How to make your own ROM-Drive for T1000

The following document is extracted from my own experience of making a new
ROM-drive for my Toshiba 1000 (with DOS 3.3). I've not tried this on any
other models.

For those who remember my post last year containing software to facilitate
making ROM-drives, I've had some reports of trouble with it. Due to the demands of my regular work and the weak response to last year's post, I've
not worked on it any more.

Nevertheless, here is the 'how-to' manual. I hope that it helps some of you
who are still struggling with DOS 2.11 on a T1000.

I have also made ROM-drives for those who don't want or can't do it themselves. E-mail me for more details.

==========

EPROM "ROM-DRIVE" for TOSHIBA 1000

The Toshiba 1000's ROM-DRIVE (drive C:) is an EPROM organized like a

3.5" double density floppy disk. The main difference between the diskette format and EPROM format is that:
i) the EPROM contains a 1K header at the beginning
ii) the EPROM has a maximum file capacity of 504K (512K minus 1K header and standard 7K of directory and FATs), while a 3.5" DD floppy has 713K.

The T1000 comes with a 256K EPROM loaded with DOS 2.11 and some utilities, even though a 512K EPROM can be accomodated. Either is a 32 pin wide DIP IC which can be easily located because it is the only one of this kind on the circuit board. The recommended 512K EPROM is a type 574000 4Meg (512Kx8) type with 150ns access time, while the 256K is a type 572000.

Below, the organization of a 3.5" DD floppy is compared to a 512K EPROM:

EPROM Floppy Disk
Addr (Hex) Addr (Hex)
0 Header 0 Boot Sector
400 Boot Sector 200 1st FAT
600 1st FAT 800 2nd FAT
C00 2nd FAT E00 Directory
1200 Directory 1C00 to B3FFF Files - 713K max.
2000 FILES - 503K max.
- 7FFFF Dummy File - 1K min.

The 512K EPROM contents from 400h to 7FFFF (a 511K block) are essentially directly copied from location 0 to 7FBFF (to end of cluster 1F9) of a 3.5" DD floppy. (For 256K EPROMs, the 255K block from location 400h is copied from disk location 0 to 3FBFF). A header

must then be inserted into the first 1K of the EPROM. Also, the FAT must be changed and a dummy file made to make the 512K (or 256K) EPROM
look like a 713K 3.5" double density diskette.

Note that at least 1K of space must be left at the end for a "dummy" file. Hence, up to 503K of files can be put on a 512K PROM (locations 2000h to 7BFFF), and up to 247K on a 256K PROM (2000h to 3BFFF).

Header File
The first three bytes of the 1K header file denote the EPROM size in reverse order hex. Therefore the first three bytes on a 512K or 256K EPROM are respectively "00 00 80" or "00 00 40". The function of
locations 3 thru 8 are unknown, but the following values have been taken from existing EPROMs:
01 02 03 00 05 14 (512K)
01 02 03 01 02 15 (256K)

Locations 9 and 10 are the last two bytes of the EPROM checksum (with location 10 the LSB), with the checksum taken from location 400h to the end (i.e. all but the 1K header file). Therefore if the checksum is "XXXXXXYYZZ" in hexadecimal, locations 9 and 10 will respectively contain YY and ZZ.
It appears that the rest of the header at least from location 16 (10 hex) can contain anything at all, including comments, documentation, etc.

Dummy File
To make the EPROM look like a full double density disk, a directory entry for a dummy file must be specified with a size which

will completely fill a double density disk. If, therefore, a 512K EPROM has 503K of files loaded, a directory entry must be created assigning a file, say of name "DUMMY", 210K of space to give the appearance that a full 713K of files are present. The attributes of this file are set HIDDEN, READ-ONLY and SYSTEM to make it at least invisible, as it has no purpose to the user.

FAT Modification:
Because the dummy file is assigned space not physically available on the EPROM, its FAT entries are truncated at the last real available cluster on the EPROM. This is cluster 1F9 (for 512K PROMs) or F9 (256K). Doing this will cause a "lost chain" error on performing a CHKDSK, but will otherwise let the dummy file's size entry convince the operating system that the ROM-DRIVE is full while foiling attempts to read nonexistent EPROM locations.

PROCEDURE

The following outlines a practical procedure for creating the binary file to be burned into EPROM. Because the EPROM format is so close to a diskette, the starting point is to prepare a diskette:

1) Disk Preparation:
Format a 3.5" double density (720K) working floppy diskette USING THE SAME OPERATING SYSTEM you want to implement on the PROM.
Install the operating system. Change the diskette label to that you wish for the ROM-DISK label.
Installation of the operationg system is best done by making a working copy of an appropriate boot diskette for your laptop using the DISKCOPY command, and erasing all visible files except

COMMAND.COM on
the working copy.

2) Determine the space left on the working diskette by performing a directory list (DIR). Calculate the remaining space for other files on the 512K (or 256K) PROM by subtracting 215040 (or 477184) from the
space left on diskette. This number is the "free PROM space". The object is to determine how many more files can be written to diskette until 215040 (210K) (or 477184 = 466K) bytes are left on diskette.

3) Load all desired files for the PROM into an empty diskette or hard drive directory. Use this directory ONLY for files which are to go on PROM. You will probably have far more than the PROM can accommodate.
Use a utility like SIZE (PC Magazine utility) to determine how much space the files will take on diskette (Note: File storage is quantized into 1K blocks on diskette, so you can't go by the file sizes as reported in a DIR).
Delete and add files to the dedicated directory until they occupy or nearly occupy the "free PROM space" determined in step 2. Make sure that you include no files or programs that modify or reconfigure themselves (as the PROM is read-only). Make sure that any program which refers to files which are modified (such as configuration files) be set if possible to look for such files outside of the ROM-DRIVE directory.
Alternately, one can add and drop files directly on the diskette until there is at least 215040 bytes (or 477184 for case of 256K PROM) left. In this case, however, once the files to be included are decided on, the disk must be reformatted and the chosen files loaded

on, so that the diskette is loaded contiguously from the first sectors.

4) After, and ONLY after, the dedicated directory is loaded with files with the proper total size, copy all files to the working diskette prepared above. There should be 215040 (477184) bytes or more left on the diskette. After this is done, no changes are to be made to the diskette, as any changes, even if they do not affect the number of bytes, may "fragment" the diskette (i.e. all files must be contiguously stored in the first available diskette sectors).
Write protect the diskette by slipping open the write protect tab. Boot the laptop from the diskette and ensure that all programs and utilities run without problems.
If any changes need to made at this point, erase all visible files on the diskette except visible system files (e.g. COMMAND.COM) and start again at step 3.

5) Once the diskette is confirmed to work, a dummy file must be made just big enough to fill the remainder of the diskette. Any file with size within 1K of the remaining diskette space will do the job. The file can contain anything.
DEBUG may be used to create a file of arbitrary size. If the desired file size in bytes, expressed in hexadecimal, is "AAAABBBB", then a file "dummy" (containing only zeros) can be produced with the following DEBUG commands:

```
n dummy
f 0 f000 ff
rcx
BBBB
rbx
```

AAAA
w 0
q

For example, AAAA=0003 and BBBB=4800 will make a file of size 215040
bytes.
Load the dummy file onto the diskette. The name assigned is arbitrary.

6) Change the attributes of the dummy file on diskette to READ-ONLY, SYSTEM and HIDDEN using a disk editor or by rebooting into DOS 5 and
using the DOS 5 ATTRIB command:
attrib +h +s +r dummy
Use the DEBUG script of the Appendix to read the diskette sectors to binary files. This is done by entering the commands of the Appendix manually in DEBUG, or by loading the commands into a text file, say 'script', and typing:
debug < script

7) For 512K EPROMs, change the FAT entry for cluster 1F9 to end-of-file by changing location 4F6 of file ROM0 to "FF" and the last four bits of 4F5 to "F" (leaving the first four untouched). Do the same to locations AF6 and AF5.
For 256K EPROMs, change terminate the FAT sequence at cluster F9 by changing locations 376h and 976h to "FF" and the last four bits of 375h and 975h to "F".

8) Concatenate the binary files into one binary file (here called 'prombody'):

copy /b rom0 + rom1 + ... romN prombody
where 'romN' is 'rom3' for 256K EPROMs or 'rom7' for 512K. This file should be of size 511K for 512K EPROMS and 255K for 256K EPROMs. The
files rom0, etc. can be discarded at this point. Calculate a checksum by adding all bytes in file 'prombody'.

9) Prepare a 1K header file with the first 11 bytes as described above. Burn the header file into the first 1K of the EPROM (or just the first 11 bytes, as the rest of the header isn't important). Burn the EPROM remainder with 'prombody' from step 8.

INSTALLATION (Jumper Settings)

Looking from the front, jumpers J17, J18 and J19 located near the PROM
socket appear thus:

o o o o o o o o o
1 2 3 1 2 3 1 2 3
JP17 JP18 JP19

For 256K PROM, short 2,3 of JP17, 1,2 of JP18 and 1,2 of JP19.
For 512K PROM, short 1,2 of JP17, 1,2 of JP18 and 1,2 of JP19.

++

Appendix - DEBUG SCRIPTS

1) Extraction of EPROM Data from Floppy Disk B: (512K)

n rom0

```
l 0 1 0 80
rcx
0
rbx
1
w 0
n rom1
l 0 1 80 80
rcx
0
rbx
1
w 0
n rom2
l 0 1 100 80
rcx
0
rbx
1
w 0
n rom3
l 0 1 180 80
rcx
0
rbx
1
w 0
n rom4
l 0 1 200 80
rcx
```

```
0
rbx
1
w 0
n rom5
l 0 1 280 80
rcx
0
rbx
1
w 0
n rom6
l 0 1 300 80
rcx
0
rbx
1
w 0
n rom7
l 0 1 380 78
rcx
0
rbx
1
w 0
q
```

2) Extraction of EPROM Data from Floppy Disk B: (256K)

```
n rom0
l 0 1 0 80
```

```
rcx
0
rbx
1
w 0
n rom1
l 0 1 80 80
rcx
0
rbx
1
w 0
n rom2
l 0 1 100 80
rcx
0
rbx
1
w 0
n rom3
l 0 1 180 78
rcx
0
rbx
1
w 0
q
```