Erledigt LENOVO IDEAPAD 14 DSDT Battery Patch

Beitrag von „grt“ vom 11. März 2019, 18:46

anleitung von rehabman bei den tomaten gefunden.

ist eigentlich nicht wirklich schwer.

im ec-device gibt es diverse speicheradressen (sitz am händy, codebeispiel könnte ich nachher mal posten) in unterschiedlichen grössen:

operation region "irgendwas", dann folgen die zeiger auf speicherbereiche:

"abcd", "eine zahl" wobei das erste eine bezeichnung ist, die zahl die grösse des speicherbereichs in bit.

darin werden informationen gespeichert und ausgelesen.

gemeinerweise erwartet nun osx, dass die infos immer in 8bit häppchen serviert werden, und das tut die dsdt eines pc-laptops nicht. wenn du dir die werte ansiehst, die hinter den bezeichnern stehen, siehst du, dass da von 1 bis 256 stehen kann...

und da muss man eingreifen. man teilt die grösseren bereiche auf (vorher prüft man, ob der speicherbereich überhaupt genutzt wird, wenn nicht, kann man ihn so lassen, wie er ist) in osx-mundgerechte 8bit "schnipsel"

beispiel: da steht abcd, 16 - beim auskommentieren gibts einen error, abcd wird im verlauf des codes verwendet.

also zelegt man abcd in abc1, 8 und abc2, 8 und hat nun 2 8bit schnipsel aus dem ursprünglichen 16 bit bereich gemacht. der ursprüngliche abcd, 16 wird gelöscht oder auskommentiert.

an der stelle, wo abcd weiterverwendet wird, muss man nun die beiden "schnipsel" verwenden, statt des ganzen in einem stück. dafür gibt es eine methode, die eingefügt und angewendet werden muss, die das kann.

für grössere bereiche 32bit oder 128bit geht man genauso vor, und wendet die dafür vorgesehenen methoden zum zusammenfügen der information aus 32 oder 128bit speicherbereichen an.

ist ziemlich viel schreibkram und man muss sich mächtig konzentrieren, sonst kanns auch mal passieren, dass batterieanzeige oder lüfterdrehzahl wie gewürfelt aussehen.

EDIT:

hier mal ein beispiel aus der "richtigen" praxis - kommt vom lenovo W520:

Code

```
1. //fix: 8-bit
2.
3. Field (ECOR, ByteAcc, NoLock, Preserve)
4. {
5. Offset (0xA0),
6. //fix: 8bit SBBM, 16,
7. //fix: 8bit SBMD, 16,
8. //fix: 8bit SBCC, 16
9. SBB1, 8,
10. SBB2, 8,
11. SMD1, 8, //split
12. SMD2, 8, //split
13. SCC1, 8, //split
14. SCC2, 8 //split
15. }
```

Alles anzeigen

die ursprünglichen 16bit speicherbereiche aufgeteilt in je 2x 8bit -> SBBM, 16 zu SBB1 und SBB2, originalcode auskommentiert

beim aufruf von SBBM werden nun statt des (nicht mehr vorhandenen) SBBM mit der methode B1B2 die beiden teilinformationen in der richtigen reihenfolge übergeben:

Code

```
1. //fix: 8bit Store (SBBM, Local7)
2. Store (B1B2 (SBB1, SBB2), Local7)
```

die methode B1B2 muss auch eingefügt werden, ich mach das meist am anfang der DSDT:

#

Code

```
1. //16 zu 8-bit
2. Method (B1B2, 2, NotSerialized)
3. {
4. Or (ShiftLeft (Arg1, 0x08), Arg0, Local0)
5. Return (Local0)
6. }
```

dasselbe für 32bit speicherbereiche:

SBCH aufgeteilt in SBC1 - SBC4

und aus

Code

```
1. Store (SBCH, BTYP)
```

wird

Code

```
1. Store (B1B4 (SBC1, SBC2, SBC3, SBC4), BTYP) /* \_SB_.PCI0.LPC_.EC__.GBIF.BTYP */
```

B1B4 muss ebenfalls eingefügt werden, damit sie aufgerufen und ausgeführt werden kann:

Code

```
1. //32 zu 8-bit
2. Method (B1B4, 4, NotSerialized)
3. {
4. Or (ShiftLeft (Arg1, 0x08), Arg0, Local0)
5. Or (ShiftLeft (Arg2, 0x10), Local0, Local0)
6. Or (ShiftLeft (Arg3, 0x18), Local0, Local0)
7. Return (Local0)
8. }
```

last but not least gibt es auch noch eine L1L4 für 128bit zu 8bit, sowie eine methode speicherbereiche, die "krumme" grössen haben, oder grösser als 128bit sind direkt schreiben und lesen zu können, ohne dass sie sozusagen "benamt" worden sind. die sind dann so richtig eklig....