18 and 22. 0 means unassign Address Strobe. Note that if Address Strobe is assigned to a pin then any A12 pin assignment will become unassigned.

EEnn>CS <pp> Assign Chip Select Signal to Pin

The Chip Select Signal is assigned to the specified pin, where <pp> may be 0 or between 18 and 22. 0 means unassign Chip Select. Note that Chip Select may share a pin with either Vpp or PGM.

EEnn>CSP {0/1} Assign polarity to Chip Select Signal

CSP defines the polarity of the Chip Select 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 or between 18 and 22. 0 means unassign Stuck-High pin. Note that if a pin is assigned to be stuck-high then any A11 pin assignment will become unassigned. 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 or between 18 and 22. 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 or between 18 and 22. 0 means unassign Program. Note that PGM may share a pin with either Chip Select 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, 18, 20, 0r 21. 0 means unassign Vpp. Note that Vpp may share a pin with either Chip Select or Output Enable.

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

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

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

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

4.3 CEE Programming Parameter Commands

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

BCK 1 enables a blank-check before programming, BCK 0 disables it. Disable blank check for EEPROMs, and for any EPROM Device Type that erases to an indeterminate (or non-FF) state.

EEnn>FF1 {0/1} Write FF during Programming Pass 1

FF1 1 causes the first programming pass through the EPROM to write FFh data to every byte. (This is necessary to erase some EEPROMs.) FF1 0 disables this feature.

EEnn>FN1 <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>FN2 <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>FP1 <p> Define Programming Pass 1

Selects one of 5 possible “fast” programming algorithms for pass 1:

<p>	Algorithm
0	Program each byte until it matches, then program it an additional n times
1	Program each byte until it matches, then program it with one additional pulse that is n * PPW[1] long
2	Program each byte until it matches (P times), then program it an additional nP times
3	Program each byte until it matches (P times), then program it one additional time with a pulse that is nP * PPW[1] long
4	Program each byte n times
5	Program each byte with one pulse that is n * PPW[1] long

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

EEnn>FP2 <p> Define Programming Pass 2

Selects one of 5 possible “fast” programming algorithms for pass 2. (See table for FP1.)

EEnn>OEV {0/1} Set +12V to Output Enable Pin 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 pin 19 or pin 20.

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

If any Fast 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>PPS <nn> Define Programming Pulse Separation

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

EEnn>PPW <nn> Define Programming Pulse Width

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

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

PTU 0 means that values given with the PPW and PPS 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 Fast 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) during programming:

<v>	Vcc during Programming
0	0V
1	5V
2	6.2V
3	12V

EEnn>VPP <v> Define Vpp During Programming

Defines the voltage on the Vpp pin during programming:

<v>	Vpp if pin 18	Vpp if pin 20 or 21
0	Switcher off	Switcher off
1	13.45V	12.70V ±0.05V
2	13.85V	13.15V ±0.05V
3	21.70V	21.00V ±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. Programming Algorithms

The ME2700 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:

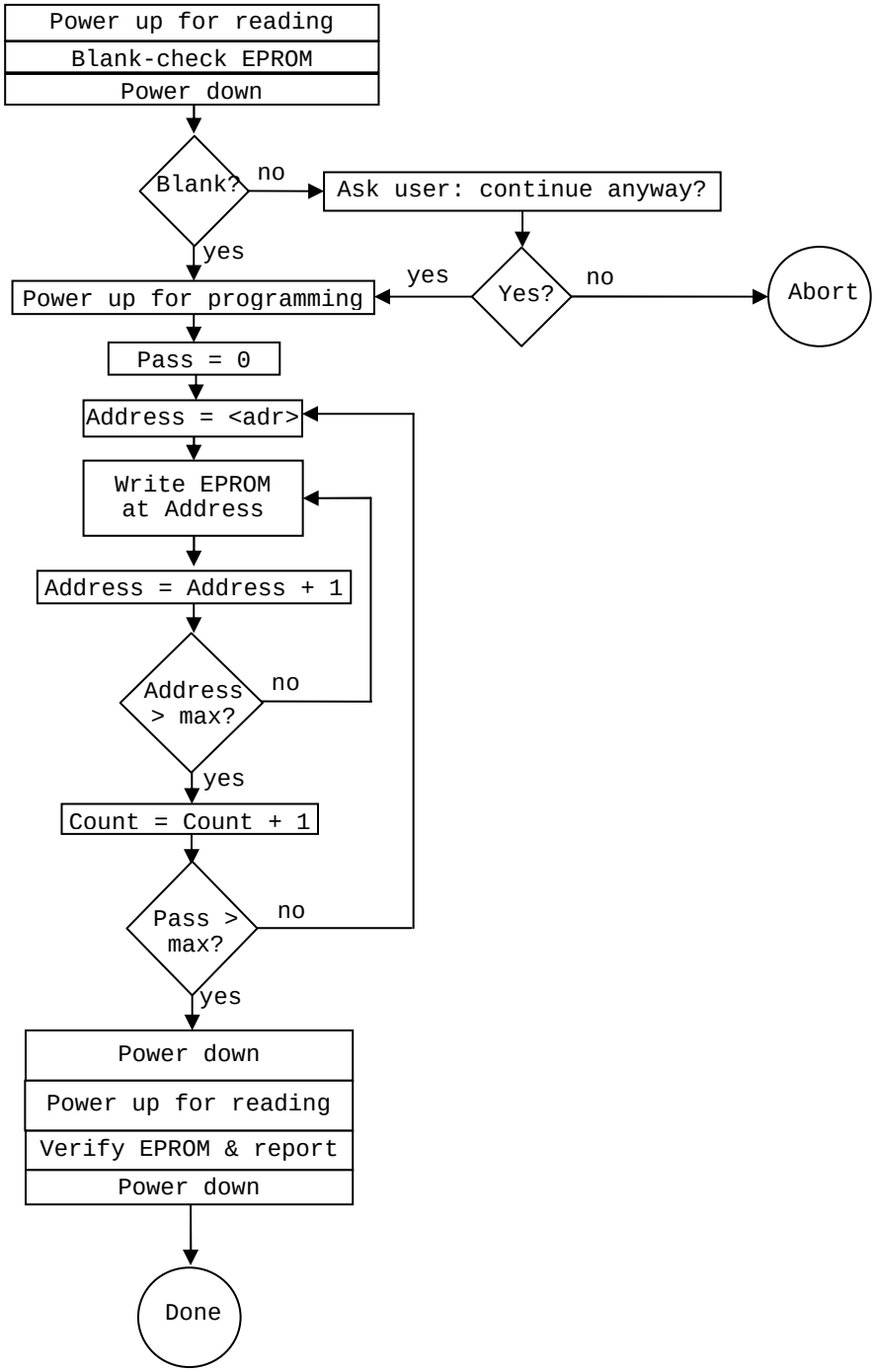
- The EPROM’s pinout
- Any stuck-high pins
- The programming voltage, Vpp
- Special programming voltage requirement for Vcc
- Special programming voltage requirement Output Enable
- The programming pulse width
- On which signal the programming pulse is applied
- The minimum separation between programming pulses
- The programming algorithm (see below)
- Whether or not to blank-check the EPROM before programming
- Whether or not to write FF 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)

During the entire programming cycle:

- The Output Enable pin is disabled, if it exists (except during verify)
- The Chip Select pin is enabled, if it exists
- The Vcc pin is raised to an elevated voltage, if required
- The Output Enable pin is raised to an elevated voltage, if required
- The Vpp pin is driven to the required 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 Fast Programming Algorithms

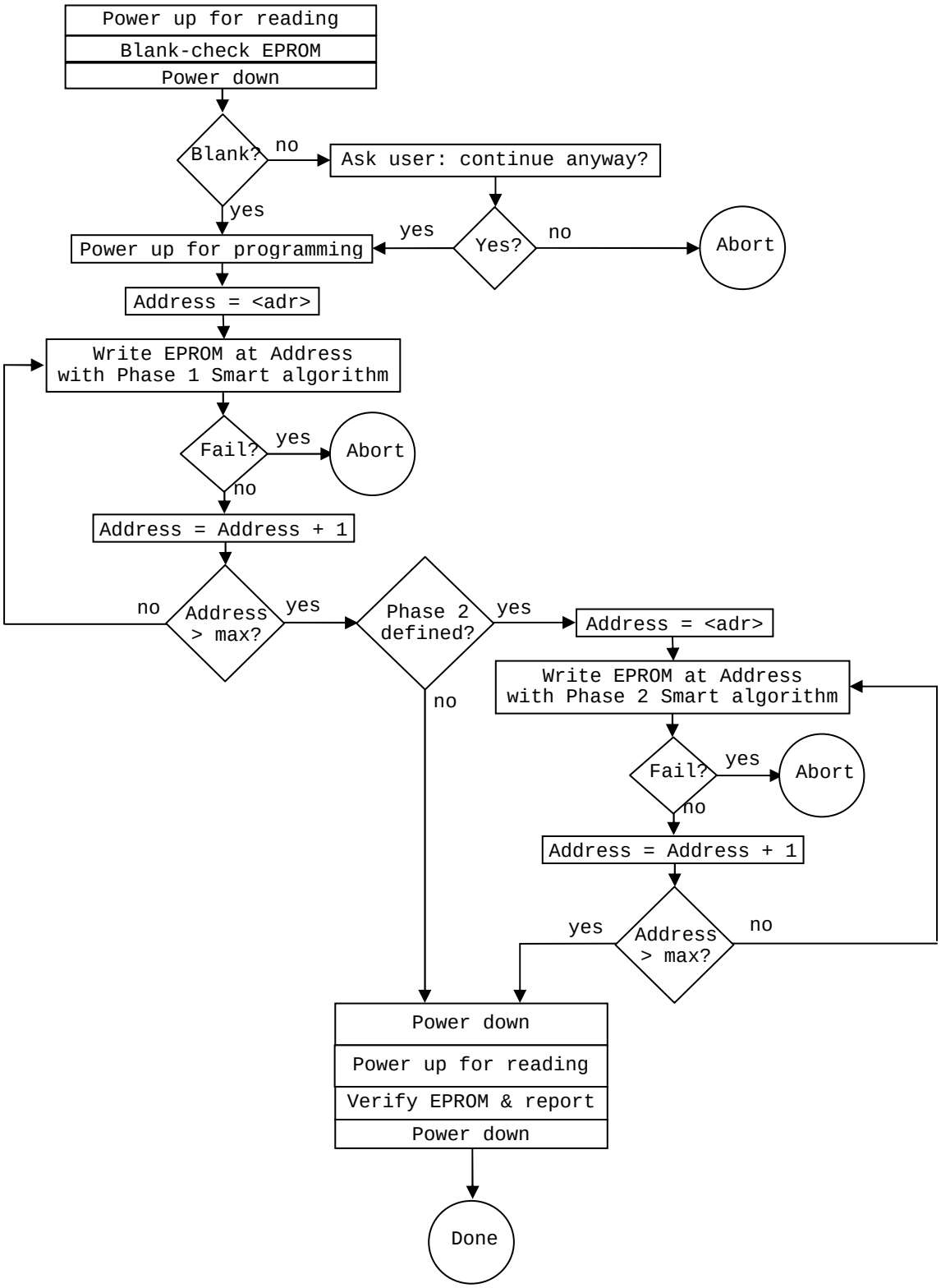
The ME2700's Fast Programming is a flexible system, designed to allow implementation of all (or at least most) Fast/Quick/Express/Rapid/Turbo/Whatever programming algorithms specified by the various EPROM manufacturers. All of these Fast Programming algorithms have either one or two phases. Each phase may be defined independently, each comprising one pass through the specified address range of the EPROM.

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

0. Program each byte until it matches, then program it an additional n times, where n is a value between 0 and 7
1. Program each byte until it matches, then program it with one additional pulse that is n * PPW long, where n is a value between 0 and 7
2. Program each byte until it matches (P times), then program it an additional nP times, where n is a value between 0 and 7, and where the maximum P is specified (and is less than 256)
3. Program each byte until it matches (P times), then program it one additional time with a pulse that is nP * PPW long, where n is a value between 0 and 7, and where the maximum P is specified (and is less than 256)
4. Program each byte n times, where n is a value between 0 and 7
5. Program each byte with one pulse that is n * PPW long, where n is a value between 0 and 7

Additionally, the first pass may be specified to write 0FFh to every byte, instead of writing buffer data. This is for erasing some type of EEPROMs, such as the Intel 2816A.

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


Power up for reading
Blank-check EPROM
Power down
Blank?
no
Ask user: continue anyway?
yes
Yes?
yes
no
Abort
Power up for programming
Address = <adr>
Write EPROM at Address
with Phase 1 Smart algorithm
Fail?
yes
Abort
no
Address = Address + 1
Address
> max?
no
yes
Phase 2
defined?
yes
Address = <adr>
Write EPROM at Address
with Phase 2 Smart algorithm
no
Fail?
yes
Abort
no
Address = Address + 1
Address
> max?
yes
no
Power down
Power up for reading
Verify EPROM & report
Power down
Done

Section 6. ME2700 Theory of Operation

6.1 Architecture

The following is a very brief description of the ME2700 Programmer's circuit operation.

6.2 Microcontroller

At the heart of the ME2700 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 ME2700 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.
- An external 8K-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

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

6.3 Logic Supplies

V2 regulates the Wall adapter's +12V down to about +5.15V, used for some of the logic, as well as for powering the EPROM. It is deliberately set a bit above the normal 5.0V, because marginally programmed bits in this class of EPROMs are more likely to fail when Vcc is high. It is normal for this regulator to get pretty warm during operation, particularly during programming. (This is the only component that should ever get warm.)

V1 regulates the +5.15V 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 +6.2V Supply

V3 regulates the Wall adapter's +12V down to 6.2V, adjustable with VR5. This power supply is used to provide special Vcc voltage for some EPROMs during programming. It is adjustable because some EPROMs need slightly different voltages, such as 5.5V or 6.5V. When these EPROMs are to be programmed, the user can adjust this supply as needed.

6.5 -5V Supply

U11, together with C17 and 18, comprise a switched-capacitor voltage inverter that produces -5V from the +5.15V 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 C13, using diode D2.

The PIC's internal voltage comparator compares the output from one of the four voltage dividers (R7-R14 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 ME2700.

Resistor R5, 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 it's 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 EPROM pins: 18, 20, 21. If Vpp is directed to pin 18, 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 data pins are buffered with a 74LVC245 bidirectional buffer. Address pins 0 through 7 are buffered with a 74LS374 8-bit latch. These two buffer chips are driven by the same 8 data pins of the PIC microcontroller, its Port D. The latch signal for the 74LS374 is also the signal that drives the BUSY light.

Pin 23 (address pin A8) and pin 22 (usually address pin A9) are buffered by 2 inverters from the 74ACT04 hex inverter, with firmware compensating for this inversion.

All of the other digital signals (A10-A12, CS, -OE, PGM, etc.) are selectable for pins 18 through 21, as required for the particular EPROM selected. These pins also may be driven to either negative voltages or higher voltages, as needed for power supplies, programming voltages, etc.

Section 7. Downloading Firmware via the Serial Port

You can load new ME2700 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 ME2700.

The ME2700 firmware is divided into two components: the ME2700 Loader Kernel (the Loader, which cannot be downloaded via the serial port) and the ME2700 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 ME2700 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.

To enter the Loader, type capital L a few times immediately after you turn on the ME2700, or immediately after issuing a Reset command. You will see the Loader message, instead of the ME2700 sign-on banner:

```
ME2700 Loader 1.0
```

When you see this message, the ME2700 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.

The PIC microcontroller has insufficient RAM to load and validate the entire Programming Firmware before writing it to FLASH. Instead, the Loader writes to FLASH whenever it gets a complete 32-word 'row' of data (where a word is 2 bytes in the hex file). This means that a failed Programming Firmware load will probably corrupt the Programming Firmware in FLASH memory. (However, the Loader itself is hardware-protected against being overwritten.)

Before you send the hex file, make sure your terminal program has XOFF/XON handshaking enabled. Handshaking is required so that the Loader can pause the file transmission from time to time, to write the received data into FLASH. Otherwise, data will be lost, and the Programming Firmware will be corrupted.

Once your terminal program is set up correctly, send the hex file (typically, me2700.hex) to the ME2700. 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)
?Ver	Flash read-back verify error

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 loaded firmware is most likely corrupted.

If you got your new firmware file from me, then I will have included a

comment that tells you how many records should have been loaded. If the reported number of loaded records does not match the number that I provided, then the firmware load was probably not successful, and the loaded firmware is most likely corrupted.

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 ME2700.

If the load fails for any reason, you can try again immediately after the failed load, when Loader message is printed. Or, you can type capital L immediately after powering on the ME2700 to invoke the Loader to try loading 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 PIC's FLASH memory is organized in 64-byte ‘rows’. The Loader assumes that each Intel Hex record fits completely within one FLASH row. However, one FLASH row may be (and probably will be) made up of several hex records. (In other words, no record is allowed to have data that is split between two 64-byte FLASH rows.) Hex files generated by MPASM meet this requirement.

If the hex file has data for only part of a FLASH row, then only the data supplied in the Hex file will be changed - the other bytes in that FLASH row will remain unchanged, because the Loader first reads the old FLASH data from each row into its RAM buffer, filling in the missing data for that row.

After each row is written, the Loader verifies the write by reading it back and comparing it to the row data in its RAM buffer.

The Loader checks the checksum for all records, including ignored records.

The following conditions will generate an error message, and increment error count that is reported after the end of the file:

1) **Checksum Error** (The checksum of the record was incorrect.)

2) **Bad Hex Error** (something besides ‘0’-‘9’ or ‘A’-‘F’ when expecting hex)

3) **Verify Error** (FLASH read-back did not match the RAM buffer data)

The Loader actually writes to FLASH when a new hex record addresses FLASH memory in a different FLASH row than the previous record, or when a type 01 record is received. This means that FLASH write-verification occurs after the address and record-type fields of the next hex data record have been received, and before its data bytes are received. And this (in turn) means that if a verify fails, then the verify error message will be printed in the middle of the next hex record, and will refer to the previous record(s).

Erasing and writing one FLASH row takes significantly longer than a character time at 9600 baud, and the FLASH write cannot be interrupted. For this reason, your terminal program must respect XOFF/XON handshaking: the Loader will issue an XOFF prior to programming a FLASH row, and then will wait for your terminal program to respond to the XOFF, accepting up to about 127 more characters after sending the XOFF. When the Loader receives no characters for 3.2 mS (3 character times at 9600 baud) after sending an XOFF, it assumes that your terminal program has paused transmission, and will program the FLASH row. The Loader will send an XON when the FLASH row programming is complete. The Loader will not receive any characters from its serial port while it is busy programming FLASH - such characters will be lost.

The hex file must end with a type 01 (End-Of-File) record. Receipt of a type 01 record causes the following actions:

1) The total number of records that were actually loaded into the FLASH is printed. (Type 01 and type 04 records, as well as any type 00 records that were not written to FLASH do not count.) This can be checked manually, to make sure no records were dropped, and the load was in fact successful.

2) The total number of errors detected is printed. Anything other than 0000 means that the load was unsuccessful, and the loaded code should not be trusted.

3) Control returns to the Loader, reprinting the Loader's brief sign-on message. (To run the newly loaded code, hit the ESC key 3 times in a row)

Section 8. ME2700 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 is not 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 three electrolytic capacitors. Reversing these capacitors can cause some excitement.

There is logic to the order of assembly: the smaller components first, the larger 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 transistors.

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 one or two pins, then re-heating these solder connections while pressing the component 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 1/4 W, 1% resistors.

√	Qty	Locations	Value	Digikey Part Number
	1	R1	887 1%	887XBK-ND
	1	R2	287 1%	287XBK-ND

Step 2. Install the following 1/4 W, 5% resistors.

√	Qty	Locations	Value	Digikey Part Number
	1	R5	6.8 5%	S6.8HCT-ND
	2	R4,R18	270 5%	270QBK-ND
	2	R9,R13	1.6K 5%	1.6KQBK-ND
	1	R7	1.8K 5%	1.8KQBK-ND
	3	R3,R30,R31	10K 5%	10KQBK-ND
	2	R8,R10	22K 5%	22KQBK-ND
	2	R12,R14	43K 5%	43KQBK-ND
	5	R16,R23,R28,R29,R33	330 5%	330QBK-ND
	6	R11,R32,R34,R35,R37,R38	2K 5%	2.0KQBK-ND
	10	R15,R17,R19-R22,R24-R27	1K 5%	CF14JT1K00CT-ND

Step 3. Install the following 1 W, 5% (small form-factor) resistors.

√	Qty	Locations	Value	Digikey Part No.
	1	R6	1K 1W 5%	1KWCT-ND
	1	R36	2.2K 1W 5%	PPC2.2KW-1CT-ND

Step 4. 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.

Note that all of the BAT46 diodes are labeled “5817” on the PC board.

√	Qty	Locations	Value	Digikey Part Number
	5	D3-D6,D11	Diode, 1N4004	1N4004-TPMSCT-ND
	6	D1,D7-D10,D12	Diode, BAT46	BAT46CT-ND

Step 5. Install all of the DIP sockets flush to the PC board, paying attention to their orientation:

√	Qty	Locations	Value	Digikey Part Number
	2	U7,U11	8-pin DIP	ED3044-5-ND
	3	U4,U6,U8	14-pin DIP	ED3045-5-ND
	3	U1,U3,U5	16-pin DIP	ED3046-5-ND
	2	U9,U10	20-pin DIP	ED3054-5-ND
	1	U2	40-pin DIP	ED3048-5-ND

Step 6. 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	Value	Digikey Part Number
	1	D2	Diode, 1N5821	1N5821-TPMSCT-ND

Step 7. Install the following 14 capacitors. Note that C20 is labeled “0.1” on the PC board, but should be loaded with a 0.047 uF capacitor.

√	Qty	Locations	Value	Digikey Part Number
	1	C20	0.047 uF, 50V	BC2686CT-ND
	2	C17,C19	10 uF, 6.3V ceramic	445-8592-ND
	5	C1,C3-C6	1 uF, 16V	445-8614-ND
	10	C2,C10-C16,C18,C21	0.1 uF, 50V	478-4855-ND

Step 8. Install the following four 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	Value	Digikey Part Number
	1	V1	MCP1700-3302E	MCP1700-3302E/TO-ND
	1	V3	LM317L	LM317LZ-ND
	1	Q9	2N3904	2N3904FS-ND
	3	Q10,Q12,Q15	2N6520	2N6520TACT-ND
	9	Q3-Q8,Q11,Q13,Q14	2N3906	2N3906FS-ND

Step 9. Install 5 trim-pots in the following locations, and set them to approximately the center of their ranges.

√	Qty	Locations	Value	Digikey Part Number
	5	VR1-VR5	1K ohm Trimpot	3306P-102-ND

Step 10. Install Q1 standing straight up on the board. For Rev B boards, Q1’s writing faces toward the **left** (toward the inductor, L1). For Rev C boards, the writing faces down, away from the board edge.

√	Qty	Location	Value	Digikey Part Number
	1	Q1	TTC004B or 2SC6043	2SC6043-AEOSCT-ND or TTC004BQ-ND

Step 11. Install Q2 standing straight up on the board. For Rev B boards, Q2's writing faces toward the **right** (toward the EPROM socket). For Rev C boards, the writing faces down, away from the board edge.

√	Qty	Location	Value	Digikey Part Number
	1	Q2	BD440	BD440S-ND

Step 12. 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	Value	Digikey Part Number
	1	J1	5.5 mm barrel connector	CP-063AH-ND
	1	J3	DA9, female	626-1561-ND
	1	J2	Header, 6-pin	A31116-ND

Step 13. 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 uH	732-1424-ND

Step 14. Install the three 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.

√	Qty	Locations	Value	Digikey Part Number
	3	C7-C9	C 100 uF 35V, low ESR	495-6004-ND

Step 15. Screw the TO-220 voltage regulator to its heatsink, and then solder this subassembly onto the PC board.

√	Qty	Location	Value	Digikey Part Number
	1	V2	LM317	LM317TGOS-ND
	1	V2	Heat sink	HS368-ND
	1	V2	6-32 x 1/2″ screw	
	1	V2	6-32 nut	

Step 16. Install the Textool 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	Value	Digikey Part Number
	1	U12	24-pin ZIF socket	

Step 17. 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	Value	Digikey Part Number
	1	SW1	SPDT switch	EG2365-ND

Step 18. 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.)

√	Qty	Locations	Value	Digikey Part Number
	1	LED1	Blue LED	C503B-BCS-CV0Z0461-ND
	1	LED2	Red LED	365-1189-ND

Step 19. **(Rework)** Install a 22K ohm resistor on the solder side of the PC board. Solder one end of the resistor to the base of Q9, and the other end to the trace that connects to the emitter of Q9. (There is a nearby via on the trace to the emitter of Q9 that is perfect for this.) Be careful not to short out the pins of Q9.

√	Qty	Locations	Value	Digikey Part No.
	1	R42	22K 1/4W 5%	22KQBK-ND

☐ **Step 20.** Install a 6-32 x 1/2″ screw and a 6-32 nut as a support leg in each of the four corner holes.

☐ **Step 21.** 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-30[1]) 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 right 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 blue "POWER" LED should light.

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

√	Measure	Measurement	Meaning
	TP6	12.0V to 12.3V	Correct operation
▨		>12.3V	Incorrect wall adapter?
▨		<12V	Incorrect wall adapter? PC board short? Wrong component somewhere?
	TP5	5.1V to 5.25V	Correct operation
▨		otherwise	R1 and R2 correct? PC board short?
	TP7[2]	3.2V to 3.4V	Correct operation
▨		otherwise	Correct component in V2? PC board short?
	TP2	Adjust VR5 for 6.20V ± 0.02V	Correct operation
▨		Can't adjust	Incorrect R15 or R16? PC board short?
	TP3	11.2V to 11.7V	Correct operation
▨		Otherwise	Check switcher circuit: L1,D2,R5,Q2,D5, etc.

Step 3. Turn the ME2700 off and install the ADM660 IC in U11. Turn the ME2700 back on, and measure the -5V supply.

√	Measure	Measurement	Meaning
	TP4	-5V to -5.25V	Correct operation
▨		Otherwise	Incorrect C17-C19? U11 installed correctly? PC board short?

[1] The Wyse Technology WY-30 was the first product that I designed professionally.
[2] U10 pin 20 on Rev B PC Boards

Step 4. Turn off the ME2700 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	Value	Digikey Part Number
	1	U1	MAX232 RS-232 transceiver	296-1402-5-ND
	1	U8	74ACT04 High-power hex inverter	296-4351-5-ND
	2	U4,U6	7407 hex open-collector buffer	296-1436-5-ND
	2	U3,U5	74LS139 dual 2-bit decoder	296-1640-5-ND
	1	U9	74LS374 8-bit D flip-flop	296-1662-5-ND
	1	U10	74LVC245A low-voltage 8-bit buffer	296-8503-5-ND
	1	U7	23K640 8K-byte serial SRAM	23K640-I/P-ND
	1	U2	PIC Microcontroller, pre-programmed with ME2700 firmware	PIC18F45K20-I/P-ND + ME2700 Firmware

U2 is a PIC microcontroller with internal flash memory. You must use a PIC that has been pre-programmed with the ME2700 Loader Kernel 1.0, or program it in place yourself, using a PC, a Microchip PICkit-3 programming device, and my program file. (J2 is the PICkit-3 compatible in-circuit programming connector for this purpose.) The PIC must also be loaded with the ME2700 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 ME2700 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 ME2700 Programmer's serial port connector, making sure to connect the transmit signal (TxD, pin 2) from the ME2700 to receive signal of the terminal and the receive signal (RxD, pin 3) from the ME2700 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 ME2700 Programmer and turn it on. On the terminal screen, you should see a sign-on banner and a prompt like this:

```
*====================================*
*               ME2700               *
*====================================*
*       Orphan EPROM Programmer      *
*         By Martin Eberhard         *
*       Firmware Version  1.03       *
*====================================*


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 ME2700 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 blue LED should now be lit. Debug if not.

√	LED	State	Meaning
	LED1 Power	On	Correct
		Off	PC Board short? IC inserted backwards?
	LED2 Busy	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 ME2700'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. If the terminal and/or ME2700 Programmer are off, then turn them back on.

Step 1. Test and Adjust Vpp Supply

☐ Connect the positive lead of your voltmeter to TP3 for the following tests.

```
>AVPP 1
Vpp set for 12.70V (Measure at TP3)
Note: Vpp will be about 0.7V higher on pin 18
>
```

Now, both LEDs should be lit:

√	LED	State	Meaning
	LED1 Power	On	Correct
		Off	LED1 Orientation?
	LED2 Busy	On	Correct
		Off	LED orientation? PC board short?

Measure the voltage at TP3

√	Measurement	Meaning	Action
	11V to 16V	Correct operation	Adjust VR1 for 12.70V ± 0.05V
	0V to 10.9V	Problem	• Correct component in R7 and R8? • Problem with Vpp Switcher circuit?
	>16V	Problem	• Correct component in R7 and R8?

```
>AVPP 2
Vpp set for 13.15V (Measure at TP3)
Note: Vpp will be about 0.7V higher on pin 18
>
```

Measure the voltage at TP3

√	Measurement	Meaning	Action
	11V to 17V	Correct operation	Adjust VR2 for 13.15V ± 0.05V
	0V to 10.9V	Problem	• Correct component in R9 and R10? • Problem with Vpp Switcher circuit?
	>17V	Problem	• Correct component in R9 and R10?

```
>AVPP 3
Vpp set for 21V (Measure at TP3)
Note: Vpp will be about 0.7V higher on pin 18
>
```

Measure the voltage at TP3

√	Measurement	Meaning	Action
	16V to 25V	Correct operation	Adjust VR3 for 21.00V ± 0.05V
	0V to 15.9V	Problem	• Correct component in R11 and R12? • Problem with Vpp Switcher circuit?
	>25V	Problem	• Correct component in R11 and R12?

```
>AVPP 4
Vpp set for 25.2V (Measure at TP3)
Note: Vpp will be about 0.7V higher on pin 18
>
```

Measure the voltage at TP3

√	Measurement	Meaning	Action
	19V to 32V	Correct operation	Adjust VR4 for 25.20V ± 0.05V
	0V to 18.9V	Problem	• Correct component in R13 and R14? • Problem with Vpp Switcher circuit?
	>32V	Problem	• Correct component in R13 and R14?

☐ **Step 2. Test Data Outputs**

```
>WD 55
>
```

The BUSY light should now be on. Use a voltmeter to measure the voltages on the data pins (9-11 and 13-17) 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 an 8K EPROM, which will have 13 address lines:

```
>ET 23
Current Device Type is 24: 57C49C
>
```

Now write a pattern to the address lines:

```
>WA 0AAA
>
```

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

```
>ED
Type 24: 57C49C, size: 8192 x 8
       -----v-----
 A7  -| 1      24 |-  Vcc     Programming Vcc = 6.2V
 A6  -| 2      23 |-  A8
 A5  -| 3      22 |-  A9
 A4  -| 4      21 |-  A10
 A3  -| 5      20 |-  -OE/Vpp Pulsed from 5V to 12.7V
 A2  -| 6      19 |-  A11
 A1  -| 7      18 |-  A12
 A0  -| 8      17 |-  D7      Supported Devices:
 D0  -| 9      16 |-  D6       WSI        WS57C49C
 D1  -| 10     15 |-  D5
 D2  -| 11     14 |-  D4
GND  -| 12     13 |-  D3
       -----------
Vpp during read: 0V
Programming pulse width: 150 uS
Programming cycles: 1
>
```

☐ Use this picture as a guide to measure the 13 address line voltages. Try it again with address bits at the opposite polarities:

```
>WA 1555
>
```

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 10K resistor in the ZIF socket, between ground (pin 12) 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 24 active state
>WD 0
>RD
Data Read: 00
>
```

☐ Use a 220 ohm resistor to pull each of the data pins high (to pin 24 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 Vcc at 5V and 6.2V

☐ Pin 24 should now measure about 5.15V.

```
>TVCC 2
Vcc pin 24 at programming level
>
```

☐ Pin 24 should now measure 6.20V.

```
>TVCC 0
Vcc pin 24 inactive state
>
```

☐ Pin 24 should now measure less than 0.3V.

Step 6. Test 12.7V Vpp on pin 20

```
>TVPP 3
Vpp pin 20 programming mode 12.7V
>
```

☐ Pin 20 should now measure 12.7V.

```
>TVPP 2
Vpp pin 20 programming mode inactive state
>
```

☐ Pin 20 should now measure about 5V

```
>TVPP 1
Vpp pin 20 read mode
>
```

☐ Pin 20 should now be less than 0.3V.

```
>TVPP 0
Vpp pin 20 powered off
>
```

☐ Pin 20 should still be less than 0.3V.

Step 7. Test Vbb and Vdd

```
>ET 4
Current Device Type is 04: 2708
 (The Busy LED should have turned off.)
>TVBD 1
-5V Vbb pin 21 and +12V Vdd pin 19 active state
>
```

☐ Pin 21 should now measure about -5.15V, and pin 19 should now measure about +12V.

```
>TVBD 0
-5V Vbb pin 21 and +12V Vdd pin 19 inactive state
>
```

☐ Pin 21 sand pin 19 should now measure about 0V.

Step 8. Test -OE

```
>TOE 2
-OE pin 20 at programming level
>
```

☐ Pin 20 should now measure about +12V

```
>TOE 1
-OE pin 20 active state
>
```

☐ Pin 20 should now measure about 0V

```
>TOE 0
-OE pin 20 inactive state
>
```

☐ Pin 20 should now measure about 5V.

Step 9. Test 25.9V Vpp on pin 18

```
>TVPP 3
Vpp pin 18 programming mode 25.9V
>
```

☐ Pin 18 should now measure about 25.9V.

```
>TVPP 0
Vpp pin 18 powered off
```

```
>
```

☐ Pin 18 should now measure less than 0.39V.

Step 10. Test 25.2V Vpp on pin 21

```
>ET 5
Current Device Type is 05: TMS2758
>TVPP 3
Vpp pin 21 programming mode 25.2V
>
```

☐ Pin 21 should now measure about 25.2V.

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

☐ Pin 21 should now measure less than 0.2V.

Step 11. Test 12V Vcc

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

☐ Pin 24 should now measure about 12V.

```
>TVCC 1
Vcc pin 24 active state
>
```

☐ Pin 24 should now measure about 5.15V.

```
>TVCC 0
Vcc pin 24 inactive state
>
```

☐ Pin 24 should now measure less than 0.2V.

Step 12. Reset When Done

```
>RESET
*====================================*
*               ME2700               *
*====================================*
*       Orphan EPROM Programmer      *
*         By Martin Eberhard         *
*       Firmware Version  1.03       *
*====================================*


Current Device Type is 0C: TMS2716
EPROM data invert off
Type ? for command list
>
```

☐ **Step 13. 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 ME2700 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 ME2700, and see that your terminal program can talk to it.

Note: The ME2700 Programmer uses a ‘File Address Offset’ when uploading and downloading files. The File Address Offset is set by the user (with the F**AO** 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 F**AO** 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 ME2700 Programmer. The ME2700 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 ME2700 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   UUUUUUUUUUUUUUUU
050: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
060: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
070: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
080: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
090: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0A0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0B0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0C0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0D0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0E0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0F0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
100: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
110: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
120: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
130: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
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         **************
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   .........*******
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 0 100           {start file capture before hitting Return}
:10680000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAE8
:10681000AAAAAAAAAAAAAAAAA9AAAAAAAAAAAAAAD8
:10682000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAC8
:10683000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAB8
:10684000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA8
:10685000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA98
:10686000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA88
:10687000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA78
:10688000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA68
:10689000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA58
:1068A000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA48
:1068B000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA38
:1068C000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA28
:1068D000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA18
:1068E000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA08
:1068F000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAF8
:10690000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAE7
:1069100001020304050607080 9AAAAAAAAAAAAAA4
:10692000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAC7
:10693000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAB7
:10694000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA7
:10695000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA97
:10696000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA87
:10697000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA77
:10698000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA67
:10699000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA57
:1069A000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA47
:1069B000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA37
:1069C000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA27
:1069D000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA17
:1069E000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA07
:1069F000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAF7
:00000001FF
>
```

{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 0 100           {start file capture before hitting Return}

S1133100AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA1B
S1133110AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA0B
S1133120AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFB
S1133130AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEB
S1133140AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADB
S1133150AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACB
S1133160AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABB
S1133170AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAB
S1133180AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA9B
S1133190AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA8B
S11331A0AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA7B
S11331B0AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA6B
S11331C0AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA5B
```

```
S11331D0AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA4B
S11331E0AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA3B
S11331F0AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA2B
S1133200AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA1A
S113321001020304050607080 9AAAAAAAAAAAAAD7
S1133220AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFA
S1133230AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEA
S1133240AAAAAAAAAAAAAAAAAAAAAAAAAAAAAADA
S1133250AAAAAAAAAAAAAAAAAAAAAAAAAAAAAACA
S1133260AAAAAAAAAAAAAAAAAAAAAAAAAAAAAABA
S1133270AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
S1133280AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA9A
S1133290AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA8A
S11332A0AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA7A
S11332B0AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA6A
S11332C0AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA5A
S11332D0AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA4A
S11332E0AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA3A
S11332F0AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA2A
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 ME2700 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 ME2700.) 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                  ..........
30: 99 99 99 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 99 99 99  ................
50: 99 99 99 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 ME2700}

```
>:10680000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAE8
:10681000AAAAAAAAAAAAAAAA9AAAAAAAAAAAAAD8
:10682000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAC8
:10683000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAB8
:10684000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA8
:10685000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA98
:10686000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA88
:10687000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA78
:10688000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA68
:10689000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA58
:1068A000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA48
:1068B000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA38
:1068C000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA28
:1068D000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA18
:1068E000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA08
:1068F000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAF8
:10690000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAE7
:106910000102030405060708 09AAAAAAAAAAAAA4
```

```
:10692000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAC7
:10693000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAB7
:10694000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA7
:10695000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA97
:10696000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA87
:10697000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA77
:10698000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA67
:10699000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA57
:1069A000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA47
:1069B000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA37
:1069C000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA27
:1069D000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA17
:1069E000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA07
:1069F000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAF7
:00000001FF
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 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 AA   ****************
130: AA 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 AA   ****************
150: AA 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 AA   ****************
170: AA 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 AA   ****************
190: AA 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 AA   ****************
1B0: AA 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 AA   ****************
1D0: AA 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 AA   ****************
1F0: AA 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 12
Current Device Type is 12: 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 55   ..... @.*UUUUUUU
0010: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0020: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0030: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0040: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0050: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0060: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0070: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0080: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0090: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
00A0: DE F0 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
00B0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
00C0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
```

```
00D0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
00E0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
00F0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0100: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0110: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0120: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0130: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0140: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0150: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0160: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0170: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0180: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0190: 55 55 55 55 55 55 55 55 55 55 12 34 56 78 9A BC   UUUUUUUUUU.4VX.<
01A0: DE F0 55 55 55 55 55 55 55 55 55 55 55 55 55 55   ^pUUUUUUUUUUUUUU
01B0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
01C0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
01D0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
01E0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
01F0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
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 ME2700 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   ..... @.*UUUUUUU
0010: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0020: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0030: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0040: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0050: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0060: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0070: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0080: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
```

```
0090: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
00A0: DE F0 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
00B0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
00C0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
00D0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
00E0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
00F0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0100: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0110: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0120: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0130: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0140: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0150: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0160: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0170: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0180: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0190: 55 55 55 55 55 55 55 55 55 55 12 34 56 78 9A BC   UUUUUUUUUU.4VX.<
01A0: DE F0 55 55 55 55 55 55 55 55 55 55 55 55 55 55   ^pUUUUUUUUUUUUUU
01B0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
01C0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
01D0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
01E0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
01F0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
Buffer checksum: 3C
>
```

Congratulations, your ME2700 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 both the Rev B and Rev C ME2700 PC Boards, without the Intersil Option. (See page 4 for the Intersil Option components. All Digikey part numbers are current, and all components are in stock at the time this was written.

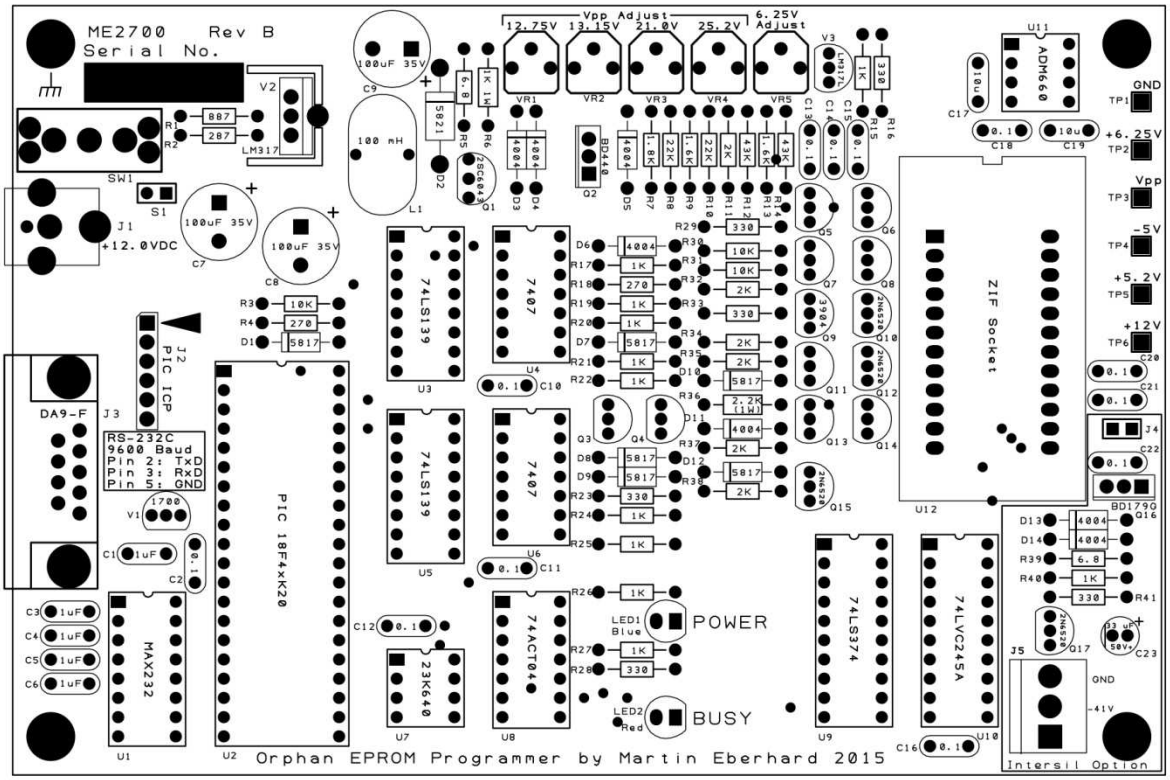
Note that Digikey charges far too much for Textool ZIF sockets. You will save about $10 if you buy this part on eBay. Look for the type that has the wider slots for the EPROM pins, so that your ME2700 can program both the normal 0.6" wide EPROMs, as well as the "Skinny DIP" 0.3" wode EPROMs.

Component	Value	Reference Name	Qty	Digikey Part Number
¼ W Resistor	887 Ω 1%	R1	1	887XBK-ND
¼ W Resistor	287 Ω 1%	R2	1	287XBK-ND
¼ W Resistor	6.8 Ω	R5	1	S6.8HCT-ND
¼ W Resistor	270 Ω	R4,R18	2	270QBK-ND
¼ W Resistor	1.6 KΩ	R9,R13	2	1.6KQBK-ND
¼ W Resistor	1.8 KΩ	R7	1	1.8KQBK-ND
¼ W Resistor	10 KΩ	R3,R30,R31	3	10KQBK-ND
¼ W Resistor	22 KΩ	R8,R10,R42	3	22KQBK-ND
¼ W Resistor	43 KΩ	R12,R14	2	43KQBK-ND
¼ W Resistor	330 Ω	R16,R23,R28,R29,R33	5	330QBK-ND
¼ W Resistor	2 KΩ	R11,R32,R34,R35,R37,R38	6	2.0KQBK-ND
¼ W Resistor	1 KΩ	R15,R17,R19-R22,R24-R27	10	CF14JT1K00CT-ND
1W Resistor	1 KΩ	R6	1	1KWCT-ND
1W Resistor	2.2 KΩ	R36	1	PPC2.2KW-1CT-ND
Silicon Diode	1N4004	D3-D6,D11	5	1N4004-TPMSCT-ND
Schottky Diode	BAT46	D1,D7-D10,D12	6	BAT46CT-ND
8-pin DIP socket		U7,U11	2	ED3044-5-ND
14-pin DIP socket		U4,U6,U8	3	ED3045-5-ND
16-pin DIP socket		U1,U3,U5	3	ED3046-5-ND
20-pin DIP socket		U9,U10	2	ED3054-5-ND
40-pin DIP socket		U2	1	ED3048-5-ND
Schottky Diode	1N5821 (30V, 3A)	D2	1	1N5821-TPMSCT-ND
Ceramic Capacitor	0.047 µF, 50V	C20	1	BC2686CT-ND
Ceramic Capacitor	10 µF, 6.3V	C17,C19	2	445-8592-ND
Ceramic Capacitor	1µF, 16V	C1,C3-C6	5	445-8614-ND
Ceramic Capacitor	0.1µF, 100V	C2,C10-C16,C18,C21	10	478-4855-ND
3.3V Regulator	MCP1700-3302E/TO	V1	1	MCP1700-3302E/TO-ND
Adj. Regulator	LM317L	V3	1	LM317LZ-ND
NPN Transistor	2N3904	Q9	1	2N3904FS-ND

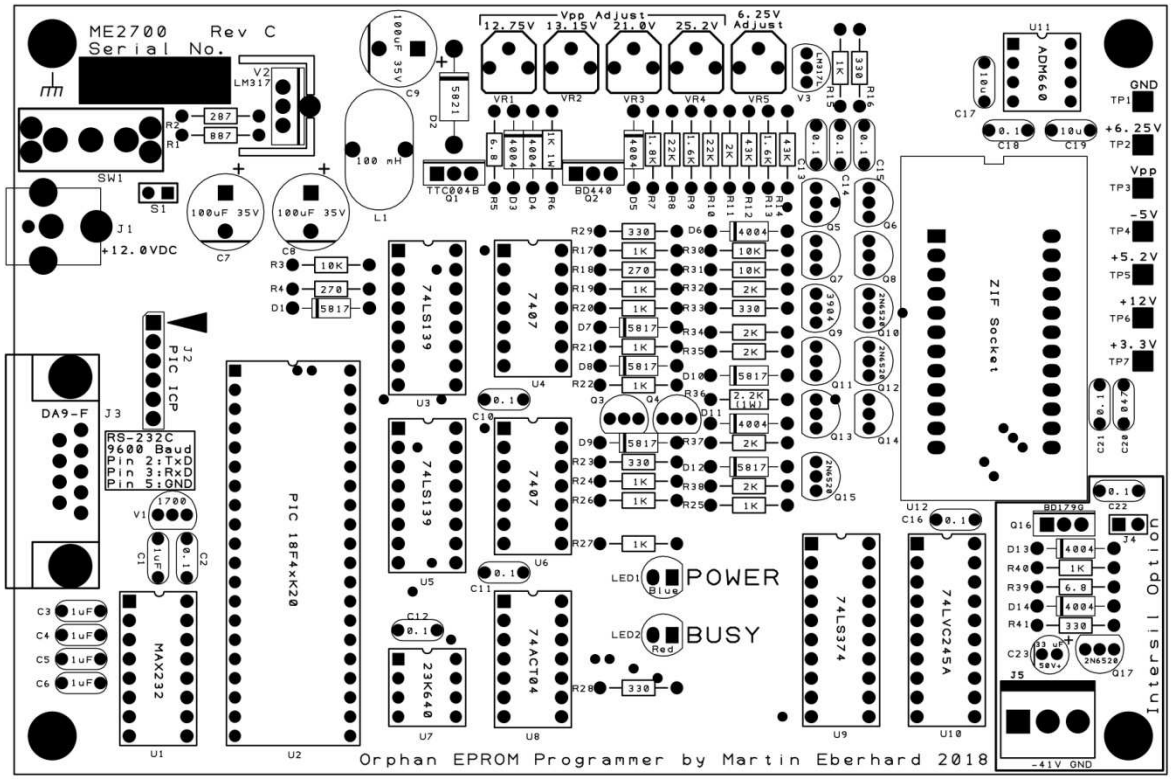
Component	Value	Reference Name	Qty	Digikey Part Number
PNP Transistor	2N6520	Q10,Q12,Q15	3	2N6520TACT-ND
PNP Transistor	2N3906	Q3-Q8,Q11,Q13,Q14	9	2N3906FS-ND
Trimpot Bournes	1 KΩ	VR1-VR5	5	3306P-102-ND
Trimpot AMP/TE	1 KΩ	VR1-VR5 (Alternate)	0	A105776-ND
NPN Transistor	TTC004B	Q1	1	TTC004BQ-ND
PNP Transistor	BD440	Q2	1	BD440S-ND
Barrel Connector	5.5mm x 2.0mm	J1	1	CP-063AH-ND
9-pin Connector	DA9F	J3	1	626-1561-ND
6-pin header	0.1” Spacing	J2	1	A31116-ND
Electrolytic Cap.	100 µF, 35V	C7-C9	3	495-6004-ND
Torroidal Inductor	100 µH, 2A	L1	1	732-1424-ND
Adj. Regulator	LM317	V2	1	LM317TGOS-ND
TO-220 Heat Sink	577102B04000G	V2	1	HS368-ND
6-32 x 1/2" screw		V2 & PCB corners	5	
6-32 nut		V2 & PCB corners	5	
24-pin ZIF socket	Textool	U12	1	Buy on eBay!
SPDT Switch	5A, 120V	SW1	1	EG2365-ND
Blue LED	5 mm	LED1	1	C503B-BCS-CV0Z0461-ND
Red LED	5 mm	LED2	1	365-1189-ND
Switched Cap. Inverting Regulator	ADM660	U11	1	ADM660ANZ-ND
RS232 Tranceiver	MAX232	U1	1	296-1402-5-ND
Hex Inverter	74ACT04	U8	1	296-4351-5-ND
Hex O.C. Driver	7407	U4,U6	2	296-1436-5-ND
Dual Data Selector	74LS139	U3,U5	2	296-1640-5-ND
8-Bit D-Flip-Flop	74LS374	U9	1	296-1662-5-ND
8-Bit Bi-dir. Buffer	74LVC245A	U10	1	296-8503-5-ND
64 kb Serial RAM	23K640	U7	1	23K640-I/P-ND

Rev B and C PCBA Component Placement

These drawings are the printed circuit board outlines and silkscreen layers for both the Rev B and Rev C PC Boards. These can help you find components on the printed circuit board assembly.



Rev B PC Board



Rev C PC Board

11.2 Rev B and C PCBA Schematics

The following pages are the schematic for the rev B and Rev C ME2700 Programmer’s printed circuit board assembly.

Busy
Power
LED2
LED1
R28 330
R27 1K
U8d 74ACT04
GND

Pin-24 Control
Q7 2N3906
Q8 2N3906
U6a 7407
U6b 7407
R18 270
R30 10K

V3 LM317L
R16 330
R15 1K
C15 0.1uF
VR5 1K
6.25V Adjust
TP2 6.25V
Q6 2N3906
R31 10K
C21 0.1uF

Low Address Latch
U9 74LS374
U12 ZIF Socket
U10 74LVC245A
C16 0.1uF
C20 0.047uF
Note: PCB is marked for C20 = 0.1 uF

U2 PIC18F45K20 Microcontroller
RD0 RD1 RD2 RD3 RD4 RD5 RD6 RD7
RA0/C12IN0- RA1/C12IN1- RA2 RA3 RA4 RA5 RA6 RA7
RB0 RB1/C12IN3- RB2 RB3/C12IN2- RB4 RB5
RC0 RC1 RC2/P1A RC3/SCK1 RC4/SDI1 RC5/SDO1 RC6/TX RC7/RX
RE0 RE1 RE2
M25V M21V P18_0 P18_1 P19_0 P19_1 P20_0 P20_1
-P24_5V M13.1V -P24_12V M12.75V -P24_6V -A9 -A8 -SRAM_CS VPPULSE SCK SI SO P21_0 P21_1 ADRCLK
SETUP -DBUFOE DBUFDIR
RE3/Vpp RB7/ICDDAT RB6/ICDCLK
Vdd Vss
R3 10K
C12 0.1uF
C2 0.1uF
J2 In-Circuit Programming Connector

U7 8KB SRAM
-HOLD -CS SCK SO SI Vdd N/C Vss
D1 BAT48 or 1N5817

Pin 18 Control:
11=Vpp_high
10=-40V
01=TTL high
00=0V
U5a 74LS139
U6c 7407
U8c 74ACT04
U6f 7407
R25 1K
R38 2K
Q15 2N6520
Vpp_high

Pin 19 Control:
11=N/A
10=12V
01=5V
00=0V
U5b 74LS139
U6d 7407
U6e 7407

Pin 20 Control:
11=Vpp
10=12V
01=5V
00=0V
U3a 74LS139
U4c 7407
U4b 7407
U4a 7407
R21 1K
R34 2K
Q12 2N6520
R20 1K
R35 2K
Q11 2N3906
R29 330
Q5 2N3906
D6 1N4004

Pin 21 Control:
11=Vpp
10=-5V
01=5V
00=0V
U3b 74LS139
U4d 7407
U4e 7407
U8e 74ACT04
R19 1K
R32 2K
Q10 2N6520
D8 BAT46
Q4 2N3906
D7 BAT46
R22 1K
R26 1K
R33 330
Q3 2N3906
Q9 2N3904
R42 22K (REWORK)
TP4 -5V

U4f 7407
U8f 74ACT04
U8b 74ACT04

R36 2.2K 1W
D11 1N4004
D9 BAT46
Q16 BD179Q
J4 Jumper for Intersil EPROM
R40 1K
D13 1N4004
D14 1N4004
R39 6.8
-41V
J5 Header 3 156
C23 33 uF 50V
C22 0.1uF
Q17 2N6520
R41 330
Intersil Option

R37 2K
Q14 2N3906
R24 1K
Q13 2N3906
D12 BAT46 or 1N5817
R23 330
D10 BAT46 or 1N5817

U8a 74ACT04
Spare Gate

U1 MAX232
C1 1uF
C4 1uF
C6 1uF
C3 1uF
C5 1uF
TxD RxD
J3 DA9F
RS-232C Tranceiver

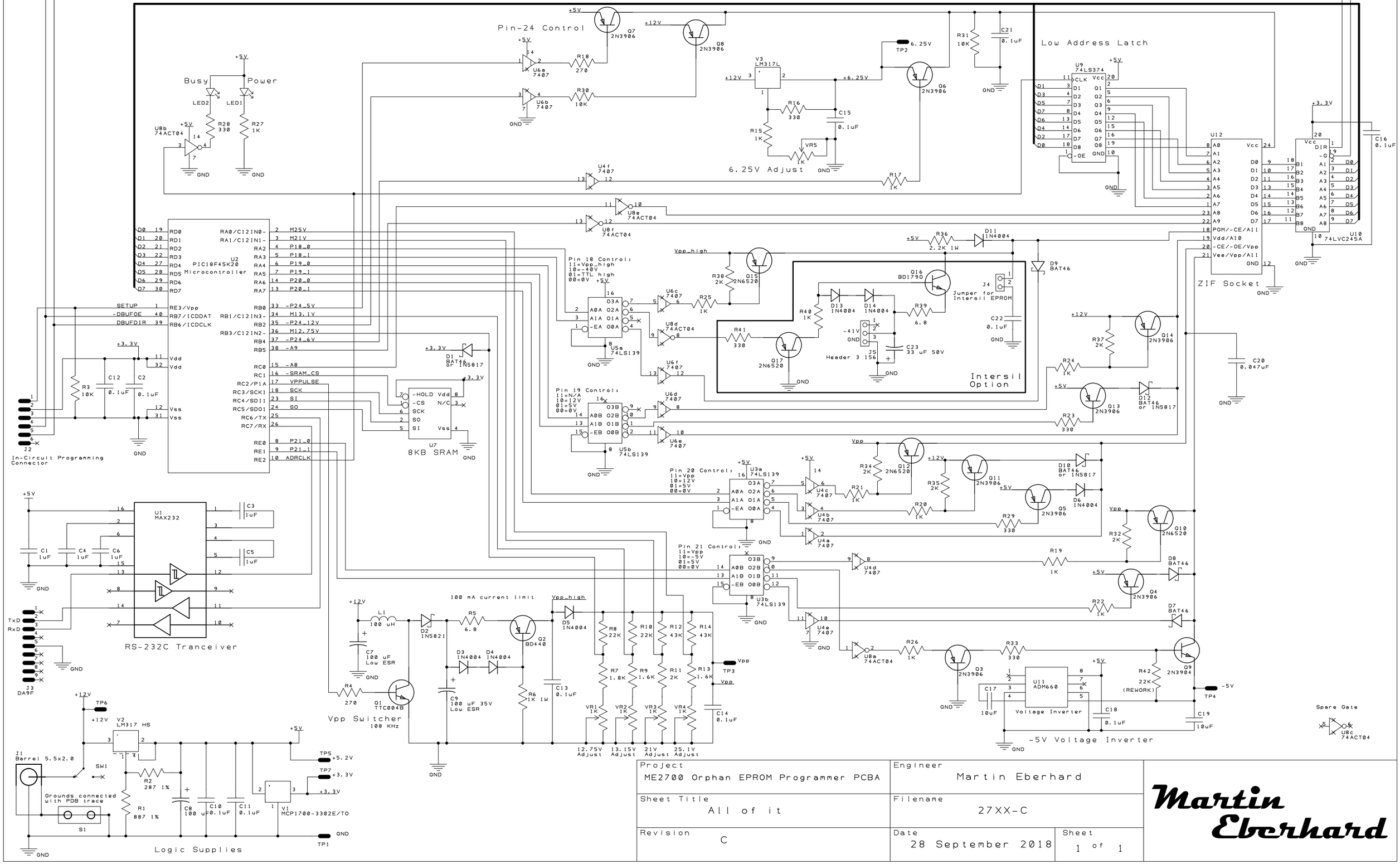
L1 100 uH
D2 1N5821
C7 100 uF Low ESR
R5 6.8
Q2 BD440
D3 1N4004
D4 1N4004
C9 100 uF 35V Low ESR
R4 270
Q1 2SC6043
R6 1K 1W
D5 1N4004
C13 0.1uF
100 mA current limit
Vpp_high
Vpp Switcher
108 KHz
R8 22K
R10 22K
R12 43K
R14 43K
R7 1.8K
R9 1.6K
R11 2K
R13 1.6K
VR1 1K
VR2 1K
VR3 1K
VR4 1K
12.75V Adjust
13.15V Adjust
21V Adjust
25.1V Adjust
TP3 Vpp
C14 0.1uF

U11 ADM660
C17 10uF
C18 0.1uF
C19 10uF
Voltage Inverter
-5V Voltage Inverter

J1 Barrel 5.5x2.0
SW1
S1
Grounds connected with PDB trace
V2 LM317 HS
R2 287 1%
R1 887 1%
C8 100 uF
C10 0.1uF
C11 0.1uF
V1 MCP1700-3302E/TO
TP5 +5.2V
TP6
TP1
Logic Supplies

Project: 27xx Orphan EPROM Programmer PCBA
Engineer: Martin Eberhard
Sheet Title: All of it
Filename: 27XX-B
Revision: B
Date: 12 November 2015
Sheet: 1 of 1

Martin Eberhard


Pin-24 Control
Busy
Power
LED2
LED1
R28 330
R27 1K
U8b 74ACT04
U6a 7407
U6b 7407
R18 270
R30 10K
Q7 2N3906
Q8 2N3906
V3 LM317L
R16 330
R15 1K
C15 0.1uF
VR5 1K
6.25V Adjust
TP2 6.25V
Q6 2N3906
R31 10K
C21 0.1uF
Low Address Latch
U9 74LS374
U12
ZIF Socket
U10 74LVC245A
C16 0.1uF
U4f 7407
R17 1K
U8e 74ACT04
U8f 74ACT04
U2 PIC18F45K20 Microcontroller
SETUP
-DBUFOE
DBUFDIR
RA0/C12IN0-
RA1/C12IN1-
RB1/C12IN3-
RB3/C12IN2-
M25V
M21V
P18_0
P18_1
P19_0
P19_1
P20_0
P20_1
P24_5V
M13.1V
P24_12V
M12.75V
P24_6V
-A9
-A8
-SRAM_CS
VPPULSE
SCK
SI
SO
P21_0
P21_1
ADRCLK
Pin 18 Control:
11=Vpp_high
10=-40V
01=TTL high
00=0V
U5a 74LS139
U6c 7407
U8d 74ACT04
U6f 7407
R25 1K
R38 2K
Q15 2N6520
Vpp_high
R36 2.2K 1W
D11 1N4004
D9 BAT46
Q16 BD179Q
Jumper for Intersil EPROM
R40 1K
D13 1N4004
D14 1N4004
R39 6.8
R41 330
Q17 2N6520
C22 0.1uF
C23 33 uF 50V
Intersil Option
Pin 19 Control:
11=N/A
10=12V
01=5V
00=0V
U5b 74LS139
U6d 7407
U6e 7407
Vpp
R24 1K
R37 2K
Q14 2N3906
Q13 2N3906
D12 BAT46 or 1N5817
R23 330
C20 0.047uF
Pin 20 Control:
11=Vpp
10=12V
01=5V
00=0V
U3a 74LS139
U4c 7407
U4b 7407
U4a 7407
R21 1K
R34 2K
Q12 2N6520
R35 2K
R20 1K
Q11 2N3906
R29 330
Q5 2N3906
D10 BAT46 or 1N5817
D6 1N4004
Pin 21 Control:
11=Vpp
10=-5V
01=5V
00=0V
U3b 74LS139
U4d 7407
U4e 7407
R19 1K
R32 2K
Q10 2N6520
D8 BAT46
Q4 2N3906
R22 1K
D7 BAT46
R26 1K
R33 330
Q3 2N3906
Q9 2N3904
U8a 74ACT04
U11 ADM660
Voltage Inverter
C17 10uF
C18 0.1uF
C19 10uF
R42 22K (REWORK)
TP4 -5V
-5V Voltage Inverter
Spare Gate
U8c 74ACT04
J2
In-Circuit Programming Connector
R3 10K
C12 0.1uF
C2 0.1uF
U7
8KB SRAM
D1 BAT48 or 1N5817
J3 DA9F
RS-232C Tranceiver
U1 MAX232
C1 1uF
C4 1uF
C6 1uF
C3 1uF
C5 1uF
TxD
RxD
L1 100 uH
D2 1N5821
C7 100 uF Low ESR
R5 6.8
Q2 BD440
D3 1N4004
D4 1N4004
C9 100 uF 35V Low ESR
R4 270
Q1 TTC004B
Vpp Switcher
108 KHz
100 mA current limit
Vpp_high
D5 1N4004
R8 22K
R10 22K
R12 43K
R14 43K
R7 1.8K
R9 1.6K
R11 2K
R13 1.6K
VR1 1K
VR2 1K
VR3 1K
VR4 1K
12.75V Adjust
13.15V Adjust
21V Adjust
25.1V Adjust
C13 0.1uF
C14 0.1uF
TP3 Vpp
J1 Barrel 5.5x2.0
SW1
S1
Grounds connected with PDB trace
R2 287 1%
R1 887 1%
C8 100 uF
C10 0.1uF
C11 0.1uF
V2 LM317 HS
V1 MCP1700-3302E/TO
TP6
TP5 +5.2V
TP7 +3.3V
TP1
Logic Supplies
Project
ME2700 Orphan EPROM Programmer PCBA
Engineer
Martin Eberhard
Sheet Title
All of it
Filename
27XX-C
Revision
C
Date
28 September 2018
Sheet
1 of 1
Martin Eberhard

Appendix A. Supported EPROMs and EEPROMs

The following is a list of the EPROMs and EEPROMs supported by the current version of the ME2700 firmware. Although they are generally in order of size, the order is arbitrary. As more EPROMs are supported in future firmware releases, new EPROM Device Types will be added to the end of the list. (In other words, these Device Type numbers will not change.)

Underlined devices have been tested on the ME2700. Devices in *italics* have not been tested (generally because the part is unavailable), but data sheets have been verified.

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

EPROMs and EEPROMs 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*

Type 03: IM6654 **512x8 EPROM** (Requires Intersil Option)
Intersil IM6654

Type 04: 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 05: 2758 **1024x8 EPROM** (Vcc=+5V, pin 19 low)
Harris HM-6758, Intel 2758, National Semiconductor MM2758Q-A, *Texas Instruments TMS2508, Texas Instruments TMS2758-JL0*

Type 06: 2716 **2048x8 EPROM** (Vcc=+5V, Vpp=25V)
AMD AM2716, Eurotechnique ET2716Q, *Fujutsi MBM2716, Hitachi HN462716,* Intel 2716, *Mitsubishi M5L2716K, MME U2716C, Motorola MCM2716, National Semiconductor MM2716, 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 07: 2716A **2048x8 EPROM** (Vcc=+5V, Vpp=21V)
(No datasheets found)

Type 08: 2716A-fast **2048x8 EPROM** (Vcc=+5V, Vpp=21V)
SGS-Thomson M2716A-fast

Type 09: 2716B **2048x8 EPROM** (Vcc=+5V, Vpp=12.7V)
AMD AM2716B

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

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

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

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

Type 0E: 57C191C **2048x8 EPROM**
Waferscale Integration WS57C191C, Waferscale Integration WS57C291C

Type 0F: 2816A **2048x8 EEPROM** (time delay for write completion)
Samsung KM2816A, *Seeq 2816A, Seeq 5516A*

Type 10: 28C16 **2048x8 EEPROM** (polled write completion)
Atmel AT28C16, *Atmel 28C16E, Catalyst CAT28C16A, Exel XLS2816A,* Exel XLS28C16A, Microchip 28C16A, *On Semiconductor CAT28C16A,* Xicor X2816B

Type 11: 2816Ai **2048x8 EEPROM** (erase before write, >10 mS write pulse)
Intel 2816A, *Seeq 52B13*

Type 12: 2732 **4096x8 EPROM** (Vpp=25V)
AMD AM2732, *Electronic Arrays EA2732Q, Fairchild F2732,* Fujitsu MBM2732, Hitachi HN472732G, Intel 2732, Mitsubishi M5L2732K, MME U2732, *Motorola MCM2732,* NEC uPD2732, *Toshiba TMM2732D*, *Toshiba TMM2732DI*

Type 13: 2732A **4096x8 EPROM** (Vpp=21V)
AMD AM2732A, *Fujitsu MBM2732A, Hitachi HN482732AG, Intel 2732A, NEC uPD2732A,* Rockwell R87C32

Type 14: 2732A-fast **4096x8 EPROM** (Vpp=21V)
SGS-Thomson M2732A-fast

Type 15: 2732B **4096x8 EPROM** (Vpp=12.7V)
AMD AM2732B

Type 16: 27C32H **4096x8 EPROM** (Vpp=12.7V)
Fairchild NMC27C32H, National Semiconductor NMC27C32H

Type 17: 27C32B **4096x8 EPROM** (Vpp=12.7V)
Fairchild NMC27C32B, National Semiconductor NMC27C32B

Type 18: TMS2532 **4096x8 EPROM** (TI-unique pinout, Vpp=25V)
Hitachi HN462532, *Motorola MCM2532, SGS M2532,* Texas Instruments TMS2532

Type 19: TMS2532A **4096x8 EPROM** (TI-unique pinout, Vpp=21V)
Texas Instruments TMS2532A

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

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

Type 1C: 57C43C **4096x8 EPROM**
Waferscale Integration WS57C43C

Type 1D: 68764 **8192x8 EPROM**
Motorola MCM68764, Motorola MCM68766

Type 1E: LH5749 **8192x8 EPROM**
Sharp LH5749

Type 1F: 27HC641 8192x8 EPROM
Atmel AT27HC641, *Atmel AT27HC642, Microchip 27HC641*

Type 20: 27HC641s 8192x8 EPROM (no blank-check)
Signetics 27HC641

Type 21: 27HC641R 8192x8 EPROM
Atmel AT27HC641R, Atmel AT27HC642R, *Microchip 27HC641*

Type 22: 57C49B 8192x8 EPROM
Waferscale Integration WS57C49B

Type 23: 57C49C 8192x8 EPROM
Waferscale Integration WS57C49C

Type 24: 52B13H 2048x8 EEPROM, 1.2 mS write pulse
Seeq 52B13H

Type 25: IM6658 1024x8 EEPROM, (Requires Intersil Option)
Intersil IM6658

Type 26: LH57191 2048x8 EEPROM
Sharp LH57191

Type 27: 28C04n 512x8 EEPROM
NEC 28C04

EPROMs and EEPROMs Sorted by Manufacturer

Numbers in parenthesis are the ME2700 Device Type. As above, chips that have been tested on the ME2700 are underlined. Those in *italics* have only had their data sheets verified.

AMD
AM2708 (04), AM2716 (06), AM2716B (09), AM2732 (12), AM2732A (13), AM2732B (15)

Atmel
AT28C04 (02), AT28C16 (10), *28C16E* (10), AT27HC641 (1F), *AT27HC642 (1F), AT27HC641R* (21), AT27HC642R (21)

Catalyst
CAT28C16A (10)

Electronic Arrays
EA2708 (04), *EA2732Q* (12)

Eurotechnique
ET2716Q (06)

Exel
X12804A (01), *XLS2816A* (10), XLS28C16A (10)

Fairchild
F2708 (04), *F2732* (12), *NMC27C16B* (0B), *NMC27C16H* (0A), NMC27C32B (17), *NMC27C32H* (16)

Fujitsu
MBM2732 (12), *MBM2732A* (13)

General Instruments
28C04 (02)

Harris

HM-6758 (05)

Hitachi
HN462532 (18), *HN462716* (06), HN472732G (12), *HN482732AG* (13)

Intel
2704 (00), 2708 (04), 2708L (04), 2716 (06), 2732 (12), *2732A* (13),
2758 (05), 2816A (11)

Intersil
IM6654 (03), *IM6658* (25)

Microchip
28C04A (02), 28C16A (10), *27HC641* (1F or 21)

Mitsubishi
M5L2716K (06), M5L2732K (12)

MME
U555C (05), *U2716C* (06)

Motorola
MCM2532 (18), *MCM2708* (04), *MCM2716* (06), *TMS2716* (0C), *MCM2732* (12),
MCM68708 (04), MCM68764 (1D), MCM68766 (1D)

National Semiconductor
MM2704 (00), MM2708 (04), *MM2716* (06), *NMC27C16H* (0A), NMC27C32B (17),
NMC27C32H (16), MM2758Q-A (05)

NEC
uPD2716 (06),*uPD2732* (12), *uPD2732A* (13), *uPD28C04*

NTE
NTE2708 (04), *NTE2716* (06)

On Semicondutor
CAT28C16A (10)

Oki
MSM2708AS (04), MSM2716AS (06)

Rockwell
R87C32 (13)

Samsung
KM2816A (0F)

Seeq
2804A (01), *2816A* (0F), *52B13* (11), 52B13H (24), *Seeq 5516A* (0F)

SGS-Thomson (ST)
SGS M2532 (18), *M2716* (06), M2716A-fast (08), M2732A-fast (14)

Sharp
LH5749 (1E), *LH57191* (26)

Signetics
2704 (00), *2708* (04), 2716 (06), *27HC641* (20)

Soviet
573RF2 (06)

Tesla
MHB8708C (04), *MHB2716C* (06)

Texas Instruments
TMS2508 (05), TMS2516 (06), TMS2532 (18), TMS2532A (19), TMS2708 (04), TMS2716 (0C), TMS2732A (1A), *TMS2758-JL0* (05)

Thomson-Mostek
ET2716Q

Toshiba
TMM322 (04), TMM323D (06), TMM323DI (06), TMM2732D (12), *TMM2732DI* (12)

Waferscale Integration
WS57C191 (0D), WS57C191B (0D), WS57C191C (0E), *WS57C291* (0D), WS57C291B (0D), WS57C291C (0E), WS57C43 (1B), *WS57C43B* (1B), WS57C43C (1C), WS57C49B (22), WS57C49C (23)

Xicor
X2804A (01), X2816B (10)