Erledigt Temperaturabhängige Lüftersteuerung für Lenovo T6X - T4XX (möglicherweise auch andere Notebooks)

Beitrag von „griven“ vom 24. Juni 2016, 23:27

Linux kann es und Windows kann es auch nur OS-X will nichts von der Reglung der Lüfterdrehzahl in Abhängigkeit zur Temperatur wissen aber warum eigentlich nicht?

OS-X bzw. Apple setzt hier hardwareseitig auf einen anderen Ansatz als die WINDOWS kompatiblen Computer und hier liegt die Crux. Während Windows und auch Linux die Temperaturen von den Sensoren des Mainboards bzw. der CPU oder auch GPU aus und regeln die Drehzahl der Lüfter dann über ACPI Calls. Apple hingegen nutzt für diesen Job den SystemManagementController (SMC) und adressiert hier komplett anders als es die ACPI Spezifikationen vorsehen. Naja so ganz im Regen stehen wir aber trotzdem nicht denn die FakeSMC und ihre Sensor Plugins geben uns ja bereits die Möglichkeit die Temperaturen zu lesen und was man auslesen kann kann man auch zum steuern verwenden...

1. Aber warum den Lüfter steuern, es geht doch auch so...
Vom Prinzip her stimmt das, der Rechner läuft auch so und der Lüfter dreht konstant mit mittlerer Drehzahl aber genau dieses Verhalten ist eher nicht optimal denn...

- Der Lüfter dreht obwohl es gar nicht nötig wäre -> Akkulaufzeit verringert und unnötige Geräuschkulisse
- Der Lüfter dreht mit konstanter Geschwindigkeit auch wenn die CPU richtig Last hat -> Gefahr der Überhitzung, Einbußen bei der Rechenleistung durch verminderten Takt

2. Wie kann man das nun ändern?
Im Grunde lässt sich das recht einfach ändern und es bedarf nur weniger Zutaten und ein wenig Zeit um den Lüfter in Abhängigkeit zur Temperatur drehen zu lassen. Grundlagenforschung in der Sache haben @Sebinouse und @Silencer im englischsprachigen Thinkpad Forum betrieben denen es auch auf dem Senkel gegangen ist das die ThinkPads unter OS-X den Föhn nicht abstellen wollten bzw. eben auch ziemlich heiß wurden weil der Lüfter nicht hoch regelt wenn Last auf der CPU und GPU liegt. Herausgekommen sind dabei 2 DSDT Edits von denen der eine ermöglicht die aktuelle Drehzahl des Lüfters auszulesen

Code

```
1. // Fan Speed reading in rpm
2. Field (ECOR, ByteAcc, NoLock, Preserve)
3. {
4. Offset (0x84),
5. HFN1, 16
6. }
```

und der andere ich darum kümmert die Geschwindigkeit mit der der Lüfter drehen soll abhängig von der Temperatur einzustellen

Code

```
1. Method (FAN0, 0, NotSerialized)
2. {
3. Store (B1B2 (^^EC.HFN1, ^^EC.HFN2), Local0)
4. Return (Local0)
5. }
6. Method (TCPU, 0, NotSerialized) // Fan Mode Accordind CPU Heatsink Temperature
7. {
8. Store (\_SB.PCI0.LPC.EC.TMP0, Local0)
9. If (LLessEqual (Local0, 0x32)) // CPU Temp is <= 50C
10. {
11. Store (Zero, \_SB.PCI0.LPC.EC.HFSP) // Set FAN Off
12. }
13. If (LGreaterEqual (Local0, 0x55)) // CPU Temp is >= 85C
14. {
15. Store (0x40, \_SB.PCI0.LPC.EC.HFSP) // Set FAN Mode Disengaged - Absolute Maximum
16. }
17. Else {
18. If (LGreaterEqual (Local0, 0x4B)) // CPU Temp is >= 75C
19. {
20. Store (0x07, \_SB.PCI0.LPC.EC.HFSP) // Set FAN Mode 7 - Maximum Speed
21. }
22. Else {
23. If (LGreaterEqual (Local0, 0x46)) // CPU Temp is >= 70C
24. {
25. Store (0x04, \_SB.PCI0.LPC.EC.HFSP) // Set FAN Mode 4 - Medium Speed
26. }
27. Else {
28. If (LGreaterEqual (Local0, 0x41)) // CPU Temp is >= 65C
29. {
```

30. Store (0x02, _SB.PCI0.LPC.EC.HFSP) // Set FAN Mode 2
31. }
32. Else {
33. If (LGreaterEqual (Local0, 0x3D)) // CPU Temp is >= 61C
34. {
35. Store (0x01, _SB.PCI0.LPC.EC.HFSP) // Set FAN Mode 1 - Lowest Speed
36. }
37. }
38. }
39. }
40. }
41. Return (Local0)
42. }

Alles anzeigen

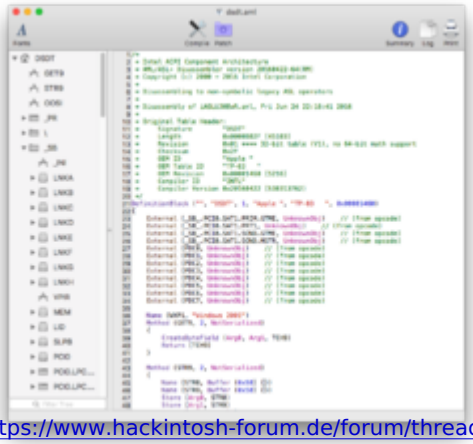
Da beide Patches schön dokumentiert sind fällt es leicht nachzuvollziehen was da eigentlich passiert. Zusammengefasst kann man sagen das der Lüfter bei Temperaturen unter 50°C abgeschaltet ist und ab 60°C wieder anspringt und dann abhängig von den definierten schwellen hoch bzw. auch wieder runter geregelt wird.

3. und wie baut man das jetzt ein und was braucht man dafür?

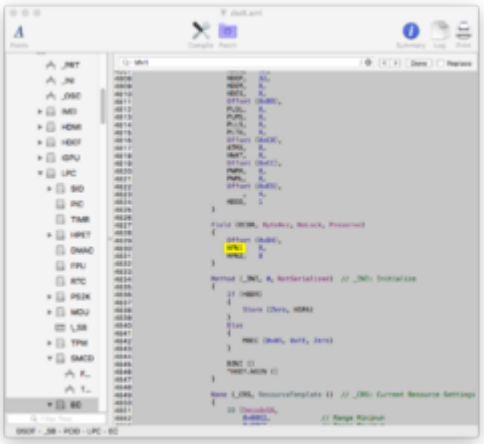
Um das Ganze jetzt in die Tat umzusetzen braucht es ein paar Tools bzw. Helferlein als da wären...

- MacIASL (gibt es im DL Bereich)
- Die DSDT des ThinkPads (Im Clover Menu F4 drücken DSDT liegt dann unter /ACPI/Origin bzw. einfach die nehmen die Ihr eh schon benutzt)
- Einen Plist Editor (X-Code oder PlistEdit pro alternativ tut es aber auch der Texteditor)
- ACPIPoll.kext

Nicht viel oder? Als erstes starten wir MacIASL und laden unsere DSDT was dann in etwa so aussehen sollte (DSDT vom T420s)



Jetzt prüfen wir ob die DSDT nicht sogar schon HFN1 enthält (bei bereits bearbeiteten DSDT´s oft der Fall) indem wir cmd+f eintippen und in der Suchleiste hfn1 eingeben. MacIASL sucht nun nach hfn1 und zeigt Euch das Ergebnis falls vorhanden auch gleich an. Bei mir sieht das so aus:

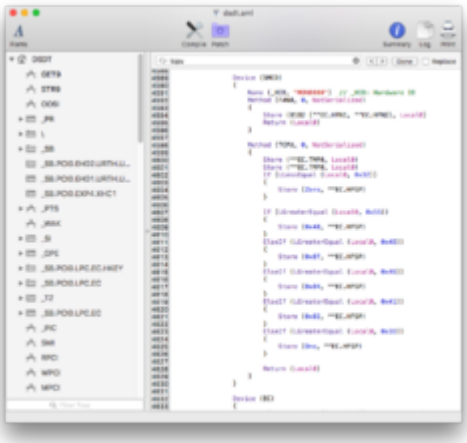


Solltet Ihr nicht fündig werden, dann ist das kein Beinbruch denn den entsprechenden Code kann man leicht selbst einfügen hierzu suchen wir nach "EC" und fügen einfach folgenden Code direkt nach der ziemlich langen Definition der OperationRegion (ECOR, EmbeddedControl, Zero, 0x0100) ein

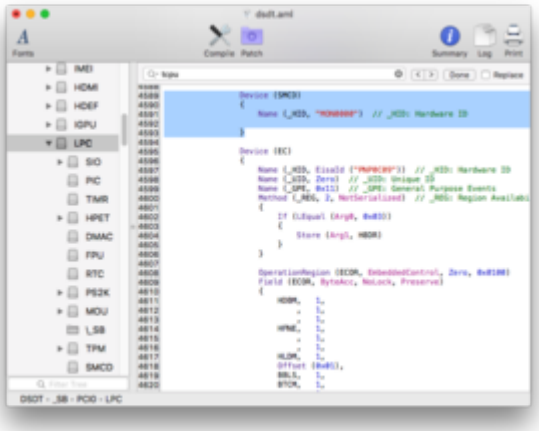
Code

1. Field (ECOR, ByteAcc, NoLock, Preserve)
2. {
3. Offset (0x84),
4. HFN1, 8,
5. HFN2, 8
6. }

In den meisten Fällen wird das aber nicht nötig sein weil HFN1 bereits definiert ist. Die Grundlagen zur Steuerung des Drehzahl ist nun geschaffen weiter geht es damit dem Patienten die Temperaturempfindlichkeit beizubringen. Hierzu bemühen wir erneut die Suchfunktion von MacIASL und suchen diesmal nach "tcpu" sofern es die Methode schon gibt finden wir sie unter dem Device SMCD sollte es sie noch nicht geben gehört sie dort hin.



Was auch immer im Device SMCD zu finden es wir ersetzen es und daher wird im ersten Schritt alles was sich im SMCD Device befindet gelöscht so, dass am Ende nur der Rumpf überbleibt (passt auf die Klammern auf)



ist das erledigt fügen wir unterhalb der Zeile "Name (_HID, "MON0000") // _HID: Hardware ID" folgenden Code ein:

Code

```
1. Method (FAN0, 0, NotSerialized)
2. {
3. Store (B1B2 (^^EC.HFN1, ^^EC.HFN2), Local0)
4. Return (Local0)
5. }
6.
7.
8. Method (TCPU, 0, NotSerialized)
9. {
10. Store (^^EC.TMP0, Local0)
11. Store (^^EC.TMP0, Local0)
12. If (LLessEqual (Local0, 0x32))
13. {
14. Store (Zero, ^^EC.HFSP)
15. }
16.
17.
18. If (LGreaterEqual (Local0, 0x55))
19. {
20. Store (0x40, ^^EC.HFSP)
21. }
22. ElseIf (LGreaterEqual (Local0, 0x4B))
23. {
24. Store (0x07, ^^EC.HFSP)
```

```
25. }
26. ElseIf (LGreaterEqual (Local0, 0x46))
27. {
28. Store (0x04, ^^EC.HFSP)
29. }
30. ElseIf (LGreaterEqual (Local0, 0x41))
31. {
32. Store (0x02, ^^EC.HFSP)
33. }
34. ElseIf (LGreaterEqual (Local0, 0x3D))
35. {
36. Store (One, ^^EC.HFSP)
37. }
38.
39.
40. Return (Local0)
41. }
```

Alles anzeigen

und kompilieren die erzeugte DSDT. Einmal kompiliert können wir sie nach /EFI/Clover/ACPI/Patched ablegen so, dass sie beim nächsten Start geladen wird. Jetzt müssen wir nur noch dafür sorgen, dass die Temperaturen regelmäßig abgefragt werden und hier kommt der ACPIPoller.kext zum tragen den wir aber auch noch schnell bearbeiten müssen den wir müssen ja definieren was einmal die Sekunde abgefragt werden soll. Hierzu einfach einen rechtsklick auf den Kext machen und "Paketinhalt Anzeigen" wählen. Im Ordner Contentsfinden wir die info.plist die wir uns jetzt mit einem plist Editor unserer Wahl öffnen. Unter demPunkt IOKitPersonalites -> ACPI Poller -> Methods tragen wir jetzt an Stelle0 tcpu ein undspeichern das Ganze dann.

Abschließend wird der ACPIPoller.kext noch mit dem KextUtility installiert und wenn wir alles richtig gemacht haben verhält sich der Föhn jetzt nach einem Neustart so, wie er soll 😆
https://youtu.be/Z6w6tNfRwG4
Wobei das Video auch zeigt, dass mein T420s wohl mal neue Wäremleitpaste braucht oO