Mzput

Jeroen F. J. Laros

December 11, 2003

Contents

1 Introduction

I have just purchased both an *MZ-700* and an *MZ-800* [3]. I've been looking for one of them for years. At last I found an MZ-800 on an internet site, and a few days after buying this one, someone contacted me (via the same internet site) and sold me an MZ-700. What luck!

2 Doing something with the MZ

So.. I had two obsolete computers. I tested both and carefully stalled the MZ-700. The MZ-800 I connected to my TV and I started to search on the internet for transfer programs. The first thing I wanted was to reliably transfer data and found that the easiest way was to convert an MZF file to WAV and then record this on a tape, which can be read by the MZ. I found a compiled Java program which does just this, but it didn't work under Linux. At this point I probably should have upgraded to the latest Java version, but I thought it would be nice if I wrote my own program. Also the absence of source code made the program virtually useless (at least for my purposes).

2.1 Mzf2raw

Under Windows 95 I converted a few MZF files to WAV with the Java program I found and closely looked at the output. After a few hours of staring at these files, I saw some patterns emerging. Next I wrote a small C program which did exactly the same thing as the Java program. Then I found the fantastic website: www.sharpmz.org, which would have spared me a few hours of staring (the only thing I didn't find there is that each byte is accompanied by a long pulse). At this site I found how the MZ stores it's information on tape. This is significantly different from the way I was using, so I extended my program to make a RAW file which is according to all conventions. I recommend using this option if you want to store a program on a tape permanently, it will store a copy of the header and a copy of the file to make a tape more reliable. It is also possible to vary the speed (but only for the fast conversion mode).
So, the work was done. All I had to do was convert to RAW and then with a program named sox I placed a WAV header in front of it and I had my WAV.

2.2 Mzf2wav

When I mailed my program to the owner of sharpmz.org, he told me that most people use Windows NT. Compiling my code for NT was a piece of cake, however, there was one problem. I don't know of a simple command line tool for NT to convert a RAW file to a WAV. The solution was to do it myself. *Mzf2wav* [6] was born.

2.3 Drawbacks of Mzf2wav

Now I wanted some speed. It is impossible to use speeds exceeding 2 times the normal transfer speed when we use an ordinary sound card. So, we need something faster: the parallel port. I found a program made by a Czech programmer which works under DOS named *TransManager* [1]. It was made entirely in 8086 assembly, so porting this program was out of the question. But the program proved it could be done anyway.

2.4 Later versions of Mzf2wav

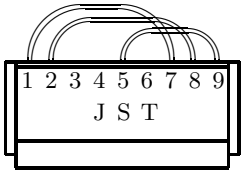
The later versions of Mzf2wav will be modified versions of Mzput. This is because Mzput has far more functionality and the code is more structured.

3 TransManager

This program had one wonderful feature: the turbo sending mode. In this mode first a *turbo loader* [1] is send to the MZ, and then the MZF is send at high speed. Luckily the turbo loader was included in the source and this is the only piece of data I didn't write.

3.1 Getting TransManager to work

I had major problems getting TransManager to work. The documentation was in Czech, a language I do not understand, and the few English lines of information I found didn't help me either. After some investigation I concluded that the program was written for the MZ-711 or 811, on which the external ports are enabled. I've got an MZ-731 and an MZ-821 with a quickdisk. So I needed to convert my MZ-821 to an MZ-811. At this page:
http://www.sharpmz.org/mz-700/usetape3.htm which is a page about upgrading your MZ-711 to an MZ-721 something is mentioned about a 'jumper plug', which, if removed disables the external outlets and enables the sense signal, which is used to display the message: 'Press play'. So now I needed to know how this plug was wired. I contacted Karl-Heinz Mau [3] (the owner of sharpmz.org) and asked him about the plug. This really nice guy was able to tell me how two of the three wires should be wired, but I knew there still was the problem of the sense signal. If I would only short two wires the MZ would still ask me to push play, where there is no play button. So with my multimeter I measured how much current will flow if sense is connected with ground and I decided it was safe. My own jumper plug was born.



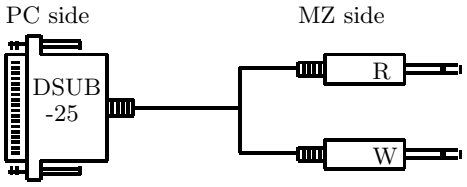
This picture represents a plug similar to the one you find at the other end of your internal tape recorder, it fits over connector P-12 of the motherboard. All you need to do is get such a plug and fit it with three wires:

Jumper plug

MZ	MZ	signal
1	7	write
2	8	read
5	9	ground

Note that you also could solder these wires to make both internal and external tape recorders available, but I prefer this option because I don't want to touch the old MZ, it's far to precious. On the other hand, if you choose to solder the wires, you can also choose to wire the sense signal to the outside.

3.2 The transfer cable



Transfer cable

PC	MZ	signal
3	read tip	read
10	write tip	write
18-25	both sleeves	ground

A cable as described above should work just fine (at least for mzput, I still have to test the other way around).

4 Alternate setup

If you want to connect the data recorder to your PC, you need to make the following cable:

Transfer cable 2

PC	Data recorder	signal
3	7	read
10	8	write
18-25	9	ground
9	4	motor on
-	6	+5V

5 Mzput

When I got TransManager to work, I tried to pull the same trick under Linux. I took my Mzf2wav program and adjusted it to write to the parallel port. This seemed to work at first, but read errors galore! So I tried to fiddle with priorities, a real-time scheduler, but still only one in twenty times the program successfully transfered a header. As you might imagine I wasn't prepared to wait until an image transfered successfully. The only satisfactory option I could think of was also the last option I wanted to explore: putting the time-critical code between CLI and STI opcodes.

5.1 CLI and STI

The CLI instruction is an instruction present in all 8086 based machines, by issuing this instruction you disable all interrupts. Interrupts themselves use this instruction to make sure they won't get interrupted by other interrupts. The STI instruction is the opposite of CLI. Always make sure you re-enable interrupts or be prepared to reboot your machine.
Now, according to conventions, you should minimise the time spent between CLI and STI, which we're obviously violating. Using my program will therefore be entirely at your own risk. I recommend *syncing* [5] your hard disks before running Mzput (although I never do it and it never hurt me, but it seems wise to do it anyway) and I highly recommend running some sort of NTP daemon (or service) to reset the time (Yes! disabling interrupts also means disabling the timer interrupt, so your computer will never know what happened!).
A note for windows users: Make a *system restore point* [4] and copy the data to a directory inside the windows tree (C:\WINDOWS\Restore for example) this will enable you to restore the registry when windows crashes.
Another major drawback is that during transfer your computer will be frozen, so don't be alarmed, it's normal. I've taken every precaution I could think of to make sure transfer will succeed and that control will be transfered back to the operating system, but I might have overlooked something, again: use at your own risk.
Anyway, this approach forced me to rearrange my code (first doing the disk read, then doing the CLI, do the transfer, then the STI. That was all, I thought.

6 The dark side

In my e-mail conversation with Karl-Heinz, porting to other operating systems came to the attention. He told me it was impossible to get this program to work under NT. This had two reasons: NT wasn't a real-time operating system and direct I/O is not possible under this operating system. At first I wasn't planning to port my program to NT at all, I don't even have NT at home; I don't even like NT (hence the title), but I like a challenge. Because of my work I'm no stranger to Microsoft OS's, in fact I've written numerous low-level programs for these OS's, but nothing like this, yet.
The only thing I needed to do was finding a way to get into ring 0.

6.1 Ring 0

The Intel processor has four privilege levels referred to as rings. In both Linux and Windows operating systems two of these rings are used: ring 3 for user applications and ring 0 for the kernel. Under Linux I could issue the CLI and STI instructions just like that, and provided I was logged in as root, it worked. However being root on a Linux box is far more than being Administrator on a Windows box. That's what I found out when I tried to run my program under Windows: a nice little pop-up informed me that a privileged instruction was about to be executed, so the program was stopped. This message shows that the CLI instruction is a 'privileged instruction' and could only be used from within ring 0. In fact all of the operating system dependent functions had to be called from ring 0. I had a problem.

6.2 Callgates

Then the so-called *callgate* [2] came to my rescue. This undocumented feature in the NT kernel is a way to run some code in ring 0. The only application I know using this mechanism are viruses, so my application might well be the first one to use this without being malicious.
The only drawback of using a callgate is that you have to assemble your ring 0 code and the call to this function. This is because all functions called through a callgate must be called with a far call and both gcc (Linux) and cl (Windows) compilers always assemble near calls. So you have to manually push the arguments of the function on the stack. You also have to make sure you return the correct value from ring 0 (I found that when you have three parameters you have to return 0C, with two arguments 08 and so on.) Furthermore you should be very careful with the code in ring 0. I've never seen so many blue screens in my life while debugging my code. But this approach still beats writing a device driver.

7 The extras

Except for being able to run under other operating systems than DOS, Mzput has another big advantage over TransManager. I was experimenting with different waveforms to send to the MZ and I found by experiment that the MZ can recieve a lot faster than I thought. So I started minimising the waveforms until the MZ started complaining, therefore they are not stable, they may not work with your MZ.
Because we are all human, we can make errors, so I've implemented an escape hatch to break out of the transfer routine. This is done by reading the keyboard ports (we can't use interrupts). By pressing the ESC key you can return control to the operating system. Note that the response time may vary between 0 and 7 seconds. This is because I only read the keyboard ports when transferring data, not while writing the gaps.

8 Putting it all together

Mzput can be described as a program that does a real-time transfer of data to the MZ series. It has three operation modes:

8.1 Operation modes

8.1.1 Non-redundant transfer mode

This mode, also referred to as fast sending mode is the simplest way to transfer a file. In this mode the long gap (a gap is a sort of marker) is only 4000 pulses and the short gap is 5000 pulses. Both header and MZF body are transfered once.

8.1.2 Redundant transfer mode

This mode is also referred to as conventional sending mode. This is the safest and slowest way to transfer an image. As far as I know everything is done according to the Sharp MZ series conventions: A long gap is 22000 pulses, a short gap is 11000 pulses. The header and body are transfered twice, so in case of read errors, a copy can be used.

8.1.3 Turbo transfer mode

This mode was not my idea. I found it in the TransManager package, but this is how I implemented it: Fist we send a turbo loader to the MZ using non-redundant transfer mode, then we alter the sending speed, and finally we send the MZF to the MZ also using non-redundant transfer mode. As mentioned earlier: I don't take credit for the turbo loader, I just copied it (and that's about

the only thing I didn't write myself). I did *disassemble* [7] the turbo loader by hand and tried to comment on it, but I couldn't find good documentation, so there are a lot of question marks in the comment.

8.2 Speeds

Five speeds are implemented at this time:

- Speed 0: this is the normal speed with normal waveforms.
- Speed 1: this is 'normal' speed with fast waveforms (see section 7).
- Speed 2: this is 2x speed with normal waveforms.
- Speed 3: this is 3x speed with normal waveforms.
- Speed 4: this is 3x speed with fast waveforms.

These speeds can be set for all operation modes, although most combinations will not work, except in specific cases. The most commonly used speeds are: speed 0 or 1 for normal sending modes and speed 2 or higher for turbo sending mode. If, however you need to transfer multiple images you might want to use speed 3 or 4 with non-redundant transfer mode.

8.2.1 Corrections

Because the waveforms used by speeds 1 and 4 are quite minimal, I suspect they will not always work. So I implemented two corrections for these speeds. You can fiddle with them if you like.

8.2.2 Polarity

I don't know if this is useful, but the -p option reverses the polarity. I never needed it, but since the MZ has one, I figured my program also needed one.

8.2.3 Defaults

Mzput will default to turbo sending mode with speed 0 as initial speed and speed 3 as turbo speed. You can vary the initial speed mode with the -i command line option, the turbo speed mode with the -t command line option and the method with either the -c (conventional), -s (fast) or -w (turbo) options. Furthermore you can correct speeds 1 and 4 with the -1 and -2 command line options.

9 Release

This release includes the source code, a Linux compiled binary and a Windows compiled binary with the necessary callgate.sys and callgate.dll to enter ring 0.

10 See also

If you're interested you might want to take a look at Mzf2wav. It is the mother of Mzput.

11 License

References

[1] The turbo loader was copied from Transmanager, which itself is based on work by Miroslav Nemecek. You can download the software from: http://mzunity.wz.cz/old/Hardware.htm

[2] The callgate mechanism used under w32 is written by Prasad Dabak, Sandeep Phadke and Milind Borate. You can find it at: http://www.sonic.net/~undoc/ntcallgate.html

[3] Additional information about the MZ hardware was provided by Karl-Heinz Mau. He also stimulated me to write a w32 version. See: http://www.sharpmz.org/index.html for almost everything about the Sharp series.

[4] More on creating restore points and restoring from those points: 198.173.130.251/clients/library.nsf/0/8da276c39480a06987256d00008381aa or:
http://www.tek-tips.com/gfaqs.cfm/lev2/67/lev3/70/pid/779/fid/3434

[5] The sync executable can be downloaded from the following site: http://www.sysinternals.com/ntw2k/source/misc.shtml#sync

[6] Mzf2wav (also written by me) can be downloaded from this site: http://www.sharpmz.org/mzfdisass.htm

[7] Mzfdisas (also written by me) can be downloaded from this site: http://www.sharpmz.org/mzf2wav.htm