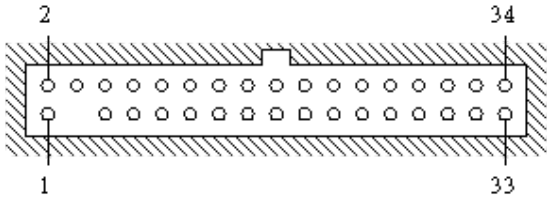
The Great SPARC Floppy Muddle

by Charles Lindsey

This is the story of how I tried to move a floppy drive from my SPARCstation-5 to my new ULTRA-2. This story may not be applicable to all such migrations between SUN machines, but I would expect it to be applicable to many other such situations.

Connections and Pinouts

We start from the Pinout of the (male) socket on the back of the floppy drive:



[A Note about nomenclature: **sockets** appear on the back of drives, and on the motherboard; **plugs** (which fit into sockets) are to be found at the ends of (ribbon) cables; **male**/**female** refer to the pins (and, in the present context, the pins on sockets are always male, and those in plugs are always female).]

PIN 3 is absent on most drives (but not on my SPARC-5 drive, which was a SONY MP-F17W-2PF), and the corresponding hole on the plug is blocked so that you can't put it in the wrong way round. In addition there is often a key slot as shown in the socket, with a matching key on the plug, ostensibly for the same purpose.

All the odd-numbered pins (well, almost all) are connected to GROUND, and it is the even pins which do all the work. The actual Pinout is as follows:

1	GROUND (usually)	2	IN	Density ON=Low OFF=High
3	NO PIN (or GND)	4		Reserved
5	GROUND	6		Reserved
7	GROUND	8	OUT	Index
9	GROUND	10	IN	Motor Enable 0
11	GROUND	12	IN	Drive Select 1
13	GROUND	14	IN	Drive Select 0
15	GROUND	16	IN	Motor Enable 1
17	GROUND	18	IN	Step Direction
19	GROUND	20	IN	Head Step

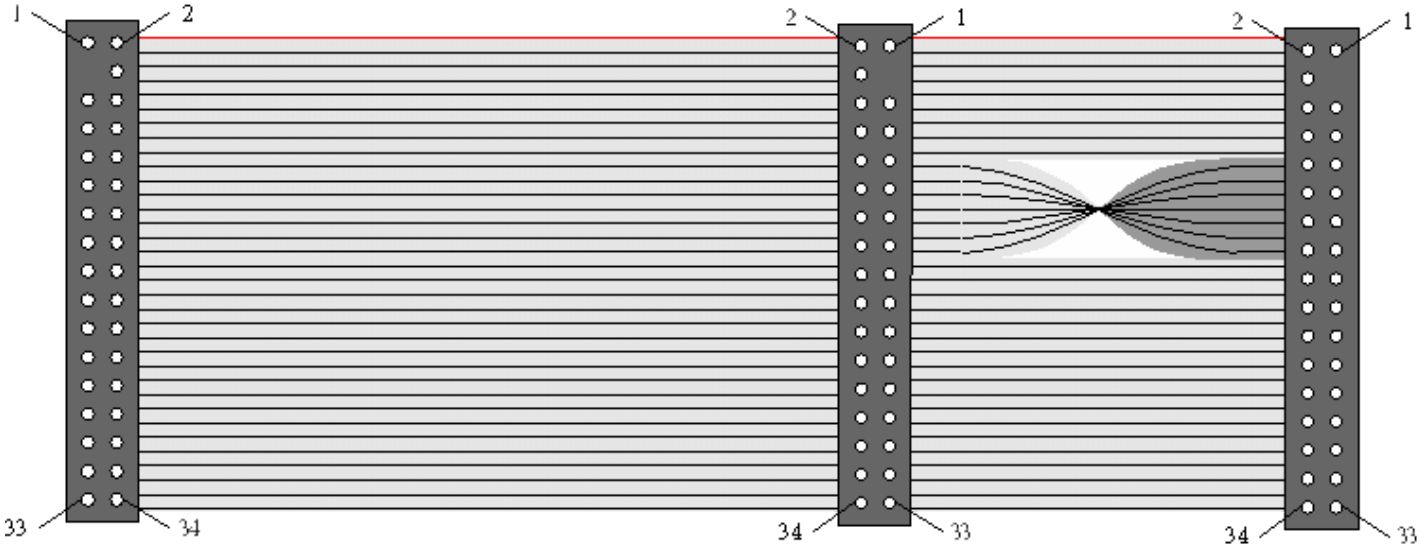
21	GROUND	22	IN	Write Data
23	GROUND	24	IN	Write Enable
25	GROUND	26	OUT	Track 0
27	GROUND	28	OUT	Write Protect
29	GROUND	30	OUT	Read Data
31	GROUND	32	IN	Head Select
33	GROUND	34	OUT	Disc Changed

I suspect that PIN 1 is used for the Eject command on drives with that facility, and consequently it is “not connected” (i.e. not GND) on some other drives. I also believe some later SUN models make use of PINs 4 and 6.

In the PC World, machines have two discs, numbered 0 and 1; hence the two Drive Select lines. The Drive is supposed to know which drive it is (there is usually a switch or a jumper to tell it), and so it doesn't listen on its inputs or speak on its outputs unless the proper Drive Select is ON. Exceptionally, there are two Motor Enable lines, so you can have either, neither or both motors running (provided a disc is loaded, of course).

Signals are TTL levels: ON is GROUND (nominal) and OFF is +5V (nominal).

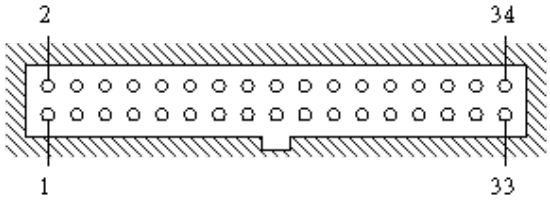
The cable is a ribbon cable with female plugs at each end. Observe that the wire in the ribbon carrying the PIN 1 signal is coloured RED. This is the *only* reliable way to be sure which end of the plug is which. The fashion in the PC world is to have one cable which connects to both drives, with the motherboard plug at one end and two drive plugs (or one for machines with 1 Drive) at the other. Before the last (or only) plug, wires 10 through 16 are separated from their brethren and twisted over, so that wire 10 connects to PIN 16 and wire 16 connects to PIN 10. Thus Drive Selects 0 and 1 are interchanged, and likewise Motor Enables 0 and 1 (the remaining wires being just GROUND anyway). The convention is that both Drives are switched /jumpered to be Drive 1, and hence the Drive at the far end of the cable actually gets selected as Drive 0. Indeed, some modern Drives are permanently wired as Drive 1, and PINs 10 and 14 are not connected to anywhere.



SUN Usage

SUN SPARC machines have only 1 Drive, which is always numbered 0; the ribbon cable has no twists, and so the Drive *must* be switched/jumpered as Drive 0 (and the Field Service Manual warns you about this).

On the motherboard of the SPARC-5 there is a (male) socket just like the one on the back of the drive. So you might suppose that its pinout would be the same as shown previously, and you would be **wrong**. It is actually like this:



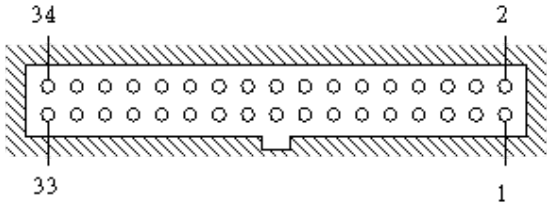
So now the key is the same side as the odd pins (and the ribbon cable is so constructed as to make it all work correctly). And, if you think about it, it does not then matter which end you attach to the motherboard and which to the drive. Observe that PIN 3 always exists on a SUN motherboard, because you are supposed to rely on the key on the side to put the plug in the right way around.

The Sad Story

So, I had a drive (and a cable) in my SPARC-5, and it was working fine. But then I upgraded to an ULTRA-2 which did not have its own floppy, and so I took the drive (and its cable) out of the SPARC-5 and transferred it to the ULTRA-2. And it didn't work . So I put it back in the SPARC-5, and it didn't work there either .

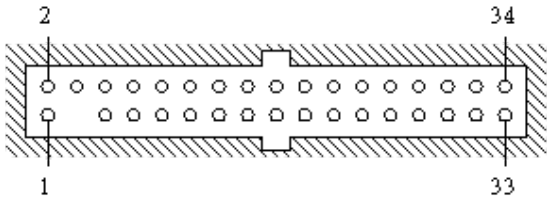
So what had gone wrong? It took me a long time to figure it out, so the following explanation is given with the full benefit of hindsight.

How many ways are there of numbering the 34 pins on one of those sockets? You have already seen two of them, but in act there are four (not counting the ones that rely entirely on PIN 3 for their orientation). And the socket on the ULTRA-2 motherboard is numbered differently to the one on the SPARC-5 (and hence needs a different cable – Part No. 530-2138 as opposed to 530-2067). Here it is:



(observe that the key is still on the same side as the odd pins). So now PIN 1 on the motherboard was connected to PIN 33 on the drive, and PIN 2 on the motherboard was connected to PIN 34 on the drive, and the drive saw all sorts of voltages on various pins that it was not supposed to see, and so it went bust. I am still not sure whether or not it damaged the Intel 82077 floppy controller chip.

So I went out and bought another drive (a SONY MPF 920). This had PIN 3 missing, but SONY were presumably so used by now to dealing with connectors with keys in funny places that it had key cutouts on *both* sides:



So which way to put the cable in? The drive looked somewhat similar to the old drive, so I assumed it would go in the same way (key at the top). And the little green light lit, and '`boot -r`' was convinced there was a drive present. But that was as far as it got. No other signs of life whatsoever. So I put the plug in the other way, and the green light went out, but still no other signs of life.

Debugging the Mess

Now there are lots of websites that purport to tell you how floppies are wired (and hey!, I'm just writing another one). And I also found a data sheet for the SONY MPF 920, and I could then see that many of its PINs were shown as "unconnected" - in particular PIN 10 (Motor Enable 0) and PIN 14 (Drive Select 0). This was a drive permanently wired as Drive 1 (as all good PC drives ought to be) and hence it was never going to work with a SUN machine which could only speak to Drive 0. So back to the shop to cadge a cable with a twist in it.

How many ways of constructing a cable are there? The answer is 72, because, if it has keys then they can have the key on the same side as the odd pins or not, and then you can have the red stripe (WIRE 1) at either end (4 ways so far), or else you can have PIN 3 blocked off in which case there are two ways (mirror images) of where it can be (now we are up to 6). But that is just one end; you still have 6 possibilities at the other end, which get you up to 36. And finally you can have PINs 10 through 16 swapped over, or not. The cable I obtained had no keys, the plugs at two ends were mirror images with PIN 3 blocked, and it had that twist. In fact, it was exactly like the cable I showed earlier on, except that it had just the one plug at the drive end. So, to plug it into the motherboard it was necessary to drill out the PIN 3 hole (and ditto at the drive end for further experimentation with the old drive).

It was clear which way it had to be plugged into the drive (red strip to the left according to the data sheet, which indeed brought the twisted wires to the right place). That left two ways to plug it into the motherboard. I tried both, and neither worked. (In passing, it may be noted that the ULTRA-2 Service Manual contains an exploded diagram showing how to fit the floppy cable, and you can follow what should be the red edge through the various twists and turns to see which end of the motherboard socket it should go to; but I misread that diagram, though on rereading it I can now see that it is impossible to insert the original SPARC-5 cable in the right way.)

Time to RTFM! In particular the **man** page for `fdio(7I)`, which gives various `ioctl`s of which the most interesting is `FDRAW`, which gives you access to (some of the) raw interface of the Intel 82077. But that **man** page refers you to the floppy-controller data sheet for the details of what all the commands do, so you have to download that from 82077AA CHMOS SINGLE-CHIP FLOPPY DISK CONTROLLER. I then wrote a program `fdfix` to issue that and various other `ioctl`s. You can find it here if you want to experiment further, though I never implemented all the available raw commands.

That showed that it thought the disc was Out, Track0 selected and Write Protect set. But then it said the same thing whether the disc was in or out, or whatever I did to it. So the next move was to download the source code of the driver `fd.c` from www.opensolaris.org, in order to figure out what those `ioctl`s actually did (oddly, all the `.h` files used by `fd.c` are already available in `/usr/include/sys`, even though many of them are peculiar to `fd.c`). And this revealed several interesting things, such that the floppy controller itself was subject to power management, and was powered down after 15 minutes of idleness, and that every operation on the disc drive starts off by turning the motor on, and automatically turns it off after 6 seconds of idleness. But I also saw that the motor in the drive was actually turning full time whenever there was a disc in it. Something was seriously wrong.

Time to trace was was actually happening within the driver. I started off using

```
dtrace -F -m fd
```

which was quite useful, but then I found an even better way of tracing. `fd.c` contains numerous calls on `FDERRPRINT`, which is actually a macro defined in `fdvar.h`, and which prints out on `/dev/console` useful diagnostics with verbosity depending on `fderrlevel` (you *do* keep a Console Window permanently available on your desktop, don't you?). So how to set that variable? There is alleged to be an `ioctl IOCTL_DEBUG`, but `fd.c` appeared to have been compiled without it, and I cannot find it in any `.h` file anywhere. So `mdb` to the rescue. (Kiddies! Don't do this at home without your parents watching; it can do serious harm to your Operating System!)

```
mdb -KF
```

That brings up `kmdb` on your *real* console (less harm of mucking up your desktop). You proceed as follows:

```
Welcome to kmdb
> fderrlevel/D
fd'fderrlevel:      3
> fderrlevel/W 1
fd'fderrlevel:      0x3             =       0x1
```

```
> fderrlevel/D
fderrlevel:     1
> ::quit -u
```

So `3` is the normal value of `fderrlevel` (nothing but occasional error messages to the Console), `1` is its debugging level (and there is also a less verbose level `2`). The `quit` (don't forget the `-u`) returns you to your desktop, after which you should `refresh` your screen.

Now I could see just which bits of `fd.c` it was going through, and it was clear that it was telling the drive to do all the right things but the driver just was not listening; indeed the drive was behaving as if all its inputs were ON (and for that matter the chip was behaving as if all the drive's outputs were ON too. Time to power down and try the plug in the motherboard the other way round; whereupon it refused to recognise that the drive even existed; which was odd because I had just done a `boot` (*not* a `boot -r`), but in spite of that, and even though `/dev/diskette` was still a soft link to `../devices/sbus@if,0/SUNW,fdtwo@f,1400000:c`, that entry in `/devices` had simply vanished. `boot` (without `-r`) is simply **not** supposed to behave in that way.

Well, temporarily reversing the plug at the drive end and doing '`devfsadm -i fd`' fixed that, but it still didn't work (the most obvious difference being that the green light on the drive was now permanently Out).

Time to inspect the signal actually on the wires. Fortunately, the 82077 data sheet claimed that its TTL outputs would deliver up to 40mA, which was plenty for my ancient genuine moving-coil voltmeter which never draws more than 1mA. So, using that, I was able to see that all the odd-numbered pins on the cable (except PIN 1) were showing GROUND and most of the even-numbered pins were showing +ve (i.e. OFF). And the even-numbered pins on the drive (those that were connected) were showing sensible values (e.g. Track0, Write Protect, Disc Changed). Moreover, when I issued any command I could make the Motor Enable 1 on the cable go to 0V (i.e. ON). Signs of life at last!!!

BUT Drive Select 1 was resolutely set OFF. Why should that be? Had I damaged that output from the chip during my earlier mis-cablings? Was that pin on the motherboard end actually connected to the proper pin on the chip? (Yes, I checked that the cable had no breaks on that wire.) So maybe the drive on an ULTRA-2 was supposed to be switched/jumpered so as to be permanently selected (I have no knowledge of whether some drives provide that option or not). For sure, `fd.c` was obeying code that was supposed to switch that pin ON (actually, the code in `fd.c` switches between selecting Drive 1 - which I see as Drive 0 because of the twist in the cable) - and Drive 0, but since it knows that Drive 1 does not exist, that should amount to the same thing). So I had to insert a thin piece of wire into PIN 12 of the Drive end of the cable and connect it to GROUND via a suitable resistor to force the Drive to be selected.

EUREKA! Now the green light was on, the motor started when instructed, and if I issued a command to move the head to some other track, Track0 went out, and came back again when I reselected track 0. So I then went all daring, switched volume management back on again, inserted a disc and *voila!* It recognised the disc and I could read its contents. It was a bit reluctant to read some other discs written on the old drive, but it was happy to reformat them, and then they were OK.

That just left the problem of the green light, which was now permanently lit. So I modified the thin piece of wire so that it joined PINs 12 and 14 together, and now Motor Enable also acts as Disc Select (which is quite enough to keep `fd.c` happy), and it means that the green light now comes on whenever the disc is actually doing something, which is quite handy.

Is the Disc IN or OUT?

Remember that the only signal sent from the Drive is Disc Changed. Whenever you insert a disc, Disc Changed goes ON, but it goes OFF again at the next operation (e.g. a `seek`). It also goes ON when you remove a disc, but in that case it stays ON even after the next (attempted) operation. So there is no easy answer to this question.

Programs that want to establish whether the disc is IN or OUT call the `ioctl FDGETCHANGE`, which first looks at Disc Changed and, if it is OFF immediately reports that the disc in IN. But if it finds Disc Changed OFF, it has to perform an experiment. It sends a `seek` command to move the head (you can actually move the head even when the there is no disc in the Drive). If, after that, Disc Changed is still ON, it knows that the disc is OUT; otherwise, it is IN.

But there is a BUG (now reported on www.opensolaris.org as Bug Id #6698413). If you have Power Management enabled (`autopm enabled` in `/etc/power.conf`, usually with a `system-threshold` of 15 minutes), and it has timed out (removing power from the 82077 chip, and possibly from the drive also), and you then send that `ioctl`, it has to wake up. So it selects the drive, turns on the motor, and sends commands to configure the Drive, including sending the head back to Track0, after which it deselects the Drive (see the end of `fd_pm_raise_power` in `fd.c`). Actually, the 82077 chip cannot deselect a Drive – it can only choose which one is selected – so what `fd.c` actually does is to select Drive 1 (which it knows does not actually exist). BUT it forgets to select the Drive again before inspecting Disc Changed, so naturally the (unselected) Drive reports that it is OFF and the `ioctl` duly reports that the disc is IN even when it is actually OUT . {Oddly, even with my piece of wire between PINs 12 and 14 forcing the disc to be selected whenever its motor was ON, the chip seemed to know that "disc 1" (as it thought) could not possibly be selected since it knew that "motor 1" was OFF.}

To demonstrate the bug, try doing

```
volcheck -v /dev/diskette
```

when there is no disc in the Drive and it has not been used for 15 minutes.

To Eject, or not to Eject?

There are two properties which the driver claims to know about the disc: whether it is `pollable` and whether it is `ejectable`. How does it know? There is absolutely no signal sent from the disc to tell you. Actually, `pollable` is not of great interest, because apparently it only arises on the Sparcstation Voyager. But `ejectable` is another matter. It arises from the "properties" associated with the driver, which mostly arise from the main computer Prom on the motherboard. In fact, that PROM is silent on the matter and so, in the absence of information to the contrary, the default setting applies, which is `ejectable` (because SUN *knows* what kind of drive it supplies with each of its machines, and nobody would be so foolish as to fit any non-SUN drive, would they?). Of course, my new SONY drive was non-ejectable; the Bad News is that no SUN documentation I have encountered tells you how to change that property; the Good News is that you can deduce it from a study of the source code `fd.c`. As `root`, you create the following file in `/platform/<platform-name>/kernel/drv/fd.conf` (where `<platform-name>` in my case was `sun4u`):

```
# Hack to make floppy non-ejectable
manual=1;
```

then you either reboot or, as `root`, do:

```
/etc/init.d/volmgt stop
update_drv -v fd
/etc/init.d/volmgt start
```

and you can then do

```
prtconf -v /dev/diskette
```

to check that the `manual` property has indeed been set. But what benefit does this bring you?

The Volume Manager is a strange beast. It thinks it *knows* when there is a Disc in the Drive without needing to look (for if the Disc that was known to be there has not yet been `ejected`, then surely it must still be there). When you first put a Disc in, you might think it would notice it (as it does when you insert a Cdrom). But No! A change on the Disc Changed PIN does *not* cause an interrupt, so nothing happens until you call `volcheck`, at which point it *does* have a look (modulo the Bug described earlier), and duly proceeds to mount the Disc if its format is recognized. So now you can see the result in the following places:

```
/floppy/floppy0
/floppy/foo
/vol/dev/aliases/floppy0
/vol/dev/diskette0/foo
/vol/dev/rdiskette/foo
```

```
/vol/dsk/foo
/vol/drdsk/foo
```

where `foo` is the Volume Name of the Disc (or `unnamed_floppy` if no Volume Name). What you see in `/floppy` is a Top-Level Directory which leads to all the files on the Disc (`floppy0` is just a soft link to `foo`). All the things you see under `/vol` are Special Devices (character special for the raw versions and block special otherwise); you can open them if you wish and explore the Disc untrammelled by its file structure (as indeed is done by my `fdfix` program). If the format of the Disc cannot be recognized, the `/floppy` directory will remain empty, and `foo` will be `unknown_format`.

But now if you take the Disc out and try `volcheck -v`, it will tell you that media is still present without even bothering to look. Even if you put in a different Disc and do another `volcheck` it will still think the old Disc is in, and it will still try to show you the old contents of `/floppy` (unless you do something that involves actually reading the Disc, in which case it will pop up a window asking you to put the old Disc back in, and Woe Betide if you try to fool it with the new one).

The **only** way to make it see sense is to call `eject` (or preferably `eject floppy`, since if there is no Disc there it cannot tell what you want ejected – or worse, if a Cdrom is present it will try to eject that). It matters not whether the Drive is `ejectable` or not. The *only* way to make the system understand that you have finished with that Disc is to call `eject [floppy]`, and it won't let you do even that if some process still has some file open beneath `/floppy` (in which case you might have to do `fuser /floppy` to find out which – not forgetting that one of the processes it then shows you will be `fuser` itself).

So what difference does it make whether the Drive is `ejectable` or not? Answer: very little. You still have to call `eject` before you remove it (or `eject floppy` afterwards). In fact the *only* difference it makes is that, if it is officially made `manual` (see above), then it will be kind enough to pop up a message saying

```
/vol/dev/rdiskette0/foo can now be manually ejected
```

whether you had already done so or not. And as a further Bug^H^H^H Feature, if you have two screens on your display (or, worse, more than one display), it always pops up that message on Display `:0,0`, rather than the Display/Screen where you issued the `eject`.

And for good measure it also puts out the same message on `stderr` and causes `eject` to return '4'. The latter is a nuisance because, if you caused the ejection by clicking `eject` in the File menu of your File Manager, it pops up a 'Watch Errors' window to exhibit that message, so you now have two popped-up windows to get rid of. I therefore recommend the following in `/etc/dt/appconfig/types/C/dtfile.dt`:

```
ACTION Eject
{
        LABEL           Eject
        ICON            SDtRM.efp
        ARG_COUNT       1
        TYPE            COMMAND
        CWD             /
        WINDOW_TYPE     NO_STDIO
        EXEC_STRING     dtksh -c 'dir="%Arg_1%"; \
                if [[ ! -d "\$dir" ]] ; then \
                        dir="`dirname \\"\$dir\\"`"; \
                fi; \
                cd "\$dir"; \
                media="`basename \\"\$dir\\"`"; \
                cd ..; \
                dir="`pwd`"; \
                until [[ "`dirname \\"\$dir\\"`" = / ]]; do \
                        media="`basename \\"\$dir\\"`"; \
                        cd ..; \
                        dir="`pwd`"; \
                done; \
```

```
                    if eject "\$media"; \
                       [ $? -eq 0 -o $? -eq 4 -a "\$media"=floppy ]; \
                    then :; \
                    else exec dtaction DttermErrorlog; fi'
            DESCRIPTION     The Eject (Eject) action ejects the media \
                            containing the file dropped on it.
    }
```

But, even with that, if you normally have an iconized ' Watch Errors' window sitting on your desktop (which is handy for spotting CDE funnies as they happen), it will still pop that one up to show you that message. So if, at the end of the day, you decide that making your Drive officially `ejectable` is not worth all that hassle, considering the small benefit achieved, then I would not blame you in the least.

Hacksaw Work

So, having got it all working, time to put the Drive back in permanently (you didn't think that I had been doing all that fiddling around with the Drive screwed into the frame it shares with the Cdrom and that frame screwed into the machine, did you? No! It was all done with the cover off and the Drive carefully balanced on the top – but be sure that the cover is only pushed back rather than removed entirely, so that the cooling air still flows correctly.)

Now, since SUNs are intended to take `ejectable` drives, the slot in the frame (both the metal inner and the plastic outer) is designed to take `ejectable` drives only, so if you try to mount a Drive with an Eject button on it, there is no hole in the frame for that button to poke through (nor is there any hole to let you see the little green light). And fixing that does require a hacksaw and a drill. But once that has been done, then it all comes together nicely.