#XDTN

This tech note is about possible data corruption when using external drives

External drive data corruption during power sequencing

When the Portfolio is reading from or writing to an external drive (e.g. ZIP, ATA or Compact Flash), if it powers down in the middle of a disk access then data may be corrupted. This is because the associated device driver has configured the drive via various IO writes, and a power down/power up sequence can leave the drive in an un-configured state temporarily. Even if the associated driver code configures all necessary IO just prior to every read or write, a power down could occur just after all that configuration and before accessing the disk. This Tech Note outlines some possible solutions to this issue.

Note that it's not a concern for internal drive C: or memory card drives A: or B: as these drives consist of SRAM. If a power down occurs in the middle of a repeated string (e.g. REP MOVS) read from or write to SRAM, following a power-up the remainder of the operation is executed correctly.

One remedy is for a driver to use Int 61h Fn 1Ch to preset IO data, where associated IO address and data pairs are written out during a power up. This can only be implemented by someone who understands which IO configurations need to take place following a power up. If more than a few IO writes are required, the XMEM vector (see Int 61h Fn 32h) can be used to execute configuration code. However, this vector is normally in use by UPDATE.COM, so it would need to be hooked by the IO configuration code which must exit by jumping to any previous vector.

The Portfolio can power down for the following 4 reasons:

- An inactivity timeout, while the BIOS is waiting for a keypress. This occurs semi-asynchronously (within the NMI ISR)
- A low battery, unless the BIOS has been configured by Int 61h Fn 26h not to do so. This occurs asynchronously.
- A user requests a power down by typing <Fn><o>
- A user requests a power down by typing OFF at the command line, or OFF is processed within a batch file. This invokes Int 61h Fn 2Dh to command a power down, which can also be invoked by a utility program.

At the BIOS level, a low battery power down can occur within the following interrupt service routines (ISRs):

- NMI (this is so the battery is checked at least once every 128 seconds)
- Int 9h (every time the Return key is pressed)
- Int 13h (after every Read, Write and Format operation)
- Int 61h (after tone generation and dialling, or prior to sounding the alarm)
- Int 4Ah (prior to generating an alarm sound, while inside the NMI ISR).

UPDATE.COM hooks Int 8h (a proxy for the NMI), Int 9h and Int 61h Fn 2Dh, and those ISRs save the byte at RAM address 8030h. This is necessary as the byte is incorrectly set to 0 by the hardware when power is switched off.

UPDATE.COM also needs to do the same for Int 13h, Int 61h Fn 15h (Alarm), 16h (Tone generation) and Fn 17h (Dial number). If data is being copied between an external drive that consumes a lot of power and one of the internal drives, then an Int 13h power-down is more likely to occur.

The inactivity timeout with the NMI service routine checks a System-Active flag in the BIOS data area:

- If the flag is set (by a key press), then the countdown timer is reloaded.
- If it's clear, then the countdown timer is decremented. If the count value reaches 0, then a power down is invoked.

However, only a key press sets the System-Active flag. Any other activity, such as a disk access, doesn't. As this wasn't a requirement for the Portfolio's local drives…

While the last 2 power down reasons listed above are under user/program control, a user may be unaware that a drive is being read or written. This is the case where an application occasionally accesses the drive, especially where data is being sampled over a period of time and saved to disk. So it's possible that a user-commanded power down can unknowingly be made in the middle of a drive access.

Whatever the trigger for a power down, there is one common power down routine in the BIOS. This routine's code execution does not allow for any hooks that could intercept the process, apart from a call to detect and execute a power-down ROM extension.

Possible solutions:

1. The System-Active flag bit can be set by an external driver prior to every drive access, thereby ensuring an inactivity-triggered power down does not occur. This is the simplest solution to avoid a power down due to perceived inactivity, but it does not cater for the other power-down causes. Details provided below.

2. Int 61h Fn 26h can be invoked to set the power state to prevent a power down due to a low battery. Following a drive access, the power state can be restored and the next NMI will check for a low battery. Int 61h Fn 27h can be used to check for a low battery, however this service provides status only and does not initiate a power down.

3. In order to address the situation where a user requests a power down without realising that the external drive is being accessed.

   - Int 61h Fn 2Dh can be hooked, so as to defer a power down requested via the OFF command or programmatically.

- However, there is currently no easy way to detect and thereby prevent (or defer) <Fn><o>, as this key sequence is processed and the power down initiated within the Int 9h ISR.

4. A more comprehensive approach is to implement a power down ROM extension on a memory card. This works irrespective of how the power down was invoked, so caters for all possible scenarios.

   The ROM extension can set a flag indicating that a power down is pending, and resume normal operations. Following any drive access, the power down pending flag can be checked and if set a power down can then be commanded.

Setting the System-Active bit

The table below shows the location of the byte containing the System-Active bit for different ROM versions:

1.030	40:122h
1.052	40:122h
1.07X	40:125h
1.13X	40:125h

A device driver can set this bit on entry as below, to reset the inactivity timer.

```
mov     bx,40h                  ;Access BDA via DS
mov     ds,bx
or      byte ptr ds:122h, 80h   ;For ROM versions 1.030 & 1.052
```

To clear the bit prior to exit:

```
mov     bx,40h                  ;Access BDA via DS
mov     ds,bx
and     byte ptr ds:122h, not 80h   ;For ROM version 1.030 & 1.052
```

This approach only prevents the Portfolio from powering down due to perceived inactivity.

Implementing a CCM ROM Extension to defer a power-down

A ROM extension can be implemented on any linear memory (SRAM or ROM) device mapped into the C0000h mapping frame. This includes drives A: & B: as well as an Expansion ROM.

A ROM extension can co-exist with a boot sector on a CCM, so that the CCM can be used as a disk as well. The ROM extension vector locations are positioned to allow that.

The Extension ROM boot sector “XRBS” shown in the listing attached uses one of the 16 bytes in the BIOS Inter Applications Communications Area (IACA) starting at address 4F0h in System RAM. This

area was defined by IBM for the IBM PC, but with no way to guarantee that a byte was not relied on by another application, it was normally unused. Address 4F6h has been selected, in case the first 6 bytes are used elsewhere. Two bits are defined for this IACA byte, bit 7 (80h) ACTIVE indicates that a drive is active and bit 6 (40h) PENDING indicates that a power down is pending.

The IACA byte should be initialised by a device driver's initialisation routine, otherwise it will still be in a previous state following a warm boot.

The device driver can set the ACTIVE bit on entry. When a power down is initiated, if the ROM extension code in the XRBS sees that the ACTIVE bit is set it can prevent the power down from occurring and will set the PENDING bit in the IACA byte. Prior to exiting, the driver should clear the ACTIVE bit. It should then check if the PENDING bit is set, indicating that a power down is pending, and if so clear it and request a power down. As ACTIVE is no longer set, the XRBS ROM extension will allow the power down to execute.