HowTo: Thunderbolt HotPlug/HotSwap Finetuning für euren Hackintosh

Beitrag von „apfelnico“ vom 4. September 2019, 10:28

> Zitat von Mork vom Ork
>
> Letztlich habe ich nicht getestet, ob es nun nur Dank der neuen BIOS Einstellungen funktioniert, oder ob auch die "NOTIFY"-Funktion ihren Teil dazu beiträgt.

Im Original https://osy.gitbook.io/hac-min…details/thunderbolt-3-fix wurde beschrieben, welche "_Exx" Funktion (Edge Triggered GPE) ein "Notify" für genau den benutzten Slot beinhaltet. Bei mir ist es "_E4C". Eine Methode Namens "NTFY" wurde ermittelt, diese wird in der DSDT benutzt. Entweder man editiert diese direkt in der DSDT, oder man versucht diese über Clovers DSDT-Patches durch Rename unbrauchbar zu machen. So sieht diese bei mir zum Beispiel aus:

Code

```
Scope (_GPE)
{
Method (NTFY, 0, Serialized)
{
If ((NOHP == One))
{
Switch (ToInteger (TBSE))
{
Case (One)
{
Notify (\_SB.PCI0.RP01, Zero) // Bus Check
}
Case (0x02)
{
Notify (\_SB.PCI0.RP02, Zero) // Bus Check
}
Case (0x03)
{
Notify (\_SB.PCI0.RP03, Zero) // Bus Check
}
Case (0x04)
{
```

```
Notify (\_SB.PCI0.RP04, Zero) // Bus Check
}
Case (0x05)
{
Notify (\_SB.PCI0.RP05, Zero) // Bus Check
}
Case (0x06)
{
Notify (\_SB.PCI0.RP06, Zero) // Bus Check
}
Case (0x07)
{
Notify (\_SB.PCI0.RP07, Zero) // Bus Check
}
Case (0x08)
{
Notify (\_SB.PCI0.RP08, Zero) // Bus Check
}
Case (0x09)
{
Notify (\_SB.PCI0.RP09, Zero) // Bus Check
}
Case (0x0A)
{
Notify (\_SB.PCI0.RP10, Zero) // Bus Check
}
Case (0x0B)
{
Notify (\_SB.PCI0.RP11, Zero) // Bus Check
}
Case (0x0C)
{
Notify (\_SB.PCI0.RP12, Zero) // Bus Check
}
Case (0x0D)
{
Notify (\_SB.PCI0.RP13, Zero) // Bus Check
}
Case (0x0E)
{
Notify (\_SB.PCI0.RP14, Zero) // Bus Check
}
```

```
65. Case (0x0F)
66. {
67. Notify (\_SB.PCI0.RP15, Zero) // Bus Check
68. }
69. Case (0x10)
70. {
71. Notify (\_SB.PCI0.RP16, Zero) // Bus Check
72. }
73. Case (0x11)
74. {
75. Notify (\_SB.PCI0.RP17, Zero) // Bus Check
76. }
77. Case (0x12)
78. {
79. Notify (\_SB.PCI0.RP18, Zero) // Bus Check
80. }
81. Case (0x13)
82. {
83. Notify (\_SB.PCI0.RP19, Zero) // Bus Check
84. }
85. Case (0x14)
86. {
87. Notify (\_SB.PCI0.RP20, Zero) // Bus Check
88. }
89. Case (0x15)
90. {
91. Notify (\_SB.PCI0.RP21, Zero) // Bus Check
92. }
93. Case (0x16)
94. {
95. Notify (\_SB.PCI0.RP22, Zero) // Bus Check
96. }
97. Case (0x17)
98. {
99. Notify (\_SB.PCI0.RP23, Zero) // Bus Check
100. }
101. Case (0x18)
102. {
103. Notify (\_SB.PCI0.RP24, Zero) // Bus Check
104. }
105. Case (0x1A)
106. {
```

```
Notify (\_SB.PC01.BR1A, Zero) // Bus Check
}
Case (0x1B)
{
Notify (\_SB.PC01.BR1B, Zero) // Bus Check
}
Case (0x1C)
{
Notify (\_SB.PC01.BR1C, Zero) // Bus Check
}
Case (0x1D)
{
Notify (\_SB.PC01.BR1D, Zero) // Bus Check
}
Case (0x1E)
{
Notify (\_SB.PC02.BR2A, Zero) // Bus Check
}
Case (0x1F)
{
Notify (\_SB.PC02.BR2B, Zero) // Bus Check
}
Case (0x20)
{
Notify (\_SB.PC02.BR2C, Zero) // Bus Check
}
Case (0x21)
{
Notify (\_SB.PC02.BR2D, Zero) // Bus Check
}
Case (0x22)
{
Notify (\_SB.PC03.BR3A, Zero) // Bus Check
}
Case (0x23)
{
Notify (\_SB.PC03.BR3B, Zero) // Bus Check
}
Case (0x24)
{
Notify (\_SB.PC03.BR3C, Zero) // Bus Check
}
```

```
149.  Case (0x25)
150.  {
151.  Notify (\_SB.PC03.BR3D, Zero) // Bus Check
152.  }
153.
154.  }
155.  }
```

Alles anzeigen

Um dann in der Folge in einer eigenen SSDT die Funktion neu zu erschaffen, mit an geeigneter Stelle modifizierte Notify, die bis zum NHI0 reicht. Sinnvoll wäre es natürlich, auch die restlichen unbearbeiteten "Cases" wieder aufzunehmen. Das kann also je nach System ganz anders aussehen, hier war es ein NUC. Und wie gesagt, deine "NTFY" hängt an "PXSX", welches du gerade "unschädlich" gemacht hast. Wenn es wichtig wäre, wäre es besser im aktiven Strang "UPSB" aufgehoben. Noch besser jedoch das Original ausgetauscht unter "Scope (_GPE)".

> Zitat von Mork vom Ork
>
> Und mit dem ganzen UPSB/DSBx Wust hatte ich ja auch geschrieben, das ich es in der Tutorial-SSDT bewusst weggelassen habe, da ich dadurch in einem längeren, hintereinandergeschalteten Strang an TB-Devices das
>
> Problem hatte, das ich die Kette dann nicht an jeder x-beliebigen Stelle unterbrechen konnte, ohne das mein Rechner umgehend einen Neustart gemacht hat. Habe ich die selbe Kette aber unter Nutzung der im Tutorial
>
> besprochenen SSDT genutzt, konnte ich die Kette an jeder x-beliebigen Stelle trennen und ggf. neu stecken.
>
> Das dadurch die bequeme Möglichkeit der Benennung der Devices mittels der _DSM-Methode verloren geht, habe ich zu Gunsten der Funktionalität ersteinmal in Kauf genommen.

Das habe ich ja verstanden und auch als richtig geschlussfolgert kommentiert. Nur ist es sinnvoll, (nicht den kompletten Strang!), sondern die ursprünglichen "DSBx" (nicht nur "DSB0") ebenfalls mit aufzunehmen (schaue doch dazu in meine überarbeitete Version). Es ändert sich somit nichts mit dem "Kettenproblem"! Aber da hier bereits nun schon eine _DSM-Methode gesetzt wird (mit dem Parameter "AAPL,slot-name"), wird dieser in der sich später aufbauenden Kette "vererbt". Es geht also gar nicht darum, eine lange Kette zu bauen und ein Gerät direkt per _DSM-Methode zu benennen, sondern dass das Gerät, unabhängig davon wie es heißt und ob der Hersteller es auch benannt hat, dass dieses Gerät eben in der PCI-Sektion des Systemberichts/Systeminformationen auftaucht. Denn hier ist es für den Nutzer sehr

einfach einzusehen, ob überhaupt ein Treiber für dieses Gerät geladen wurde. Ansonsten tappt man da im Dunkeln. Kannst du gern ausprobieren. _DSM-Methode von z.B. "DSB1" entfernen, schon sieht man an diesem Strang in der PCI-Sektion keine Geräte mehr (die natürlich dennoch funktionieren).

Hast du meine überarbeitete SSDT mal ausprobiert? Ich möchte ja nur helfen, einerseits Dinge zu verschlanken, andererseits wichtige Dinge zu integrieren.