

Erledigt

"Du hast ja Alles" - hmmm vielleicht, wenn ich einen Laptop habe.

Beitrag von „Brumbaer“ vom 28. Januar 2018, 21:17

Ziel dieses Threads ist es den Vorgang einen Laptop zu hackintoshisieren zu beschreiben.
Jeden Tag ein paar Zeilen.

Heute die Einleitung, wieso, weshalb, warum.

Morgen die ersten Schritte und ab übermorgen, Problemlösungen.

Das Ziel ist keine Installationsanleitung zu liefern, eher zu unterhalten und dabei ein paar Herangehensweisen zu beschreiben.

Die ersten beiden Folgen werden wenig technisch sein, das wird sich später etwas ändern.

Wieso. weshalb, warum ?

"Du hast aber auch alles" pflegt Carola zu sagen, meist anklagend, meist im Zusammenhang mit Computern.

Allerdings stimmt das nicht, na ja nicht so richtig.

Ich habe zum Beispiel keinen Hackintosh-Laptop.

Mein ständiger Begleiter ist ein iPadPro, es begleitet mich fast überall hin. Es kann so ziemlich alles, was ich unterwegs von einem Rechner erwarte, aber hin und wieder wäre es gut Mac Software unterwegs verwenden zu können, z.B. beim Hackintosh Stammtisch.

Aber Laptop und iPadPro mit sich rumzuschleppen ist etwas übertrieben, wie Hosenträger und Gürtel und wie das endet kann man in "Spiel mir das Lied vom Tod" sehen. Ich habe immer mal nach einem Surface Pro geschickt, aber die waren mir unter Berücksichtigung des Risikos, dass die Hackintoshisierung nur teilweise klappt, zu teuer.

Ich habe immer gesagt, "wenn noch einmal einen Laptop, dann wieder einen 17 Zöller, denn Bildschirmplatz ist durch nichts zu ersetzen".

[@DSM2](#) berichtete über einen Dell - 17 Zoll. Und dann noch Touch, das passt ja wie -, wie - , das passt ja gar nicht.

Das iPadPro ist hart an der Größen-Grenze, um es immer mit mir herumzuschleppen. Als iPadPro Ersatz, kommt ein 17 Zöller somit nicht in Frage.

Also was gebe ich auf mein dummes Geschwätz von vorhin und suche nach einer 15" Variante. Es gibt sie tatsächlich, aber egal welchen Test man liest, immer sticht der zu dunkle Bildschirm

negativ heraus. Und ein dunkler Schirm bei einem "ständigen Begleiter" ist wie Windows - nicht vertretbar.

Also gut bzw. schlecht, und nach einem anderen Modell gesucht.

Und irgendwann zwischen der Entscheidung nach einem Two in One zu suchen und dem Verwerfen des Dell hat sich der Gedanke eingeschlichen, dass 13 Zoll eigentlich genug sein sollten, denn das iPadPro hat ja in etwa diese Größe .

Von 17" auf 13" in nicht ganz 2 Stunden; mal beim Guinnessbuch nachschauen ob das Rekord verdächtig ist.

Es heißt immer Lenovo sei die erste Wahl für Laptops, super kompatibel und ganz einfach. Google weißt mich auf das Lenovo Miix 520 hin. Ein 13 Zoll - Two In One mit nicht dunklem Display und "tadaaah" 8250u oder 8550u. Das klingt gut, aber der Preis von 1100€ ist an der Schmerzgrenze, wenn man nicht weiß in wie weit sich das Gerät hackintoshisieren lässt.

Für das Geld bekommt man:

- 8250u,
- UHD620 IGPU,
- 8GB Ram,
- 256GB NVMe,
- 12,2" IPS, 1920x1200,
- BT und WiFi,
- Touchscreen,
- Stift,
- Fingerabdruck-Lesegerät,
- 2 Kameras,
- LTE Modem.

Bis zu BT mach ich mir keine Sorgen. Die BT und WiFi Karte kann man vermutlich tauschen, aber bei den anderen Dingen muss man sich auf Scheitern gefasst machen. Aber Touch hätte ich schon sehr gerne.

Berlin hat nicht alles, auf jeden Fall keinen Miix 520. Also gut, Internet bemühen. Lenovo bietet das Gerät auf seiner Webseite in Platin an. Aber ich habe es im Web in Grau gesehen und finde ein Grau passt besser zu mir.

Also gegoogelt und siehe da, es sieht so aus als wäre die Farbe Vertriebskanal abhängig. Platin im Lenovo Webshop und Grau bei den Händlern. Und siehe da, es gibt Händler, die haben so viele von den Grauen, dass sie sie verkaufen und der eine oder andere hat sogar einen auf Lager.

Gesehen, bestellt und regelmäßig die emails gecheckt , ob es denn schon eine Versandbenachrichtigung gibt. Denn ich habe bestenfalls fast alles, Geduld z.B. habe ich keine. Am nächsten Tag die Erlösung, email da, aber keine Trackingnummer; da aus dem Außenlager versandt. Warum das Außenlager die Nummer nicht mitteilen kann, verstehe ich nicht. Was soll's, ich versteh so vieles nicht.

Nächster Tag kein Paket und schon weiß man wo der Nick Brumbaer herkommt. Die Paketdienste liefern bei uns meist vor 13 Uhr. Am darauffolgenden Tag habe ich einen Termin um zwölf. Bis elf ist kein Paket da, muß gleich weg, noch kurz zum Briefkasten, die Briefpost reinholen. Mißmutig brummelnd nach draußen schlurfen, auf die Paketdienste, die nie kommen, wenn man auf sie wartet, und auf die Händler, die keine Trackingnummern verschicken können, schimpfen. Und während ich die Post aus dem Briefkasten nehme und mich über die Ungerechtigkeit der Welt auslassen will, fährt ein Paketbote vor. Na geht doch, wenn man so ruhig, geduldig und gelassen wartet wie ich, wird das belohnt. Es ist ein Paket für Carola.

Der Paketbote kommt ungefähr 10 Minuten nachdem ich das Haus verlassen habe und hat das Paket gleich wieder mitgenommen. Der Paketdienst hinterlegt nicht zustellbare Pakete an einer Tankstelle in der Nähe. Ab 17:00 kann es abgeholt werden, nur noch zwei Stunden.



Da isser ja.
Ein Lenovo, kann ja nicht so schwer werden.
Schau mer mal.

Beitrag von „Doctor Plagiat“ vom 28. Januar 2018, 21:33

Bin sehr neugierig, habe es gleich mal abonniert.

Beitrag von „Dr.Stein“ vom 28. Januar 2018, 21:34

Ich find deine kleinen Tagebucheinträge echt schön.
... freue mich sehr auf eine Fortsetzung. 😄

Beitrag von „derHackfan“ vom 28. Januar 2018, 21:39

Es kann nur einen geben! 👍

Beitrag von „Brumbaer“ vom 30. Januar 2018, 18:17

Den Miix aufgestellt, Tastatur und Netzteil angeschlossen und Windows gestartet. Die Installation ging flott und die Windows Update Orgie fiel dank des relativ neuen Systems weitestgehend aus. Kurz mal in den Gerätemanager geschaut, welche USB Ports belegt sind und welche PCI Geräte verwendet werden.

Die Geräte des Intel Chipsatzes werden von 13.2 nur teilweise unterstützt, BT auch, aber WiFi nicht. Noch halbherzig nach anderen Geräten geschaut und nach einem [BIOS Update](#) gesucht, denn das geht bestimmt nur über Windows. Es gibt tatsächlich eins und - Glückes Geschick - es lässt sich installieren.

Alles gut soweit.

Das hat alles Methode

Die in Foren beschriebenen Wege in den macos Himmel basieren auf einem Installerstick. Ich verwende einen geringfügig anderen Weg.

Ich verwende einen USB Stick mit EFI Partition und einem fertig installierten macos.

Ziel des Hackintoshierens ist ein Rechner, der wie eine Ente watschelt, wie eine Ente quakt, aber doch kein Mac ist.

An einem Mac muss man am macos nichts schrauben, damit es läuft. So soll es auch bei einem Hackintosh sein.

Deshalb sollten alle Anpassungen in der EFI erfolgen. Wenn alle Anpassungen in der EFI erfolgen, kann man das macos auf jedem System mit einer passenden EFI verwenden.

Wenn unsere EFI mit dem macos auf dem Stick läuft haben wir unser Ziel erreicht.

Das macos wurde an meinem Haupthackintosh auf dem Stick installiert und wird bei der Installation jedes neuen Hacks wiederverwendet.

Neben dem System befindet sich noch ein Ordner mit fürs Hackintoshieren nützlichen Programmen und ein paar Benchmarks auf dem Stick.

Das hat den Vorteil, dass ich nach der Installation eines Systems, alles zur Hand habe, was ich fürs Finetuning brauche.

Und nebenbei fällt noch ein Notfall System ab, solange die EFI eines Hacks noch ein Booten zulässt.

An Apple a day, keeps the doctor away

Eva bringt Männer dazu Obst zu essen und EFI bringt Systeme zum booten. Da diese EFI vornehmlich als Startpunkt für neue Systeme dienen soll, kommt sie ziemlich nackt daher, wie Eva mit dem Apfel.

Alle Patches und Häkchen, die man nicht sicher braucht werden gelöscht. Nur FakeSMC, Unsolid, IntelMausiEthernet - Mäuse wird man halt nicht los - und ein noch anzupassendes USB Kext kommen in den Kext Ordner. Von den EFI Treibern wird im Clover Installer als einzige Option eine der AptioFix Variationen gewählt. Im Moment versuche ich es immer zuerst mit AptioMemoryFix.

Ganz nach oben

Aber damit bekommt man nicht jedes System zum Laufen, also kommen die gebräuchlichsten Kexte und EFI Treiber in das oberste Verzeichnis der EFI Partition. Warum dorthin ? Ich brauche sie nur in Verbindung mit der EFI Partition , deshalb auf diese Partition. Und wenn die Treiber und Kexte auf dem selben Laufwerk, wie die Ziel-Ordner sind, kann ich sie hin und her kopieren ohne jedesmal etwas löschen zu müssen. Spart ein paar Clicks.

Sobald der Stick bootet und man den Finder zu sehen bekommt, wird eine Kopie des EFI Ordners angelegt und ein Auszug der IORegistry gespeichert, die interne Festplatte formatiert und schließlich macos auf der internen installiert. Und dann geht es ans Finetuning.

Egal ob man einen Installer oder ein macos zum Laufen bekommen will, es hilft wenn man einen schnellen USB-Stick einsetzt. Ich verwende einen 64 GB SanDisk Extreme, der ist für einen Stick recht schnell.

Da stehen Eva und EFI und locken mit ihren Äpfeln. Auweia, das sollte ich anders formulieren.

Obstsalat

Wie bekommt man nun den Apple-Miix zustande ? Indem man EFI ihrer Nacktheit beraubt.

Die CPU ist trotz der 8 vorne eine aus der Familie derer von Kaby Lake. Es sollten keine weiteren Anpassungen notwendig sein.

Die PCIId des USB Chipsatzes ist 0x9d2f8086. Die kennt 10.13.2 nicht, also im USB Kext, den Treiber für das Vorgängermodell geladen. Der Blick in den Gerätemanager unter Windows hatte gezeigt, dass die USB Buchse HS01 und SS01 verwendet und es noch 14 weitere USB Kanäle gibt (insgesamt 10 HS und 6 SS).

Die Einträge im USB Kext angepasst. Ein Eintrag mehr als das 15 Port-Limit erlaubt, also verzichte ich auf SS05. An HS05 befinden sich Touchpad und Tastatur, da SS und HS mit dem selben Index gerne an den selben Anschluss gehen und HS05 schon benutzt wird, wird SS05 vermutlich nicht benutzt. Wie man das macht ist [hier](#) beschrieben.

Die IGPU des 8250u hat die PCIId 0x59178086 und die wird von macos nicht unterstützt. "Kein Problem", machen wir einen Kext Eintrag, der den Treiber lädt, haben wir gerade mit dem USB Controller gemacht. Die Methode funktioniert zwar beim 8700K, aber nicht bei der 8250u. Wie jetzt ?

Buntes Treiben, hinter verschlossenen Türen

"Glaube mir, ich stehe auf der Gästeliste, lass mich rein".

Treiber für PCI Geräte werden meist an Hand ihrer PCIId ausgewählt. Nicht passende PCIId, kein Treiber. Weiß man, dass ein Treiber mit dem Gerät funktioniert obwohl der Treiber die PCIId nicht kennt, kann man den Treiber über einen Kexteintrag wie bei der USB Geschichte gezeigt, laden. Genau genommen fügt man die PCIId der Liste, der unterstützen Geräte, bei.

"Ein Blick auf die Gästeliste und ich nenne dir einen passenden Namen".

Was aber, wenn der Treiber die PCIId abfragt und anhand ihrer verschiedene Operationen vornimmt ? Dann klappt das ganze nicht mehr.

An der Stelle springt Clover ein. Clover bietet unter Devices die Option eine Fake Id zu setzen. Das Gerät bekommt nun eine neue PCIId und der Treiber, der nach der Id fragt, denkt es ist ein passendes Gerät.

"Ich kann beweisen, dass ich der bin, ich habe einen Ausweis (gefälscht, muss ja keiner wissen)".

Clover verändert einen Eintrag in der Registry. Der Wert in der Registry stammt ursprünglich aus einem Register des PCI Gerätes. Manche Treiber fragen nun nicht die Registry nach der PCIId, sondern verwenden eine Funktion des PCI Gerätetreibers, die die Register liest, um die PCIId direkt vom Gerät zu holen. Da funktioniert die FakeId nicht. Da hilft nur das PCIFakeId

Kext. Es patched die Funktion des Gerätetreibers, die Register liest und gibt bei Abfrage der Id Register die gewünschten Werte zurück.

"Ein DNS, wäre mir das Reinkommen nicht Wert"

Nun könnte ein Kext die Register ohne Hilfe des PCI Treibers lesen, aber das ist so aufwändig, dass es keiner macht, so wie auch niemand einen DNS Test am Empfang verlangt.

Um die IGPU zum Laufen zu bringen wird man vermutlich einen falschen Namen und den dazugehörigen Ausweis brauchen. Also Fake Id in Clover und FakePCIId mit passendem Injector Kext.

Die IGPU unterstützt verschiedene Ausgänge. Deren Konfiguration beschreibt der ig-platform.id Eintrag in der config.plist.

Unsolid, damit die SSD nicht APFS formatiert wird, haben wir schon.

Die Maus hat ihre Schuldigkeit getan, die Maus kann gehen, denn wir haben keinen Ethernet Anschluss. Außerdem hat Eva Angst vor Mäusen. Also raus mit IntelMausi.

Das soll erst mal genügen. Alles andere kommt mit dem Finetuning.

Wenn alles richtig konfiguriert ist, sollten wir von Eva einen ganzen und von EFI einen angebissenen Apfel bekommen können.

Endlich geht's los

TackTacktakTackTack... kennt ihr das ? Nicht die Trommeln aus Jumanjii, die machen Bumm, Bumm, Bumm. Ein Specht - ist es auch nicht.

Das ist das Geräusch, das die F2 Taste macht, wenn man einen Lenovo Laptop auffordert das BIOS zu öffnen.

Aber ich frage mich, ob ich etwas falsch gemacht habe. Denn das was ich zu sehen bekomme ist kein BIOS, das ist vielleicht ein B, bei großzügiger Auslegung ein BI, aber niemals ein BIOS.

Wenn es nicht viel einzustellen gibt, kann man auch nicht viel einstellen. Secure Boot aus, Fast Boot vorsichtshalber aus.

Und los geht's.

"Huston, wir haben einen Boot". Aaaaaaaaapfel, Waaaaaaaarte balken, Crash.

Ok, ziemlich weit hinten vermutlich Grafikkartentreiber.

Also -v gesetzt. Aber es bringt keine Erleichterung, der Crash kommt irgendwie kurz nach Initialisierung der IGPU.

In Ruhe nochmals die Debug Meldungen beim Starten gelesen und siehe da LPSS wird nicht geladen oder gefunden, was auch immer.

Na und, deshalb crashed man nicht. Der Chipsatz ist auch nicht neuer als der in meinem großen Rechner und der zeigt den Fehler nicht. Ein kurzer Check zeigt, dass auf dem Stick noch ein frühes 10.3.2 ist - früh wie in Beta. Den großen mit dem System auf dem Stick gebootet und ein System Update laufen lassen. Zurück an den MiiX und Neustart. LPSS Fehler weg, aber immer noch ein Crash. Vielleicht ein USB 3.0 Problem. Also einen USB 2.0 Hub zwischen Rechner und Hub und Crash.

Macht nichts, einfach eine falsche PCIId setzen und der Rechner lädt den Intel-Treiber nicht und der Rechner bootet. Keine Beschleunigung, aber immerhin Finder. Soweit die Theorie. Denkste System crashed, wie immer schwarzer Bildschirm, Neustart.

Crash bei falscher ID schließt eigentlich ein Problem mit dem Intel Treiber aus, denn der wird bei falscher PCIId nicht geladen.

Neueres Clover, Crash, anderes AptioFix, Crash, IGPU wieder als IGPU konfiguriert, crash.

Unter Windows, war der für die Grafik reservierte Speicherplatz 128MB, aber vielleicht setzt ja erst Windows den Wert hoch, also IntelGraphicsDVMTFixup.kext rein. Neustart, Crash.

IntelGraphicsFixup.kext rein, Neustart, Crash.

Neustart, geht.

Im Finder EFI Ordner kopiert, Registry Auszug erstellt. Schnell AppleALC in den Kext Ordner, erste unterstützte Id in der config.plist gesetzt.

Neustart, Crash.

AppleALC wieder raus, Neustart, Crash.

Gesicherten EFI Ordner zurückgespielt, Neustart, Crash.

Wie bitte ? Aber, aber ... es ging doch, Crash.

Alle möglichen Häkchen gesetzt, gelöscht, Crash. Patches, rein, raus und im Ringelrein, Crash. Prozessor gegoogelt, Fehler gegoogelt, nichts.

Rechner ausgeschaltet, erst still in der Ecke gesessen, dann Kopf gegen die Wand gehauen, was gäbe ich jetzt für ein paar tröstende Worte von Eva oder EFI, aber ihre Äpfel können sie behalten ich bin allergisch gegen Äpfel.

Rechner einschalten, bootet.

Stutz, Neustart, Crash. Was war anders ? Ich habe an Eva und EFI gedacht, also an Eva und EFI denken, Neustart, Crash. Ich habe Äpfel verflucht, also Äpfel verfluchen, Neustart, Crash. Ich habe den Rechner ausgeschaltet, also den Rechner ausschalten, Neustart und bootet.

Ich kann es mir nicht erklären. Wie die Jungfrau zum Kinde, kommt der Rechner über das Ausschalten zum Finder.

EFI Ordner von dem ganzen Testkram bereinigen, ausschalten, Neustart, bootet.

Ok , System auf der internen Platte installieren. Eigentlich wollte ich die Windows NVMe nicht formatieren, falls ich sie noch mal brauche, also eine andere einbauen. Ich habe ja alles, also

auch noch eine SM960 512 GB.

Ich habe ja alles, auch eine gehörige Portion Pessimismus, also schließe ich die Augen und überlege ob ich zu Lösungsmittel, um die Klebestellen zu öffnen, oder gleich zur Flex greifen soll. Auf der Suche nach der geeigneten Stelle für die Flex fallen mir 6 kleine Schrauben unter dem Aufsteller auf.

Wirklich klein und für einen Imbus Schraubendreher gedacht, den vermutlich Ameisen für ihre Uhren verwenden.

Aber ich habe ja alles, auch ein Werkzeug Set von iFixit and da ist auch ein Bit für diese Schrauben dabei. 48 Drehungen später öffnet sich ein Spalt im Gehäuse. Zum iFixit Set gehört auch eine stumpfe Klinge um beim Gehäuse-Öffnen, Schnapper zurückzudrücken und das Gehäuse lässt sich damit ganz leicht öffnen.

Die Kabel zum Bildschirm sind gerade lang genug, so dass man an das Innenleben kommt ohne erst Kabel zu lösen. Und zum ersten Mal, zum allerersten Mal sind Fluch und Lenovo-Entwickler nicht gleichbedeutend.



Die NVMe liegt direkt unter der Prozessor Heatpipe. Ich bin mir nicht sicher, ob das eine gute Idee ist. Aber egal, löst man eine Kreuzschlitzschraube lässt sich die Heatpipe soweit anheben, dass man die NVMe tauschen kann. Gesagt getan.

Und etwas weiter oben kann man die WiFi Karte sehen. Da das Gehäuse schon offen ist, könnte ich die ja bei der Gelegenheit tauschen. Ich hab ja alles, also habe ich auch noch eine DW1830 rum liegen. Super, passt nur nicht, sie ist links 2 mm zu breit, Pech. Aber ich hab ja alles, also auch noch eine DW1560 und die passt. Antennen ab, Karte raus, Antennen dran, DW1560 rein. Gehäuse zu und Rechner eingeschaltet und nichts Gar nichts. Nicht einmal das hässliche Lenovo Logo erscheint, selbst die Hintergrundbeleuchtung bleibt dunkel. "Rechner, ich weiß wo deine Schrauben sind". Also Schrauben raus, Schnapper entschnappen und gleich das Kabel wieder einstecken. Aber da ist kein Kabel, das man stecken müsste. Zeit für ein paar ausgesuchte Schimpfworte. Als erstes die DW1560 herausnehmen , denn da kommt man besser ran als an die NVMe. Bildschirm zur Seite halten, damit es keinen Kurzschluss gibt, einschalten und ich werde sofort mit dem Anblick eines Lenovo Logos gequält. Ich habe nie vermutet Masochist zu sein, aber diese Qual erfüllt mich mit Freude. DW1560 wieder rein und nichts geht mehr.

Ich habe noch mehrere Apple WiFi Karten mit M.2 Adapter, aber die sind dreimal so groß und passen nicht. Also alte Karte rein , Rechner zu und eine neue 1560 bestellt, vorsichtshalber eine von/für Lenovo.

Vom Stick gebootet, Installer gestartet, erster Neustart ... zweiter Neustart und ... Crash. Ausschalten, einschalten und ein bisschen weiter , aber fertig wird er nicht. Verzweiflung macht sich breit. Ich ersetze den Stick durch eine SSD in einem USB Adapter, nur aus praktischen Gründen. Es ist einfacher das Kabel am Adapter umzustecken, als für jede Änderung den Stick vom Mixx an den großen und zurück zu bringen. Und es schont die Anschlüsse an Rechnern und Stick

Ich habe nicht erwartet, dass das was ändert und so ist es auch. Und die Änderungs-, Test-Orgie wiederholt sich. Auch schon verworfene Konfigurationen werden wieder ausprobiert, mal von USB, mal von der internen Platte gebootet, nur ist alles noch langsamer, weil ich statt neu zu starten jedes Mal ein- und ausschalte.

Und aus heiterem Himmel - ich nehme an er ist heiter, es ist gegen 3:00 Uhr morgens und ziemlich dunkel, Eva und EFI schlafen bestimmt schon - bootet der Rechner, von USB und als er, als hätte ich es noch nicht gemerkt, mir mitteilt, dass das System auf Grund eines Fehlers neugestartet wurde, bin ich bereit die Legalisierung des Waffenbesitzes zu unterstützen und eine Youtube Anleitung wie man aus einem Two in One ein Three in One - Laptop, Tablet und 45er Kugel - macht vorzubereiten.

Aber nach wenigen Minuten verschwindet der rote Schleier vor meinen Augen und ich kann den reflexartigen Drang unterdrücken die Meldung wegzuklicken. Stattdessen schaue ich mir den Bericht an.

Code

1. Anonymous UUID: D5D7FFE6-EA0D-D727-1E16-7BA6D7E183AD
- 2.
- 3.
4. Tue Jan 23 17:15:16 2018
- 5.
- 6.
7. *** Panic Report ***
8. panic(cpu 2 caller 0xfffff8002f6f2e9): Kernel trap at 0xfffff7f85621d08, type 14=page fault, registers:
9. CR0: 0x0000000080010033, CR2: 0x0000000000000033, CR3: 0x0000000258824050, CR4: 0x00000000003627e0
10. RAX: 0x0000000000000000, RBX: 0xfffff80268b70f0, RCX: 0x0000000000000202, RDX: 0xfffff80f52a92b5
11. RSP: 0xfffff91201536f0, RBP: 0xfffff9120153720, RSI: 0xfffff91201537c0, RDI: 0xfffff80268b70f0

12. R8: 0xffffffff9120153560, R9: 0x0000000000000008, R10: 0x0000000000000010, R11:
0x0000000000000000
13. R12: 0xffffffff80268b70f0, R13: 0xffffffff7f85691398, R14: 0x0000000000000000, R15:
0x00000000000004006
14. RFL: 0x0000000000010202, RIP: 0xffffffff7f85621d08, CS: 0x0000000000000008, SS:
0x0000000000000010
15. Fault CR2: 0x0000000000000033, Error code: 0x0000000000000000, Fault CPU: 0x2, PL:
0, VF: 0
16.
17.
18. Backtrace (CPU 2), Frame : Return Address
19. 0xffffffff91201531a0 : 0xffffffff8002e505f6
20. 0xffffffff91201531f0 : 0xffffffff8002f7d604
21. 0xffffffff9120153230 : 0xffffffff8002f6f0f9
22. 0xffffffff91201532b0 : 0xffffffff8002e02120
23. 0xffffffff91201532d0 : 0xffffffff8002e5002c
24. 0xffffffff9120153400 : 0xffffffff8002e4fdac
25. 0xffffffff9120153460 : 0xffffffff8002f6f2e9
26. 0xffffffff91201535e0 : 0xffffffff8002e02120
27. 0xffffffff9120153600 : 0xffffffff7f85621d08
28. 0xffffffff9120153720 : 0xffffffff7f85639388
29. 0xffffffff9120153760 : 0xffffffff7f856390f0
30. 0xffffffff91201537b0 : 0xffffffff7f85639037
31. 0xffffffff9120153810 : 0xffffffff7f856367e5
32. 0xffffffff9120153850 : 0xffffffff7f8563ba0c
33. 0xffffffff9120153880 : 0xffffffff7f85640088
34. 0xffffffff91201538d0 : 0xffffffff7f85661134
35. 0xffffffff9120153950 : 0xffffffff7f856624d9
36. 0xffffffff9120153990 : 0xffffffff7f856631c2
37. 0xffffffff91201539e0 : 0xffffffff7f8565a68c
38. 0xffffffff9120153a20 : 0xffffffff7f8565eded
39. 0xffffffff9120153a80 : 0xffffffff7f85617283
40. 0xffffffff9120153b10 : 0xffffffff7f83b97e99
41. 0xffffffff9120153b60 : 0xffffffff7f83b9734c
42. 0xffffffff9120153b80 : 0xffffffff7f83b974ca
43. 0xffffffff9120153bb0 : 0xffffffff7f84516c80
44. 0xffffffff9120153bf0 : 0xffffffff7f84517352
45. 0xffffffff9120153c70 : 0xffffffff7f8450f158
46. 0xffffffff9120153cc0 : 0xffffffff8003464cff
47. 0xffffffff9120153d10 : 0xffffffff80034c0ee7
48. 0xffffffff9120153d70 : 0xffffffff8002f292f2

49. 0xffffffff9120153dc0 : 0xffffffff8002e55c30
50. 0xffffffff9120153e10 : 0xffffffff8002e32cbd
51. 0xffffffff9120153e60 : 0xffffffff8002e45b7b
52. 0xffffffff9120153ef0 : 0xffffffff8002f5952d
53. 0xffffffff9120153fa0 : 0xffffffff8002e02926
54. Kernel Extensions in backtrace:
55. com.apple.iokit.IOACPIFamily(1.4)[8794C760-FDD9-3664-ADED-4A9BBEC6E517]@0xffffffff7f83b96000->0xffffffff7f83b9efff
56. com.apple.driver.AppleACPIPlatform(6.1)[1804645B-B360-305E-B1BE-916F5E3E1CC4]@0xffffffff7f85611000->0xffffffff7f856acfff
57. dependency: com.apple.iokit.IOACPIFamily(1.4)[8794C760-FDD9-3664-ADED-4A9BBEC6E517]@0xffffffff7f83b96000
58. dependency: com.apple.driver.AppleSMCRTC(1.0)[3F01C7A4-754F-39DD-A872-B4FAF0442276]@0xffffffff7f84be4000
59. dependency: com.apple.iokit.IOPCIFamily(2.9)[C08F7FC1-78A4-3A1B-BFE2-C07080CF2048]@0xffffffff7f83734000
60. dependency: com.apple.driver.AppleSMC(3.1.9)[259F0A4B-0AAB-30F3-8896-629A102CBD70]@0xffffffff7f83b9f000
61. com.apple.iokit.IOGraphicsFamily(517.22)[2AEA02BF-2A38-3674-A187-E5F610FD65B7]@0xffffffff7f84500000->0xffffffff7f84546fff
62. dependency: com.apple.iokit.IOPCIFamily(2.9)[C08F7FC1-78A4-3A1B-BFE2-C07080CF2048]@0xffffffff7f83734000
- 63.
- 64.
65. BSD process name corresponding to current thread: WindowServer

Alles anzeigen

Und da ist er, der Silberstreif am Horizont - oder doch die Morgendämmerung.
Mal sehen ob ich EFI einen guten Morgen bereiten kann.

Beitrag von „Brumbaer“ vom 1. Februar 2018, 14:21

Aller Anfang ist schwer

Der heutige Exkurs führt uns in die unheimlichen Gefilde der Maschinensprache.

Was hier beschrieben wird ist es eher komplex und bedarf eines soliden Grundwissens. Ich weiß nicht so recht ob ich Evas Begleiter rufen und bei ihnen beginnen soll, oder nur das Ergebnis präsentieren, weil es eh keinen interessiert und zu wenig Infos ein Nachverfolgen des

Vorgehens sowieso unmöglich macht.

Ich lasse Adam in Ruhe und gehe einen Mittelweg. Wem's zu viel wird einfach aufhören, nächstes Mal wird es wieder weniger "speziell". Aber vielleicht regt es den einen oder anderen an mal etwas tiefer in die Materie ein zu steigen.

Nach dieser obskuren Einleitung sei gesagt, dass ich nicht weiß ob das Problem mit einem der tausend gebetsmühlenartig angewendeten Standard Rehab DSDT Patches nicht aufgetreten wäre. Das spielt auch keine Rolle, da ich sie bewußt nicht angewandt habe, konnten sie auch nichts verhindern.

Panic auf der Titanic

Dank AptioMemoryFix, funktioniert das NVRam und macos liefert den, in meiner letzten Post gezeigten, Fehlerbericht.

Kerneltrap sagt einem, wo das Problem aufgetreten ist. Das ist oft nicht sehr aussagekräftig. Und ohne Anhaltspunkte sagt einem der Wert auch nicht viel. Interessanter ist da der Backtrace.

Aus Computersicht werden immer nur Unterprogramme ausgeführt. Ob App, Programm, Funktion, Methode, Subroutine oder was auch immer für die CPU ist es ein Unterprogramm, das ausgeführt wird und dann dorthin zurückkehrt von wo es aufgerufen wurde.

Babuschka

Jedes Unterprogramm hat eine Aufgabe zu erfüllen. Ist die Aufgabe komplex oder hat man andere Unterprogramme, die Teile der Aufgabe lösen können, so bekommt ein solches eine Teilaufgabe zugewiesen und wird aufgerufen. Und es selbst ruft vermutlich wieder Unterprogramme auf. Unterprogramme, können sich sogar selbst aufrufen. Wie in einer virtuellen Babuschka hat man Unterprogramme in Unterprogrammen in Unterprogrammen. Dabei entsteht eine Hierarchie. Ein aufgerufenes Unterprogramm ist tiefer als das aufrufende Unterprogramm. Grob kann man sagen je tiefer, desto Detailarbeit. Es gilt zu beachten, dass ein Unterprogramm im Sinne von Quellcode an sich keine Position in der Hierarchie hat, sondern erst der Aufruf die Position festlegt. Unterprogramme können so je nach Programm weiter oben oder weiter unten oder sogar mehrmals in der Hierarchie vorkommen.

Errötend folgt er ihren Spuren

Jede Zeile im Backtrace zeigt uns in der ersten Spalte, die Adresse des Stackframes und in der zweiten Spalte die Rücksprungadresse eines Unterprogramms.

Die Rücksprungadresse in der ersten Zeile zeigt wohin das gerade ausgeführte Unterprogramm zurückspringen wird, wenn es fertig ist. Die nächste Zeile zeigt uns wohin das Unterprogramm, das dieses Unterprogramm aufgerufen hat, springen wird, wenn es fertig ist usw.. Je tiefer ein Unterprogramm in der Hierarchie steht, desto weiter vorne steht es im Backtrace.

Wir können also sehen, dass der Computer zum Zeitpunkt des Absturzes ein Unterprogramm ausgeführt hat, das danach zur Adressen 0xfffff8002e505f6 zurückkehren würde.

Boah, Boah, Boa Constrictor, 0xfffff8002e505f6 meine Güte, da bin ich aber ... sprachlos.

Das beste kommt zum Schluß

Zugegebenermaßen klingt 0xfffff8002e505f6 nicht hilfreich.

Aber am Ende des Tunnels ist Licht - keine Angst, wir gehen nicht drauf zu, wir schauen es uns nur an.

"Kernel Extensions in backtrace:" sagt uns in welchen Kexten die Rücksprungadressen liegen und somit welche Kexte die Unterprogramme aufgerufen haben, die am Ende zum Crash führten.

Die dependency Zeilen sagen uns in welchen Kexten das Kext, das in deren Abhängigkeit steht, Unterprogramme aufruft, aber nicht in welchen Kexten, die Kexte wieder Unterprogramme aufrufen. Aber es ist an dieser Stelle irrelevant.

Wenn wir die dependency Zeilen streichen, bleiben 3 Zeilen übrig. Am Anfang steht die Kennung des Kextes und am Ende welchen Speicherbereich das Kext belegt. Rücksprungadressen, die in diesem Speicherbereich liegen, zeigen also auf die Stelle an der das Kext ein Unterprogramm aufgerufen hat.

Hier habe ich die Rücksprungadressen, die sich einem der Kexte zuordnen lassen farblich markiert.

Backtrace (CPU 2), Frame : Return Address

0xfffff91201531a0 : 0xfffff8002e505f6

0xfffff91201531f0 : 0xfffff8002f7d604

0xfffff9120153230 : 0xfffff8002f6f0f9

0xfffff91201532b0 : 0xfffff8002e02120

0xfffff91201532d0 : 0xfffff8002e5002c

0xfffff9120153400 : 0xfffff8002e4fdac

0xfffff9120153460 : 0xfffff8002f6f2e9

0xfffff91201535e0 : 0xfffff8002e02120

0xffffffff9120153600 : 0xffffffff7f85621d08
0xffffffff9120153720 : 0xffffffff7f85639388
0xffffffff9120153760 : 0xffffffff7f856390f0
0xffffffff91201537b0 : 0xffffffff7f85639037
0xffffffff9120153810 : 0xffffffff7f856367e5
0xffffffff9120153850 : 0xffffffff7f8563ba0c
0xffffffff9120153880 : 0xffffffff7f85640088
0xffffffff91201538d0 : 0xffffffff7f85661134
0xffffffff9120153950 : 0xffffffff7f856624d9
0xffffffff9120153990 : 0xffffffff7f856631c2
0xffffffff91201539e0 : 0xffffffff7f8565a68c
0xffffffff9120153a20 : 0xffffffff7f8565eded
0xffffffff9120153a80 : 0xffffffff7f85617283
0xffffffff9120153b10 : 0xffffffff7f83b97e99
0xffffffff9120153b60 : 0xffffffff7f83b9734c
0xffffffff9120153b80 : 0xffffffff7f83b974ca
0xffffffff9120153bb0 : 0xffffffff7f84516c80
0xffffffff9120153bf0 : 0xffffffff7f84517352
0xffffffff9120153c70 : 0xffffffff7f8450f158
0xffffffff9120153cc0 : 0xffffffff8003464cff
0xffffffff9120153d10 : 0xffffffff80034c0ee7
0xffffffff9120153d70 : 0xffffffff8002f292f2
0xffffffff9120153dc0 : 0xffffffff8002e55c30
0xffffffff9120153e10 : 0xffffffff8002e32cbd
0xffffffff9120153e60 : 0xffffffff8002e45b7b
0xffffffff9120153ef0 : 0xffffffff8002f5952d
0xffffffff9120153fa0 : 0xffffffff8002e02926

Kernel Extensions in backtrace:

com.apple.iokit.IOACPIFamily(1.4)[8794C760-FDD9-3664-ADED-4A9BBEC6E517]@0xffffffff7f83b96000->0xffffffff7f83b9efff

com.apple.driver.AppleACPIPlatform(6.1)[1804645B-B360-305E-B1BE-916F5E3E1CC4]@0xffffffff7f85611000->0xffffffff7f856acfff

dependency: com.apple.iokit.IOACPIFamily(1.4)[8794C760-FDD9-3664-ADED-4A9BBEC6E517]@0xffffffff7f83b96000

dependency: com.apple.driver.AppleSMCRTC(1.0)[3F01C7A4-754F-39DD-A872-B4FAF0442276]@0xffffffff7f84be4000

dependency: com.apple.iokit.IOPCIFamily(2.9)[C08F7FC1-78A4-3A1B-BFE2-C07080CF2048]@0xffffffff7f83734000

dependency: com.apple.driver.AppleSMC(3.1.9)[259F0A4B-0AAB-30F3-8896-629A102CBD70]@0xffffffff7f83b9f000

com.apple.iokit.IOGraphicsFamily(517.22)[2AEA02BF-2A38-3674-A187-E5F610FD65B7]@0xffffffff7f84500000->0xffffffff7f84546fff

dependency: com.apple.iokit.IOPCIFamily(2.9)[C08F7FC1-78A4-3A1B-BFE2-C07080CF2048]@0xffffffff7f83734000

Es ist alles relativ

Wie bekommen wir nun raus, was da passiert ? Der Programmcode eines Kextes befindet sich für gewöhnlich in der Datei im MacOS Ordner des Kextes. Der Name der Datei steht in der Info.plist des Kextes und stimmt meist mit dem Namen des Kextes überein.

Wie kommt man nun von einer Rücksprungadresse zu einer Stelle im Programm bzw. der Datei ?

Man berechnet den Abstand zwischen der Rücksprungadresse und dem Anfang des Kextes. Dazu zieht man einfach die Startadresse des Kextes von der Rücksprungadresse ab.

Das sieht dann so aus

Backtrace (CPU 2), Frame : Return Address

```
0xffffffff91201531a0 : 0xffffffff8002e505f6
0xffffffff91201531f0 : 0xffffffff8002f7d604
0xffffffff9120153230 : 0xffffffff8002f6f0f9
0xffffffff91201532b0 : 0xffffffff8002e02120
0xffffffff91201532d0 : 0xffffffff8002e5002c
0xffffffff9120153400 : 0xffffffff8002e4fdac
0xffffffff9120153460 : 0xffffffff8002f6f2e9
0xffffffff91201535e0 : 0xffffffff8002e02120
0xffffffff9120153600 : 0x10d08
0xffffffff9120153720 : 0x28388
0xffffffff9120153760 : 0x280f0
0xffffffff91201537b0 : 0x28037
0xffffffff9120153810 : 0x257e5
0xffffffff9120153850 : 0x2Aa0c
0xffffffff9120153880 : 0x2F088
0xffffffff91201538d0 : 0x50134
0xffffffff9120153950 : 0x514d9
0xffffffff9120153990 : 0x521c2
0xffffffff91201539e0 : 0x4968c
0xffffffff9120153a20 : 0x4dded
0xffffffff9120153a80 : 0x062830
```



```

0xffffffff9120153b10 : 0x1e99
0xffffffff9120153b60 : 0x134c
0xffffffff9120153b80 : 0x14ca
0xffffffff9120153bb0 : 0x16c80
0xffffffff9120153bf0 : 0x17352
0xffffffff9120153c70 : 0x0f158
0xffffffff9120153cc0 : 0xffffffff8003464cff
0xffffffff9120153d10 : 0xffffffff80034c0ee7
0xffffffff9120153d70 : 0xffffffff8002f292f2
0xffffffff9120153dc0 : 0xffffffff8002e55c30
0xffffffff9120153e10 : 0xffffffff8002e32cbd
0xffffffff9120153e60 : 0xffffffff8002e45b7b
0xffffffff9120153ef0 : 0xffffffff8002f5952d
0xffffffff9120153fa0 : 0xffffffff8002e02926
Kernel Extensions in backtrace:
com.apple.iokit.IOACPIFamily(1.4)[8794C760-FDD9-3664-ADED-
4A9BBEC6E517]@0xffffffff7f83b96000->0xffffffff7f83b9efff
com.apple.driver.AppleACPIPlatform(6.1)[1804645B-B360-305E-B1BE-
916F5E3E1CC4]@0xffffffff7f85611000->0xffffffff7f856acfff
dependency:                com.apple.iokit.IOACPIFamily(1.4)[8794C760-FDD9-3664-ADED-
4A9BBEC6E517]@0xffffffff7f83b96000
dependency:                com.apple.driver.AppleSMCRTC(1.0)[3F01C7A4-754F-39DD-A872-
B4FAF0442276]@0xffffffff7f84be4000
dependency:                com.apple.iokit.IOPCIFamily(2.9)[C08F7FC1-78A4-3A1B-BFE2-
C07080CF2048]@0xffffffff7f83734000
dependency:                com.apple.driver.AppleSMC(3.1.9)[259F0A4B-0AAB-30F3-8896-
629A102CBD70]@0xffffffff7f83b9f000
com.apple.iokit.IOGraphicsFamily(517.22)[2AEA02BF-2A38-3674-A187-
E5F610FD65B7]@0xffffffff7f84500000->0xffffffff7f84546fff
dependency:                com.apple.iokit.IOPCIFamily(2.9)[C08F7FC1-78A4-3A1B-BFE2-
C07080CF2048]@0xffffffff7f83734000

```

Lesbar ist anders

In der Datei stehen einfach nur ein Haufen Bytes. Hin und wieder lässt sich ein Text erkennen, aber alles in allem kommt einem das spanisch vor, außer man ist Engländer, dann kommt es einem griechisch vor. Weiß jemand wie es einem Spanier oder einem Griechen vorkommt ?

Disassembler machen aus Maschinencode Assemblercode: Der ist besser lesbar, hält aber dem

Vergleich mit C oder einer anderen Hochsprache nicht stand.

Ich verwende Hopper um die Datei zu öffnen und zu disassemblieren, aber es geht auch mit Bordmitteln. Damit jeder die hiesigen Schritte nachvollziehen kann verwende ich deshalb den Terminal Befehl otool statt Hopper.

Alles eine Frage der Perspektive

Ok, wo fangen wir an ? Tief in der Hierarchie sind wir dem Fehler näher, aber erkennen möglicherweise vor lauter Bäumen den Wald nicht. Ganz oben sind wir möglicherweise so weit vom Wald weg, dass wir keine Bäume sehen. Also eeene meeene Miste..... oben, warum nicht, das sollte uns zumindest einen groben Hinweis geben worum es geht.

Die "oberste" Rücksprungadresse in einem Kext ist 0x0f158 in IOGraphicsFamily. Das ist ein systemeigenes Kext und somit in /System/Library/Extensions zu finden. Mit einem Rechtsklick auf das Kext, kann man sich den Paketinhalt anzeigen lassen. Und wie immer wenn man etwas zeigen will, verwendet dieses Kext nicht die "normale" Kext-Struktur, sondern schmeißt alle Dateien zusammen. Der Finder zeigt uns zwei ausführbare Dateien iogdiagnose und IOGraphicsFamily. Wir könnten nun in der Info.plist nachschauen, welche Datei, den Programmcode für das Kext enthält, aber ich wage es zu behaupten, dass es die Datei mit dem Namen des Kextes ist.

Zerleger

"otool, otool bitte, auf die Bühne". Wir öffnen das Terminal. Und schreiben

Code

1. otool -tV

Nach dem V folgt ein Leerschritt. -tV bedeutet Text-Segment anzeigen und mit Symbolen disassemblieren. Das Text Segment beinhaltet, anders als er der Name vermuten lässt, Programmcode.

Nun ziehen wir die IOGraphicsFamily Datei aus dem Finder auf das Terminal Fenster, das spart es und den Pfad zu tippen.

Code

1. otool -tV /System/Library/Extensions/IOGraphicsFamily.kext/IOGraphicsFamily

Dahinter kommt noch ein Leerschritt und die Angabe der Datei in die wir den Assemblercode geschrieben haben wollen. Da wir hier ein Unix Derivat haben kommt ein > vor den Dateinamen.

Code

```
1. otool -tV /System/Library/Extensions/IOGraphicsFamily.kext/IOGraphicsFamily
   >~/IOGF.txt
```

Drücken wir nun Eingabe haben wir das Assemblerlisting in der Datei IOGF.txt im Benutzerverzeichnis. Die Tilde ~bedeutet Benutzerverzeichnis.

Zuviel ist zuviel

Öffnen wir die Datei werden wir von der schieren Anzahl von Zeilen erschlagen. Aber Gott sei Dank, müssen wir uns nicht alle Zeilen anschauen, sondern nur die in der näheren Umgebung unserer Rücksprungadressen. 0x0f158 ist die Rücksprungadresse die uns Interessiert also suchen wir nach 000f158 im Text. das 0x lassen wir weg, weil das Format der Adressen in der Datei kein 0x enthält und die Nullen fügen wir hinzu um die Suche einzugrenzen.

Eine Code Zeile beginnt mit der Adresse, gefolgt vom Befehl und dessen Parametern und gegebenenfalls einem Kommentar.

Kalt

Code

```
1. Schnipp
2.
3.
4. 000000000000f147 callq __ZN13IOFramebuffer18setPlatformConsoleEP8PE_Videojy ##
   IOFramebuffer::setPlatformConsole(PE_Video*, unsigned int, unsigned long long)
5. 000000000000f14c movq (%r14), %rax
6. 000000000000f14f movq %r14, %rdi
7. 000000000000f152 callq *0x978(%rax)
8. -----
9. 000000000000f158 movl %eax, %ebx
10. -----
11. 000000000000f15a testl %ebx, %ebx
```

```

12. 000000000000f15c jne 0xf077
13. 000000000000f162 movq %r12, %rdi
14. 000000000000f165 callq __ZN23IOFramebufferUserClient8withTaskEP4task ##
    IOFramebufferUserClient::withTask(task*)
15. 000000000000f16a jmp 0xf182
16.
17.
18. Schnipp

```

Alles anzeigen

Rücksprungadressen zeigen hinter den Aufruf, denn dort soll das Programm ja weiter machen, wenn es aus dem Unterprogramm zurückkommt. Aufrufe von Unterprogrammen heißen call irgendwas.

Wir scrollen nun solange nach oben, bis wir einen Namen sehen, wo sonst Adressen stehen. Das ist der Name des Unterprogrammes dessen Code wir gerade sehen entdecken.

Code

```

1. Schnipp
2.
3.
4. 000000000000eeb0 popq %rbp
5. 000000000000eeb1 retq
6. __ZN13IOFramebuffer13newUserClientEP4taskPvjPP12IOUserClient:
7. 000000000000eeb2 pushq %rbp
8. 000000000000eeb3 movq %rsp, %rbp
9. 000000000000eeb6 pushq %r15
10.
11.
12. Schnipp

```

Alles anzeigen

`__ZN13IOFramebuffer13newUserClientEP4taskPvjPP12IOUserClient` ist C++ Compiler Gibberish. Es geht los mit der Klasse `IOFramebuffer`, dann kommt der Unterprogrammname, `newUserClient` und dann der Typ des Parameters, `IOUserClient`.

Wir wissen nun der Aufruf erfolgt sobald ein `Framebuffer` einen neuen `UserClient` anlegt. Die restlichen Texte zwischen diesem Namen und dem Aufruf sind genauso wenig hilfreich. Probieren wir es mit der nächst tieferen Rücksprungadresse.

Warm

Das ist die 0x17352. Suchen nach 00017352 und nach oben scrollen zum Unterprogrammnamen zeigt uns.

Code

```
1. Schnipp
2.
3.
4. __ZN13IOFramebuffer4openEv:
5. 0000000000017198 pushq %rbp
6. 0000000000017199 movq %rsp, %rbp
7. 000000000001719c pushq %r15
8. 000000000001719e pushq %r14
9. 00000000000171a0 pushq %r13
10. 00000000000171a2 pushq %r12
11. 00000000000171a4 pushq %rbx
12. 00000000000171a5 subq $0x48, %rsp
13.
14.
15. Schnipp
16.
17.
18. 0000000000017302 cmpq $0x0, 0x1e066(%rip)
19. 000000000001730a jne 0x17348
20. 000000000001730c movq 0xfdc5(%rip), %rax
21. 0000000000017313 movq (%rax), %rbx
22. 0000000000017316 leaq 0xcc56(%rip), %rdi ## literal pool for: "AppleClamshellState"
23. 000000000001731d xorl %esi, %esi
24. 000000000001731f callq 0x17324
25. 0000000000017324                                     leaq
    __ZN13IOFramebuffer16clamshellHandlerEPvS0_P9IOServiceP10IONotifier(%rip), %rdx
    ## IOFramebuffer::clamshellHandler(void*, void*, IOService*, IONotifier*)
26. 000000000001732b xorl %ecx, %ecx
27. 000000000001732d xorl %r8d, %r8d
28. 0000000000017330 movl $0x2710, %r9d
29. 0000000000017336 movq %rbx, %rdi
30. 0000000000017339 movq %rax, %rsi
31. 000000000001733c callq 0x17341
```

```

32. 0000000000017341    movq    %rax,    __ZL20glOFBclamshellNotify(%rip)    ##
    glOFBclamshellNotify
33. 0000000000017348    movl    $0xc, %edi
34. 000000000001734d    callq   __ZN13IOFramebuffer18readClamshellStateEy    ##
    IOFramebuffer::readClamshellState(unsigned long long)
35. -----
36. 0000000000017352    cmpl   $-0x1, 0x1d78b(%rip)
37. -----
38. 0000000000017359    jne    0x174a9
39. 000000000001735f    leaq   0xb05f(%rip), %rdi    ## literal pool for: "backlight"
40. 0000000000017366    xorl   %esi, %esi
41.
42.
43. Schnipp

```

Alles anzeigen

Höre ich da ein Aaah ? Wir sind in der Open Methode eines FrameBuffers. Aber wichtiger ist der Aufruf der zu unserer Rücksprungadresse gehört heißt readClamshellState. Entweder erkundigt sich macos nach der Qualität der Muscheln oder es hat etwas mit dem Zuklappen des Laptops zu tun.

Also auf zum nächsten Rücksprung, 0x16c80.

Heiß

Code

```

1. Schnipp
2.
3.
4. __ZN13IOFramebuffer18readClamshellStateEy:
5. 0000000000016bc6    pushq %rbp
6. 0000000000016bc7    movq  %rsp, %rbp
7. 0000000000016bca    pushq %r15
8. 0000000000016bcc    pushq %r14
9. 0000000000016bce    pushq %r13
10. 0000000000016bd0    pushq %r12

```

```

11. 0000000000016bd2 pushq %rbx
12. 0000000000016bd3 pushq %rax
13. 0000000000016bd4 movq %rdi, %r14
14. 0000000000016bd7 movl __ZL23gIOFBLastClamshellState(%rip), %r15d ##
    gIOFBLastClamshellState
15. 0000000000016bde movq
    __ZZN13IOFramebuffer18readClamshellStateEyE9lidDevice(%rip), %rdi ##
    IOFramebuffer::readClamshellState(unsigned long long)::lidDevice
16. 0000000000016be5 testq %rdi, %rdi
17. 0000000000016be8 jne 0x16c64
18. 0000000000016bea leaq 0xdacf(%rip), %rdi ## literal pool for: "PNPOCOD"
19. 0000000000016bf1 xorl %esi, %esi
20. 0000000000016bf3 callq 0x16bf8 laut Hopper IOService::nameMatching(char const*,
    OSDictionary*)
21. 0000000000016bf8 movq %rax, %rbx
22. 0000000000016bf9 movq %rbx, %rdi
23. 0000000000016bfe callq 0x16c03 laut Hopper
    IOService::getMatchingServices(OSDictionary*)
24. 0000000000016c03 movq %rax, %r13
25. 0000000000016c06 testq %rbx, %rbx
26. 0000000000016c09 je 0x16c14
27. 0000000000016c0b movq (%rbx), %rax
28. 0000000000016c0e movq %rbx, %rdi
29. 0000000000016c11 callq *0x28(%rax)
30. 0000000000016c14 testq %r13, %r13
31. 0000000000016c17 je 0x16c58
32. 0000000000016c19 movq (%r13), %rax
33. 0000000000016c1d movq %r13, %rdi
34. 0000000000016c20 callq *0x128(%rax)
35. 0000000000016c26 movq %rax, %r12
36. 0000000000016c29 leaq 0xda98(%rip), %rsi ## literal pool for: "IOACPIPlatformDevice"
37. 0000000000016c30 movq %r12, %rdi
38. 0000000000016c33 callq 0x16c38 laut Hopper OSMetaClassBase::metaCast(char const*)
    const
39. 0000000000016c38 testq %rax, %rax
40. 0000000000016c3b je 0x16c4e
41. 0000000000016c3d movq
    __ZZN13IOFramebuffer18readClamshellStateEyE9lidDevice(%rip) %r12, ##
    IOFramebuffer::readClamshellState(unsigned long long)::lidDevice
42. 0000000000016c44 movq (%r12), %rax
43. 0000000000016c48 movq %r12, %rdi

```

```

44. 00000000000016c4b callq *0x20(%rax)
45. 00000000000016c4e movq (%r13), %rax
46. 00000000000016c52 movq %r13, %rdi
47. 00000000000016c55 callq *0x28(%rax)
48. 00000000000016c58                                     movq
    __ZZN13IOFramebuffer18readClamshellStateEyE9lidDevice(%rip), %rdi    ##
    IOFramebuffer::readClamshellState(unsigned long long)::lidDevice
49. 00000000000016c5f testq %rdi, %rdi
50. 00000000000016c62 je 0x16c93
51. 00000000000016c64 movq (%rdi), %rax
52. 00000000000016c67 leaq 0xda6f(%rip), %rsi ## literal pool for: "_LID"
53. 00000000000016c6e leaq -0x2c(%rbp), %rdx
54. 00000000000016c72 xorl %ecx, %ecx
55. 00000000000016c74 xorl %r8d, %r8d
56. 00000000000016c77 xorl %r9d, %r9d
57. 00000000000016c7a callq *0x8d0(%rax)
58. -----
59. 00000000000016c80 testl %eax, %eax
60. -----
61. 00000000000016c82 jne 0x16c93
62. 00000000000016c84 xorl %eax, %eax
63. 00000000000016c86 cmpl $0x0, -0x2c(%rbp)
64. 00000000000016c8a sete %al
65.
66.
67. Schnipp

```

Alles anzeigen

Den Unterprogrammnamen brauchen wir nicht suchen, denn den kennen wir schon vom Aufruf, readClamshellState. Das sagt schon alles. Die Frage ist wie das Unterprogramm das macht. Wir versuchen nicht den Code zu analysieren, wir halten uns erst mal nur an die Texte. Wie jeder DSDT Junky weiß ist "PNP0C0D" die Kennung für ein Lid Device. Und wer es nicht weiß kann es googeln. Lid ist ein Deckel, eine Klappe, aber wir waren ja schon soweit, dass es was mit der Klappe zu tun hat. Wir können mit dem IORegistryEditor oder in der DSDT nachschauen, ob unser Rechner so ein klappriges Gerät hat. Er hat. Wir gehen also davon aus, dass das Programm nach dem Gerät mit dieser Kennung sucht - warum sonst würde die Kennung hier erwähnt. Nun stellt sich die Frage ob es das richtige Gerät ist, "IOACPIPlatformDevice" lässt vermuten, dass hier nachgeschaut wird ob es sich bei dem Gerät um ein IOACPIPlatformDevice handelt. Das sind ziemlich viele Vermutungen, aber sie lassen sich mit Hopper erhärten. Hopper liefert

mehr Informationen und zeigt, dass mit "PNP0C0D" IOService::nameMatching(char const*, OSDictionary*) und mit dessen Ergebnis IOService::getMatchingServices(OSDictionary*) aufgerufen wird. Das ist die Art, wie man nach einem Service(Gerät) mit dieser Kennung gesucht.

Hopper bestätigt auch, dass "IOACPIPlatformDevice" für einen DynamicCast und somit für eine Überprüfung des Gerätetyps verwendet wird.

Der nächste Text ist "_LID". Unterstrich und 3 Buchstaben schreit nach einem DSDT Namen. Also die DSDT geöffnet und danach gesucht und siehe da es gibt eine Methode namens _LID in dem Gerät LID0, das vom Typ PNP0C0D ist. Es scheint nicht zu gewagt zu sein, zu behaupten, dass hier die Methode _LID ausgeführt wird. Leider gibt auch Hopper an der Stelle keinen Namen aus. Das ist aber kein Problem, da die nächste Rücksprungadresse im Backtrace innerhalb des aufgerufenen Unterprogrammes liegt. Die nächste Rücksprungadresse ist 0x14ca in IOACPIFamily.

Wenn man nun AppleACPIPlatform mit otool disassembliert, wird man feststellen, dass die Rücksprungadresse in evaluateInteger liegt. Es wird also _LID ausgeführt und das Ergebnis als Integer interpretiert.

Land in Sicht

An der Stelle können wir sagen, dass der Crash dann erfolgt, wenn die _LID Methode des Gerätes vom Typ "PNP0C0D" aufgerufen wird. Das ist wie Amerika entdecken. Die Frage ist gehen wir an Land und erforschen den neuen Kontinent ? Wasser und Vorräte müssen wir nicht ergänzen, Anzeichen von Skorbut gibt es auch keine, wir müssen also nicht, wir können Amerika auch Columbus überlassen.

Einfach nur die Klappe halten

Da wir wissen wo es kracht, können wir überlegen wie wir es verhindern. Wenn das Gerät keine _LID Methode hat, kann es mit der _LID Methode keine Probleme geben. Oder man lässt ihn erst gar kein passendes Gerät finden.

Ich entscheide mich für die Methode mit der Methode.

Damit _LID nicht aufgerufen werden kann, wird die Methode einfach umbenannt. Mit einem passenden ACPI Patch macht das Clover sehr schön. Wir wollen _LID durch etwas anderes, sagen wir, BLID ersetzen. Der zu findende Wert ist 5F4C4944. Den ersetzen wir mit 424C4944.

Der Rechner wird jetzt nicht mehr auf ein Schließen der Klappe reagieren, aber hoffentlich nicht mehr crashen.

Und immer wieder geht die Sonne auf

Also config.plist angepasst. Neustart und - alles ist gut und Neustart und immer noch gut.

Der Fehler tritt nicht mehr auf, aber wir haben ihn nicht behoben, wir haben nur das Symptom bekämpft. Genau genommen haben wir nicht mal den Fehler gefunden, denn der tritt in einem der tieferen Unterprogramme auf, aber ein bootfähiges ist mir im Moment erstmal genug.

Hexenwerk

Der letzte Schritt, das Feststellen was passiert, scheint wie Hexenwerk und vielleicht etwas willkürlich. Das ist es aber nicht.

Wenn man sich aus dem Assemblercode einen Pseudocode (etwas das aussieht wie Quellcode, aber keiner ist) bastelt bekommt man so etwas wie

Code

1. Unterprogramm __ZN13IOFramebuffer18readClamshellStateEy
- 2.
- 3.
4. ZuSuchen = IOService::nameMatching ("PNP0C0D")
5. ListeDerGefundenGeräte = IOService::getMatchingServices(ZuSuchen)
- 6.
- 7.
8. ErsterEintragInDerListe bestimmen
- 9.
- 10.
11. Wenn es keinen gibt hau ab.
- 12.
- 13.
14. ACPIPlatformDevice = OSDynamicCast (IOACPIPlatformDevice, ErsterEintragInDerListe)
- 15.
- 16.
17. Ist der Null dann hau ab
- 18.
- 19.
20. ACPIPlatformDevice.evaluateInteger("_LID");

Alles anzeigen

Wenn man Erfahrung mit IOKit hat, erkennt man die Vorgehensweise. Das ist Standard, wie man halt ein Gerät eines Gerätetypes, der in der DSDT vorkommt, findet, hat also nicht mit Gerate oder Hellsehen zu tun.

Raten oder "Sich-Denken" muss man nur, wen einem das Wissen fehlt, führt aber zum selben Ziel - zumindest, wenn man richtig rät.

Auf Regen folgt der Sonnenschein

Nächstes Mal wird's wieder einfacher und alltagstauglicher.

Beitrag von „andreas_55“ vom 1. Februar 2018, 20:33

Zitat von Brumbaer

für einen Imbus Schraubendreher gedacht, den vermutlich Ameisen für ihre Uhren verwenden.

ooooh, Brumbaer.

Diese Schreibe,... und dann schließt das ganze noch mit einem Cliffhänger. 👍

Und jetzt höre ich auf zu lesen und freue mich auf morgen Abend mit dem dritten Teil...

Danke.

Beitrag von „derHackfan“ vom 1. Februar 2018, 20:55

Jo, sehr interessant ... nur leider heisst das Ding Inbus. 🐜

Bauer und Schaurte: -> <https://de.wikipedia.org/wiki/Innensechskant>

Beitrag von „coopter“ vom 1. Februar 2018, 21:02

[@Brumbaer](#)

Hut ab , 😄 (Auch wenn meine Kompetenz hier nichtig ist .)
Lyrik und Wissenschaft !
Das als Buch und ich kaufe es sofort. 😄

Beitrag von „McRudolfo“ vom 1. Februar 2018, 21:17

[@Brumbaer](#)

da bin ich ganz bei [@coopter](#): einfach großartig geschrieben! Inhalt und Stil - einfach Klasse!
Ich freue mich schon auf den nächsten Teil! 👍

Beitrag von „andreas_55“ vom 2. Februar 2018, 20:39

Zitat

Dazu zieht man einfach die Startadresse des Kextes von der Rücksprungadresse ab.

Habe mal die erste rote Return Adress genommen und den Startspeicherbereich von IOACPIFamiliy:

$0xffffffff7f85621d08 - 0xffffffff7f83b96000 = 0x1A8BD08 \neq 0x10d08$

Was mache ich denn hier falsch?

Edit: Hab was vergessen: Cooler dritter Teil. 👍

Beitrag von „coopter“ vom 2. Februar 2018, 21:45

So , ein Innensechskantschraubenschlüssel der Marke Inbus (S/B) 😬

Zum Glück kein Nimbus . 😊

Beitrag von „Brumbaer“ vom 2. Februar 2018, 23:31

[Zitat von andreas 55](#)

Habe mal die erste rote Return Adress genommen und den Startspeicherbereich von IOACPIFamily:

$0xffffffff7f85621d08 - 0xffffffff7f83b96000 = 0x1A8BD08 \neq 0x10d08$

Was mache ich denn hier falsch?

Nichts.

Ich habe unten bei den Kexten die Farben vertauscht. Das ist alles.
Vielen Dank fürs aufmerksame lesen.

cu

Beitrag von „Dr.Stein“ vom 2. Februar 2018, 23:44

[@Brumbaer](#)

Habe ich es jetzt richtig verstanden... er läuft? 100% 😊

Beitrag von „Brumbaer“ vom 3. Februar 2018, 00:04

Er läuft als Laptop, aber bei den Sonderfunktionen, wie Touch, Kameras und LTE sehe ich mitternachtsblau.

Es folgen noch drei oder vier Artikel, die das Einrichten der Laptop Funktionen beschreiben, und dann muss ich mich entscheiden, ob ich Zeit in Touch, Kameras und LTE investieren will. Was ohne Datenblätter der Chips/Bauteile extrem aufwändig wäre. Leider sind Kameras und Touch über I2C angebunden, da besteht wenig Hoffnung einen Treiber zu finden. Für die Kameras kann man vielleicht einen Linux Treiber, an dem man sich orientieren kann, finden, für Touch sieht es da schlecht aus. Aber ich habe ja noch etwas Zeit.

Beitrag von „al6042“ vom 3. Februar 2018, 00:41

Wegen der Touch-Geschichte solltest du mal mit [@BlackOSX](#) in Kontakt treten... Sein Xiaomi hat auch ein Touchpad an I2C hängen und bei ihm tut es wohl...

Beitrag von „Brumbaer“ vom 3. Februar 2018, 00:54

Das Touchpad funktioniert. Ich spreche vom Touchscreen von Wacom. Aber vielleicht gibt es einen HID Modus der sich ansprechen lässt.

Googlen zeigt es gibt wohl ein VoodooI2C das auch Touchscreens unterstützt. Schau ma mal.

Beitrag von „al6042“ vom 3. Februar 2018, 01:06

Oh... "Touch" und "Touch" sind an der Stelle dann zwei verschiedene Sachen...
Upps... 😊

Beitrag von „Brumbaer“ vom 3. Februar 2018, 13:22

Dumdidum

Dumdidum, Dumdidum, Dumdidum, Dumdidum.

Oberwasser, ich habe Oberwasser, denn nun bootet der Miix endlich, wie er soll, wann er soll, Dumdidum.

Ja, ja, ja, jetzt wird wieder in die Hände gespuckt!

Und als nächstes, werden all die Kexte, Patches und Häkchen, die ich in meiner Verzweiflung hinzugefügt hatte, wieder entfernt, Dumdidum.

Hinterher, sehen Config.plist, Other und drivers64UEFI Ordner ganz normal aus und - das System bootet immer noch. Bevor ich es vergesse: Dumdidum.

The sound of music

So jetzt was Einfaches, Sound. Der Miix hat einen ALC298. Das weiß ich von Blick durchs Fenster, aber ich hätte es auch im IORegistryExplorer feststellen können.

Ich bin ein großer AppleALC Fan und er hat mich bisher noch nicht im Stich gelassen. AppleALC braucht Lilu, aber das habe ich schon im Other Ordner.

Nach AppleALC gegoogelt und da wird ein Link zu den "Supported" Codecs angeboten und - Dumdidum - die Layouts 3, 11, 13, 28, 29, 47 und 72 werden angeboten. Wo fängt man am besten an ? Wie wäre es mit dem Anfang. Ruckzucki, Layout Id 3 in der Config gesetzt, Neustart und Booti - Dumdidum - Systemsteuerung, Ton, sieht gut aus, mal ein Böng und mal ein Bing und mal ein Tschingderassabum.

Plaudertasche

Strg-Leerschritt und nichts. Immer noch Dumdidum, aber eine Oktave tiefer, etwas knurriger, ich bin scheinbar leicht zu verstimmen. Siri angeklickt, "Öffne iTunes", Siri öffnet iTunes. Dumdidum, wieder zurück in normaler Tonlage.

Siri, kann die Systemeinstellungen öffnen, aber nicht ihr eigenes Kontrollfeld, das BT Kontrollfeld hingegen schon. Egal. Die Siri-Öffnen-Taste auf Strg Leerschritt gelegt und - geht doch.

Es kommen Töne raus, es gehen Töne rein, Layout Id 3 muss es sein.

Trari-Trara, die Post ist da

Zufälle gibt's, da bringt am selben der Tag der Postbote doch tatsächlich die DW1560. Es ist dem Postboten gegenüber ungerecht von zufällig zu reden, der ist sehr zuverlässig.

"6 Schrauben, 6 Schrauben, die gehen leicht auf, der Rechner, der Rechner, der öffnet seinen Bauch". Auch nicht besser als Dumdidum, aber noch habe ich Oberwasser.

Ok, Original raus, DW1560 rein. Antennen angeschlossen, Bildschirm hochgehalten, Rechner eingeschaltet und es kommt ein Logo, eigentlich ein Grund zur Freude, wenn es nicht so hässlich wäre, aber nicht hässlich genug um mir die Laune zu verderben. Dumdidum.

Neustart, weder WiFi noch Bluetooth. Keine Überraschung, ist ja kein Standard Mac Gerät. Ich habe mit Absicht eine Lenovo Variante genommen, für den Fall, dass es eine Whitelist gibt.

IORegistry Editor zeigt aber, dass die Teile erkannt werden. Falls man keine Mac kompatible Karte hat, ist der Standardweg für WiFi die Verwendung von FakePCIID und FakePCIID_Broadcom_WIFI und für Bluetooth die Verwendung von BrcmPatchRAM2 und BrcmFirmwareData.

Reinkopiert und siehe da WiFi WiFi, aber BT BTed nicht.

Laut IORegistryExplorer handelt es sich um einen BCM20702A mit der VendorId 0x489 und der ProductId 0xe07a. Das sind USB Ids, äquivalent zur PCIId.

Die DeviceClass ist 0xFF, das ist seltsam normalerweise ist sie bei BT Geräten 0xE0.

"Kein Problem", denke ich noch, "trägst den Wert in BrcmPatchRAM2 nach". Gesagt, getan und was auch immer ich mache, es geht nicht. BrcmPatchRAM2 und BrcmFirmwareData wieder raus.

Da gibt's doch ne App für, oder en Kext ?

Apples BT Treiber befinden sich als PlugIns im IOBluetoothFamily.kext. PlugIns sind Kexte, die im PlugIns Ordner anderer Kexte abgelegt sind.

- BroadcomBluetoothHostControllerUSBTransport.kext
- BroadcomBluetooth20703USBTransport.kext
- CSRBluetoothHostControllerUSBTransport.kext
- CSRHIDTransitionDriver.kext
- IOBluetoothHostControllerTransport.kext
- IOBluetoothHostControllerUARTTransport.kext
- IOBluetoothHostControllerUSBTransport.kext
- IOBluetoothSerialManager.kext
- IOBluetoothUSBDFU.kext

Die CSR können aus der Liste raus (nicht aus dem Ordner), falscher Hersteller, da waren es nur noch 7.

UART und Serial Manager haben mit speziellen Funktionen zu tun, können auch raus, da waren es nur noch 5.

IOBluetoothHostControllerTransport scheint nichts mit USB zu tun zu haben, also erst mal raus, da waren es nur noch 4.

- BroadcomBluetooth20703USBTransport.kext
- BroadcomBluetoothHostControllerUSBTransport.kext
- IOBluetoothUSBDFU.kext
- IOBluetoothHostControllerUSBTransport.kext

Mit plist und Tücke

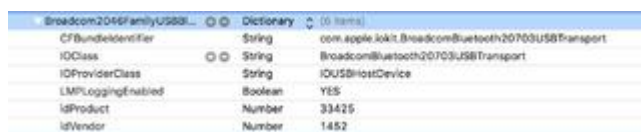
Kexte, die als Gerätetreiber dienen, haben in ihrer Info.plist meist IOKitPersonalities Einträge, in denen die Geräte identifiziert werden, die sie treiben sollen.

Fangen wir mit der Info.plist des BroadcomBluetoothHostControllerUSBTransport.kext an. Tausende Einträge, alle rufen dieselbe Klasse auf und unterscheiden sich durch idProduct und idVendor.

Die Controller basieren alle auf den 2045 und 2046 Serien von Broadcom also laut Google nur BT2.0 und BT 2.1. "Ihrrrr seieieieied rrrraus".

Das nächste Broadcom Kext ist BroadcomBluetooth20703USBTransport.

Die Einträge unterscheiden sich wieder nur in idProduct und idVendor. Der folgende Eintrag ist typisch:



Key	Type	Value
CFBundleIdentifier	String	com.apple.iokit.BroadcomBluetooth20703USBTransport
IOClass	String	BroadcomBluetooth20703USBTransport
IOProviderClass	String	IOUSBHostDevice
LMPLoggingEnabled	Boolean	YES
idProduct	Number	33425
idVendor	Number	1452

Alle verwenden eine Klasse, die BroadcomBluetooth20703USBTransport heißt. Dank Google wissen wir, dass der Chipsatz 20703 BT4.1 unterstützt, der 20702 aber nur BT4.0. Aufeinanderfolgende Teilenummer beim selben Hersteller, da kann man den Treiber schon mal probieren, kann funktionieren, muss aber nicht.

Also lassen wir für das BT Modul diesen Treiber laden. Sowas haben wir schon mal für den USB Treiber gemacht. Der Eintrag wird kopiert und in das USB Kext im Other Ordner eingesetzt. Es

muss nicht das USB Kext sein, man kann auch ein neues Kext machen, oder irgendein anderes nehmen, aber ich habe das Kext genau für solche Anpassungen eingeplant also hinein. idProduct und idVendor müssen noch angepasst werden.

IOKitPersonalities	Dictionary	(3 items)
BT	Dictionary	(7 items)
CFBundleIdentifier	String	com.apple.iokit.BroadcomBluetooth20703USBTransport
IOClass	String	BroadcomBluetooth20703USBTransport
IOProviderClass	String	IOUSBHostDevice
LMPLoggingEnabled	Boolean	NO
idProduct	Number	57466
idVendor	Number	1161
IOProbeScore	Number	2000
Air7,1	Dictionary	(7 items)

BrcmPatchRAM2 und BrcmFirmwareData werden nicht gebraucht. Neustart und WiFi und BT. Läuft, läuft bei mir.

PS.
Dumdidum

Beitrag von „Dr.Stein“ vom 3. Februar 2018, 14:55

Sound, wlan und BT :p
glückwunsch 😊

Beitrag von „ebs“ vom 3. Februar 2018, 15:44

Das ist ja spannender als ein Krimi. Freue mich schon auf die nächste Folge. Technische Hintergründe werden wunderbar erklärt. Deshalb bin ich für die Einführung des Rangs **rofessor** Brumbear.

Beitrag von „Dr.Stein“ vom 3. Februar 2018, 15:49

[@ebs](#)

Das wird nix. Nicht weil wir nicht wollen sondern wegen anderer Gründe

Beitrag von „Harper Lewis“ vom 3. Februar 2018, 16:49

Besten Dank, läuft auch bei mir. Meine BCM94352ZAE_3 / DW1560 hat die vendor-id 0xa5c (2652) und die product-id 0x216f (8559). Chipsatz: 20702A3.

Beitrag von „Bernd.H“ vom 3. Februar 2018, 18:34

[@Brumbaer](#) Du hast ja gar keine Zeit mehr zu arbeiten oder bist schon Rentner ODER hast du deswegen, wegen der Sache hier, gekündigt ? 😊

Beitrag von „andreas_55“ vom 3. Februar 2018, 20:31

...oh, das ist gut.

Ich habe ja schon meinen eigenen USB-Kext für den Coffee-Lake (nach-)gebaut, aber gerade eben ist mir nach Teil 4 nochmal klar geworden, dass ja meine Abhängigkeit von anderen Kexten, wie z.B. den BrcmPatch... und BrcmFirmware... (vielleicht) gar nicht mehr so da ist. Ich kann ja meinen USB-Kext um den notwendigen Treiber ergänzen, da ich nun weiß wo, ich zu suchen habe. Und auf die Idee kommen, falls mein Modell zu neu ist, Vorgänger(treiber)modelle mit auszuprobieren. Dann kann also der USB-Kext entsprechend angepasst auch noch WiFi und BT abdecken. Ein weiterer Schritt in die Unabhängigkeit. 😊

Und aus Deinem sehr analytischen Herangehen an das Log und die Rücksprungadressen und das Finden von PNP... und _LID nehme ich mit, dass sich für uns das tiefe Buddeln lohnt und man sich dann auch einfach trauen sollte, z.B. so eine Methode auch mal (probeweise)

lahmzulegen. Das geht ja dann wieder mit Clover-ACPI-Patch sehr gut.

Und keine Angst vor langen Hex-Zahlen, die können alle z.B.in den Apple-Taschenrechner (im Programmierer-Modus) einfach reinkopiert werden und dann kann man ganz einfach damit arbeiten (0xfffff7f8450f158 - 0xfffff7f84500000 = 0x0f158 und von dieser Adresse aus führte der Weg ins Licht).

Na ja, ... wenn man nun noch weiß, dass es sowas wie „otool“ gibt und die Syntax kennt, oder „Hopper“ und ...

... oder man fragt doch besser jemanden wie Brumbaer.

Beitrag von „Harper Lewis“ vom 4. Februar 2018, 09:53

WiFi läuft auf dem Lenovo aber (noch) mit FakePCIID und FakePCIID_Broadcom_WIFI, oder?

Beitrag von „Brumbaer“ vom 4. Februar 2018, 09:57

Ja, der Treiber fragt wohl die Register ab und braucht deshalb FakePCIID.

Beitrag von „Harper Lewis“ vom 4. Februar 2018, 11:12

Komischerweise wird der Brcm4360-Treiber bei mir mit neueren Clover-Versionen nicht mehr geladen, das Device taucht dann auch nicht mehr im IOReg auf. Mit einer älteren Clover-Version (39irgendwas) funktioniert das mit FakePCIID und FakePCIID_Broadcom_WIFI hingegen prima. Alles unter 10.12.6.

Beitrag von „jboeren“ vom 4. Februar 2018, 11:20

Ich lese mit! Ich drück dir die Daumen [@Brumbaer!!!](#)

Beitrag von „Brumbaer“ vom 6. Februar 2018, 22:51

Das Leben geht weiter

Dumdidum war gestern. Das Oberwasser ist zurückgegangen.

Jetzt, da ich eine Netzwerkverbindung habe, kann ich die Bildschirmfreigabe verwenden.

Ich mache das gerne, denn ich arbeite am liebsten an meinem großen Rechner.

Der Bildschirm ist da wo ich ihn gut sehen kann, die Tastatur ist do, wo ich gut tippen kann und das Trackpad ist da, verd. .. wo ist mein Trackpad ?

Wiedergefunden, in einem ordentlichen Haushalt kommt halt nichts weg.

Außerdem lassen sich mit der Bildschirmfreigabe mal schnell Dateien kopieren. In diesem speziellen Falle hat die Bildschirmfreigabe den zusätzlichen Vorteil, dass Tastatur und Maus beim ferngesteuerten Rechner nicht per USB angebunden sind. Nach einem Wake funktionieren Tastatur und Maus nicht mehr, obwohl die Geräte noch in der IORegistry stehen. Das externe USB Laufwerk geht interessanterweise noch, BT scheint auch nicht zu gehen. Also mal schnell die Kennung der USB Ports auf extern gestellt und die Gleichstellung von internen und externen USB Anschlüssen belegt.

Das böse Erwachen lässt sich auch nicht durch das Dunkle Erwachen beeinflussen.

Hibernatemode, sollte nichts ändern, aber man kann's ja mal versuchen. Man versucht's und es ändert nichts.

Der Unterschied zwischen in- und externen Gerät ist, dass das externe Gerät ein eigenes Netzteil hat und nicht am internen Akku hängt. Ich will jetzt nicht hängen bleiben, weder am Akku, noch einem Sleep/Wake Problem, also zur Seite geschoben.

Problemkind

Ich habe auf mehr als zehn Motherboards macos installiert und noch nie solch eine Anhäufung von Problemen erlebt, sollte irgendetwas bei der Installation auf dem Miix einfach nur so funktionieren, mache ich ein dickes Kreuz in den Kalender.

(Duracell)Hase oder Meister Lampe ?

Batterie oder Hintergrundbeleuchtung ? Hintergrundbeleuchtung sollte einfacher sein und ich bin für jedes Erfolgserlebnis dankbar. Aber man bekommt so selten, was man sich wünscht,

also Batterieverwaltung.

Nichts einfacher als das

"Da legst du das Kext vom Rehabman in den Other Ordner und fertig", habe ich verstanden, ist ja ganz einfach. Nach dem Kext gegooglelt, gedownloadet und in den Other Ordner gepackt, Neustart, Energie sparen öffnen. Ladezustand 0% und ein klicken auf *Batteriestatus in der Menüleiste anzeigen* setzt den Haken nicht. Sieht nach Brandschutzklasse B1 aus, selbstlöschend.

Ich merke schon, das wird heute kein Spass.

Wer lesen kann, ist klar im Vorteil

Rehbmans Artikel gelesen und da steht irgendwo, dass man die DSDT patchen muss, wenn Felder in der Operationregion des Embedded Controllers länger als 8 Bit sind.

Also einen Blick in die DSDT werfen.

Darin Steht Dieser Text

Code

1. Schnipp
- 2.
- 3.
4. OperationRegion (ECF2, EmbeddedControl, Zero, 0xFF)
5. Field (ECF2, ByteAcc, Lock, Preserve)
6. {
7. XXX0, 8,
8. XXX1, 8,
9. XXX2, 8,
10. Offset (0x14),
11. VCMD, 8,
12. VDAT, 8,
13. VSTA, 8,
14. Offset (0x20),
15. RCMD, 8,
16. RCST, 8,
17. Offset (0x60),
18. TSR1, 8,
19. TSR2, 8,
20. TSR3, 8,

21. TSI, 4,
22. HYST, 4,
23. TSHT, 8,
24. TSLT, 8,
25. TSSR, 8,
26. CHGR, 16,
27. Offset (0x6A),
28. Offset (0x6B),
29. Offset (0x6C),
30. Offset (0x6D),
31. Offset (0x6E),
32. TSRT, 8,
33. Offset (0x72),
34. CHGT, 8,
35. NPST, 8,
36. PCVL, 8,
37. Offset (0x7F),
38. LSTE, 1,
39. Offset (0x80),
40. ECWR, 8,
41. XX10, 8,
42. XX11, 16,
43. B1DC, 16,
44. B1FV, 16,
45. B1FC, 16,
46. XX15, 16,
47. B1ST, 8,
48. B1CR, 16,
49. B1RC, 16,
50. B1VT, 16,
51. BPCN, 8,
52. Offset (0xC0),
53. MGI0, 8,
- 54.
- 55.
56. Schnapp

Alles anzeigen

Treiber greifen auf die Register von Schaltkreisen (Hardware-Register) zu. Die Register sehen nach außen hin aus wie normale Speicherzellen. Aber tatsächlich spiegeln die Bits und Bytes

Schaltzustände des Schaltkreises wieder und/oder lösen Vorgänge und Abläufe aus statt sich nur Werte zu merken. Diese Register stehen für gewöhnlich an festen Adressen. So hat jede PCIe Karte einen für ihre Zwecke reservierten Speicherbereich. In diesem gibt es einen festen Bereich für die PCIe Konfigurationsregister. In diesem sind die ersten Register für die VendorId und ProductId vorgesehen.

Auch einige BIOS-Variablen stehen an festen Speicherstellen.

Der Befehl *OperationRegion* erzählt dem BIOS von solchen Speicherbereichen und ermöglicht dadurch den Zugriff auf Systemvariable und Hardware-Register. Wir suchen den *OperationRegion* Befehl, der den Zugriff auf den Speicherbereich *EmbeddedControl* ermöglicht, weil Rehabman schreibt, dass es um die geht. *EmbeddedControl* ist einer von 9 in der ACPI Spezifikation definierten Standardspeicherbereichen. Ein weiterer dieser Bereiche ist *PCI_Config*, der Zugriff auf die erwähnten Konfigurationsregister erlaubt.

Ich kann die Lektüre der ACPI Spezifikation nur jedem empfehlen. 1036 Seiten purer Poesie, vollgepackt mit spannender Action und herzerreißenden Dramen, ein Muß für jeden, jeden pffffff, ich komme später darauf zurück.

Denen, deren Herz der Spezifikation nicht standhalten würde, sei gesagt, dass der Befehl am Anfang eine Operationregion mit dem Namen ECF2 definiert, die die 255 Bytes ab Adresse 0 im Speicherbereich des EmbeddedControllers belegt.

Ordnung ist das halbe Leben

ECF2 ist einfach nur ein Speicherbereich von 255 Bytes. Das macht den Zugriff auf Register, Funktionen und Schaltzustände zwar möglich, aber nicht einfach.

Mit dem *Field*-Befehl, kommen Ordnung und Übersichtlichkeit in die Sache. Mit dem Befehl weist man einzelnen Speicherzellen bzw. Bitbereichen Namen zu, bevorzugt solche, die die Funktion beschreiben.

Ein Eintrag im *Field*-Befehl besteht aus dem Register Namen und der Länge des Registers in Bits. Das nächste Register beginnt and dem Bit, das auf das letzte des vorhergehenden Registers folgt. Statt eines Namens kann dort auch ein *Offset*-Befehl stehen. In diesem Fall beginnt das nächste Register bei Bit 0 des Bytes, das so viele Bytes vom Anfang des Speicherbereichs entfernt ist, wie der Wert in Klammern angibt.

Die erwähnten Konfigurationsregister, die jeweils 16 Bit lang sind, würden wie folgt definiert und abgefragt.

Code

1. // Ein Bereich von 64 Byte am Anfang des PCI Config Space.
2. // Die Länge ist nicht wirklich wichtig, solange die Register rein passen.
3. // Laut PCI Spezifikation ist der "normale" Config-Space 64 Byte lang

4. OperationRegion (CONF, PCI_Config, Zero, 64)
5. Field (CONF, ByteAcc, Lock, Preserve)
6. {
7. VNID, 16, // Vendor Id 16 Bit lang
8. PRID, 16, // Product Id 16 Bit lang
9. }
- 10.
- 11.
12. Method (GTVE, 0, NotSerialized) // Methode gibt den Inhalt des Registers mit der VendorId zurück
13. {
14. return (VNID)
15. }

Alles anzeigen

Lange Kerls

Uns interessieren, die Register in der Miix DSDT, die länger als 8 Bit sind. Das sind in der DSDT des Miix 9 Stück. Von denen interessieren uns nur die, die auch benutzt werden. Im Suchfeld von MaciAsl, kann man sehen, wie oft das Suchwort gefunden wurde. Wenn da eine 1 steht, kann man das Register ignorieren. Jedes der 9 Register einmal suchen und es bleiben 6 übrig. AML kennt nur wenige "Schreib-Befehle". Store ist einer, wie der Name schon sagt. Store (denWert, nachHier) - speichert den Wert des ersten Parameters im zweiten. Mathematische Operationen können mit zwei oder drei Parametern - Add (das, zudem, speichereHier) - aufgerufen werden. Werden sie mit drei Parametern aufgerufen, so wird das Ergebnis der Operation im letzten Parameter gespeichert. Schauen wir uns die Stellen an, an denen die Register verwendet werden, stellen wir fest, dass die Register in der DSDT immer nur gelesen werden. Alle langen Kerls haben 16 Bit Länge und werden immer nur gelesen. Das macht das Leben einfacher, denn wir müssen uns nur mit Lesebefehlen von jeweils 2 Byte Länge beschäftigen.

Divide et impera

Nun kann Apples ACPIInterpreter, wohl an dieser Stelle nicht mit Bitlängen über 8 arbeiten. Also muss man jeden 16 Bit Zugriff in zwei 8 Bit Zugriffe aufteilen und das Ergebnis zu einem 16 Bit-Wert zusammenfügen. Rehabman zeigt eine Möglichkeit, die mir aber nicht liegt. Ich will nicht so viel tippen und jedes Register in zwei zu zerlegen ist umständlich und dann beim Lesen jedesmal beide Teilvariablen anzugeben auch. Details findet man in Rehabman's Hilfethread zum Thema Batterieanzeige.

Vitamin C

Ich verwende eine bei C gebräuchliche Methode. Ich lege ein Byte Array über den Speicherbereich und greife darüber auf einzelne Bytes zu. Beim Lesen einer Variable muss ich nur den ersten Offset angeben und ich muss die Original Felddefinition nicht ändern bzw. eine neue anlegen.

Die Methode, um ein 16 Bit Wort zu lesen, soll RDWD heißen, und hat einen Parameter. den Offset des ersten Bytes vom Beginn der Operationregion:

Code

1. Method (RDWD, 1, NotSerialized) // definiere Methode RDBY die einen Parameter hat.
2. {
3. OperationRegion (BYAR, EmbeddedControl, Arg0, 2) // Lege eine OperationRegion an, die im EmbeddedControl Speicherbereich an der durch den Parameter
4. // angegebenen Adresse liegt und 2 Byte lang ist.
5. Field (BYAR, ByteAcc, NoLock, Preserve) // der Speicherbereich wird in Low und High Byte unterteilt
6. {
7. BLOW, 8,
8. BHGH, 8,
9. }
- 10.
- 11.
12. Name (TEMP, Buffer (0x02) { }) // Lege einen 2 Byte Buffer an
13. Store (BLOW, Index (TEMP, Zero)) // Kopiere das Low-Byte des Speicherbereichs in das erste Byte des Buffers
14. Store (BHGH, Index (TEMP, One)) // Kopiere das High-Byte des Speicherbereichs in das zweite Byte des Buffers
15. Return (TEMP) // Gib den Buffer aus, der dann von der aufrufenden Methode als 16 Bit Wert interpretiert wird.
16. }

Alles anzeigen

Die RDWD-Methode schreibt man in der DSDT, direkt hinter den Field Befehl für die Operationregion ECF2.

Abstand halten

Jetzt braucht's nur noch die Offsets der Register. Das ist einfach: Der Offset-Befehl direkt vor ECWR sagt uns, dass ECWR bei Byte 0x80 steht. XX10 ist 8 Bit, ein Byte, weiter also bei 0x81.

XX11 wieder 8 Bit weiter, also bei 0x82. XX11 ist 16 Bit lang, somit ist B1DC 2 Byte weiter, bei 0x84.

Die Liste der Felder mit ihren Abständen (Offsets)

Code

1. Schnipp
- 2.
- 3.
4. Offset (0x80),
5. ECWR, 8, // 0x80
6. XX10, 8, // 0x81
7. XX11, 16, // 0x82
8. B1DC, 16, // 0x84
9. B1FV, 16, // 0x86
10. B1FC, 16, // 0x88
11. XX15, 16, // 0x8A
12. B1ST, 8, // 0x8C
13. B1CR, 16, // 0x8D
14. B1RC, 16, // 0x8F
15. B1VT, 16, // 0x91
16. BPCN, 8, // 0x93
17. Offset (0xC0),
18. MGI0, 8, // 0xC0
- 19.
- 20.
21. Schnapp

Alles anzeigen

Nun verwenden wir Suchen und Ersetzen um jeden Aufruf von B1DC, außer dem im Field Befehl, durch RDWD(0x84). Entsprechend geht man bei den anderen Registern vor.

Bei der Gelegenheit sehen wir, dass die Aufrufe im Gerät BAT0 erfolgen.



Das ist dann wohl unsere Batterie. BAT0 könnte natürlich auch für Fledermaus 0 stehen, aber wenn mein Leben davon abhängen würde, würde ich eher auf Batterie tippen.

Warum die Felder B1XX heißen, wenn die Batterie Nummer 0 ist, ist eines der vielen Lenovo Geheimnisse. Es gibt noch mehr BAT Geräte, aber offensichtlich sind sie nur Platzhalter.

Beim Durchsuchen der SSDTs findet man in der SSDT-8 Zugriffe auf einige der Register. Also muss man auch in der SSDT-8 die Zugriffe durch RDWDs ersetzen. Das passiert ebenfalls mit suchen und Ersetzen. Allerdings muss man die Methode noch zu den External Deklarationen hinzufügen, damit MaciASL weiß, dass es sie gibt..

Code

1. Schnipp
- 2.
3. External (_SB_.PCI0.I2C1.TPL1, DeviceObj) // (from opcode)
4. External (_SB_.PCI0.LPCB.H_EC, DeviceObj) // (from opcode)
5. External (_SB_.PCI0.LPCB.H_EC.RDWD, MethodObj)
6. External (_SB_.PCI0.LPCB.H_EC.B1CI, UnknownObj) // (from opcode)
7. External (_SB_.PCI0.LPCB.H_EC.B1DC, UnknownObj) // (from opcode)
8. External (_SB_.PCI0.LPCB.H_EC.B1DI, UnknownObj) // (from opcode)
- 9.
10. Schnapp

Die DSDT im patched Ordner speichern und Neustarten.

Close, but no cigar

Die gute Nachricht ist, dass sich das Batteriesymbol in der Menüleiste anzeigen lässt, die schlechte ist, dass der Ladezustand immer 0% ist. Netzteil abgeklemmt, Ladezustand immer

Ich könnte der DSDT eine _BIX Routine hinzufügen, oder erst einmal einen Blick in die Quellen des AppleSmartBatteryManager werfen.

Sucht man in den Quellen nach _BIX findet man eine ganze Menge, darunter folgendes:

Code

```
1. Schnipp
2.
3.
4. /*****
5. * AppleSmartBatteryManager::getBatteryBIX
6. * Call DSDT _BIX method to return ACPI 4.x battery info
7. *****/
8.
9.
10. IOReturn AppleSmartBatteryManager::getBatteryBIX(void)
11. {
12.     DebugLog("getBatteryBIX called\n");
13.
14.
15. Schnapp
```

Alles anzeigen

Das ist offensichtlich die Methode, die die _BIX Methode der Batterie liest. Suchen wir nach getBatteryBIX finden wir nur einen Aufruf der Methode, der Rest gehört zu deren Definition.

Code

```
1. Schnipp
2.
3.
4. fProvider->getBatterySTA();
5. if (fBatteryPresent)
6. {
7.     if (fUseBatteryExtendedInformation)
8.         fProvider->getBatteryBIX();
9.     else
10.        fProvider->getBatteryBIF();
```

11. if (fUseBatteryExtraInformation)
12. fProvider->getBatteryBBIX();
13. fProvider->getBatteryBST();
14. }
- 15.
16. Schnapp

Alles anzeigen

Man sieht, dass getBatteryBIX nur aufgerufen wird, wenn fUseBatteryExtendedInformation wahr ist. Ansonsten wird _BIF verwendet. Also nach fUseBatteryExtendedInformation suchen und wir finden:

Code

1. Schnipp
- 2.
- 3.
4. fUseBatteryExtendedInformation = useExtendedInformation->isTrue();
5. if (fUseBatteryExtendedInformation && kIOReturnSuccess != fProvider->validateBatteryBIX())
6. fUseBatteryExtendedInformation = false;
- 7.
8. Schnapp

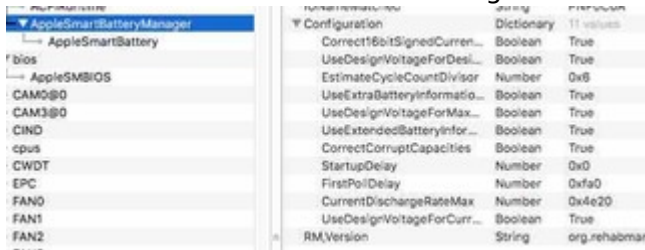
Der Wert hängt von einer anderen Variablen ab und von validateBatteryBIX. Nur anhand des Namens, kann man vermuten, dass in der Methode überprüft wird ob _BIX vernünftige Werte liefert. In unserem Falle wird _BIX das nicht tun, es existiert ja nicht. Der Batterietreiber wird in unserem Falle _BIF verwenden, da validateBatteryBIX false liefern wird und deshalb fUseBatteryExtendedInformation ebenfalls false ist. Das fehlende _BIX is somit nicht unser Problem.

Spannung

IORegistry in der linken und die ACPI Spec in der rechten, überprüfe ich ob die Werte in _BST und _BIF Sinn machen. Tun sie. Die Design Voltage ist zwar als unbekannt markiert, aber das ist laut Spezifikation erlaubt.

Dem aufmerksamen Betrachter mag oben der Eintrag Configuration im AppleSmatBatteryManager aufgefallen sein. Ich bin versucht "ein bisschen spät" zu sagen,

aber mir ist er auch nicht früher aufgefallen.



UseDesignVoltageFor... springt sogleich ins Auge. Also noch mal in den Programmcode geschaut und festgestellt, dass das Kext für die Laufzeitberechnung auf die Design Voltage zurückgreift. Es merkt aber nicht, dass der Design Voltage Wert auf unbekannt steht und schon wird mit -1 multipliziert statt mit 7 irgendwas Volt, es sei denn man lässt das Kext die DesignVoltage ignorieren und stattdessen die aktuelle Batterie Spannung verwenden.. Also gut in der Info.plist unter Configuration die beiden Uses... auf NO gesetzt. Neustart und siehe da: 100%. Dabei fühle ich mich inzwischen als sei meine fast Batterie leer.

Dann kommt die Hintergrundbeleuchtung eben morgen dran.

PS.

Man könnte statt die beiden Uses... zu ändern, natürlich auch in der _BIF Methode einen Wert für die Design Spannung vorgeben. 0x1E00, würde sich anbieten, denn das ist die in _BST angegebene momentane Batteriespannung.

Man könnte auch AppleSmatBatteryManager so ändern, dass es eine validateDesignVoltageMethode gäbe, mit deren Hilfe dieUses... automatisch auf false gesetzt werden würden, falls die DesignVoltage nicht definiert ist.

Beitrag von „DSM2“ vom 7. Februar 2018, 09:21

Da scheint das Lenovo ja in gewisser Hinsicht echt eine Zicke zu sein, dennoch schön zu sehen wie du das Ding "einfach" dazu zwingst. 🙌👍

Beitrag von „KayKun“ vom 7. Februar 2018, 09:53

[@griven](#) und [@al6042](#) sagt noch mal das mein HP Klingonen Laptop ne zicke ist !!!!

[@Brumbaer](#) Nice Job und immer wieder ein spaß zu lesen

Beitrag von „umax1980“ vom 7. Februar 2018, 10:13

Ich bin immer wieder faziniert über die extrem leicht zu lesende Schreibweise trotz der echt kompliziert anmutenden Thematik.

Bei dem Foto auf der Startseite fiel mir auf, dein Schreibtisch könnte meiner sein ...

Beitrag von „burzlbaum“ vom 7. Februar 2018, 10:45

Du hast echt einen tollen Schreibstil! Macht beim Lesen echt Spaß, auch wenn ich bei vielen Bereichen (mangels Fachkenntnis) kurz aussteigen muss.

Bin in sehr gespannt was am Ende deiner Odyssee dann alles funktionieren wird und drücke die Daumen!

Beitrag von „grt“ vom 7. Februar 2018, 17:59

[Zitat von burzlbaum](#)

was am Ende deiner Odyssee dann alles funktionieren wird

da würde ich auf 100% wetten, so wie ich den [@Brumbaer](#) kenne

Beitrag von „Brumbaer“ vom 8. Februar 2018, 16:10

Illuminati

Hintergrundbeleuchtung, ist noch so eine Sache mit der ich mich für gewöhnlich nicht rumschlagen muss. Nach meinen bisherigen Erfahrungen mit dem Miix befürchte ich, dass das eher in die Richtung "Ins Dunkle zu treiben" als in die Richtung "hoch im Licht" geht.

Und wieder beginnt es mit "das geht ganz einfach". Eine SSDT, um ein PNLF Device zu erzeugen, ein Kext in den Other Ordner und gut ist. Meine Reaktion auf solch optimistische Worte ist ein wissendes, aber gequältes Lächeln.

Ich bin bei der Recherche über zwei Kexte gestolpert eins arbeitet über ACPI und eins über den Intelchipsatz, beide benötigen ein PNLF Device und beide gelten als überholt. Es gibt allerdings eine Patch Anleitung von rehabman.

Ich habe die SSDT installiert und zwei Kexte ausprobiert, eins crashte, das Intel basierte funktionierte überraschenderweise.

Dann erinnerte ich mich, dass es in Clover ein AddPNLF Häkchen gibt, also SSDT raus und Häkchen an, geht immer noch.

Dann hab ich das Kext auch noch rausgeschmissen und es geht immer noch.

Dann, und dann, dann war ich verblüfft.

Hintergrundbeleuchtung durch Clover Häkchen. Weil ich gerade solch eine Erleuchtung hatte, gleich noch die Tastenkombination für die Tastaturbeleuchtung ausprobiert. Die Tastaturbeleuchtung ist scheinbar eine reine Tastaturfunktion und geht deshalb ohne Zutun.

Direkt im Anschluss stieß ich einen lauten Schmerzensschrei auf, weil ich mir auf die Kinnlade getreten bin, die hat ja auch nichts auf dem Boden verloren. Frau und Kinder kommen angerannt, vom Schmerzensschrei alarmiert. Aber ich umarme sie nur und wir tanzen ausgelassen einen Ringelrein. Noch schnell die Nachbarn auf ein Glas Schampus eingeladen und die Feuerwerksreste von Sylvester verschossen und den Tag im Kalender ganz dick angestrichen. Alles um das Ereignis gebührend zu feiern. Und euch Zweiflern und Kleingläubigen sei gesagt, es gibt auch Dinge bei Installation von macos auf dem Miix, die ganz leicht von der Hand gehen. Na ja, da es um den Miix geht, sollte ich vielleicht erst mal die Finger nachzählen.

Rückblende

Nach so einem positiven Erlebnis ist es Zeit einmal innezuhalten und an Vergangenes zu denken. Dazu gehört das _LID Problem. Wir erinnern uns, beim Lesen des Klappenstatus klappte der Mac zusammen und nichts ging mehr. Nun hatten wir gerade die DSDT gepatched,

weil macos beim Lesen vom 16 Bit Werten aus dem EmbeddedControl Speicherbereich Schwierigkeiten hatte. Vielleicht, aber auch nur vielleicht, ist das Geklapper ja auch so was. Also schauen wir uns die `_LID` Methode einmal an.

Code

```
1. Method (_LID, 0, NotSerialized) // _LID: Lid Status
2. {
3. If (LEqual (ECRD (RefOf (LSTE)), One))
4. {
5. Return (Zero)
6. }
7. Else
8. {
9. Return (One)
10. }
11. }
```

Alles anzeigen

Die Methode heißt `_LID` und hat 0 Parameter, so weit, so geöhft. Die Methode selbst ist eher übersichtlich.

`ECRD` macht etwas mit `LSTE` und wenn das Ergebnis 1 ist, wird 0 zurückgegeben ansonsten 1.

Die zwei Rückgabebefehle mit Konstanten und das `else`, sollten keine Problem machen.

Das lässt uns nur die `If`-Abfrage. `ECRD` ist kein Standard AML(ACPI Machine Language) Befehl, muss also eine Methode sein. `ECRD`, klingt wie "EmbeddedControllerRead" oder "Elf Clowns Rasen Davon".

`RefOf` ist ein AML Befehl und entspricht in etwa dem Adresse-Operator in C. `RefOf(LSTE)` ist dann sowas wie ein Zeiger auf `LSTE`. `RefOf` wird verwendet, wenn man ein Objekt an eine Methode übergeben und sicher gehen will, dass der Wert des Objektes erst in der Methode ermittelt wird. Das ist wichtig, wenn sich der Wert zwischen Aufruf und Abfrage in der Methode, z.B. durch Optimierungen oder asynchrone Ereignisse, ändern könnte. `RefOf` erzeugt einen fatalen Fehler, wenn es das Objekt nicht gibt. Ein kurzes Suchen in der `DSDT` findet `LSTE`.

`LEqual` überprüft ob seine beiden Parameter gleich sind.

Wenn es hier irgendwo knallt, dann muss es in der `ECRD` Methode passieren, oder die Jungs haben doch noch einen übergebliebenen Böller gefunden. Schauen wir nach was `ECRD` macht.

Code

1. Method (ECRD, 1, Serialized)
2. {
3. Store (DerefOf (Arg0), Local0)
4. Return (Local0)
5. }

Weniger ist mehr

Das ist wirklich nicht viel. *DerefOf* behandelt den Parameter als Zeiger auf ein Objekt und bestimmt dessen Inhalt. Mit *Store* wird dieser in einer lokalen Variablen gespeichert und diese zurückgegeben. Somit liest *ECRD* den Inhalt des Objektes auf das sein Parameter zeigt und gibt ihn zurück. Eine Methode um den aktuellen Wert einer Variablen zu bestimmen. Bei einer normalen Variablen würde man das nicht machen, also handelt es sich vermutlich um ein Hardware Register oder etwas Ähnliches, dass sich jederzeit außerhalb der Programm-Kontrolle ändern könnte.

DerefOf klappt zusammen, wenn es das Objekt auf das sein Argument zeigt nicht gibt. Aber *LSTE* gibt es.

Code

1. Schnipp
- 2.
- 3.
4. OperationRegion (ECF2, EmbeddedControl, Zero, 0xFF)
5. Field (ECF2, ByteAcc, Lock, Preserve)
6. {
7. XXX0, 8,
8. XXX1, 8,
9. XXX2, 8,
10. Offset (0x14),
11. VCMD, 8,
12. VDAT, 8,
13. VSTA, 8,
14. Offset (0x20),
15. RCMD, 8,
16. RCST, 8,
17. Offset (0x60),
18. TSR1, 8,

19. TSR2, 8,
20. TSR3, 8,
21. TSI, 4,
22. HYST, 4,
23. TSHT, 8,
24. TSLT, 8,
25. TSSR, 8,
26. CHGR, 16,
27. Offset (0x6A),
28. Offset (0x6B),
29. Offset (0x6C),
30. Offset (0x6D),
31. Offset (0x6E),
32. TSRT, 8,
33. Offset (0x72),
34. CHGT, 8,
35. NPST, 8,
36. PCVL, 8,
37. Offset (0x7F),
38. LSTE, 1,
39. Offset (0x80),
- 40.
- 41.
42. Schnapp

Alles anzeigen

Alte Bekannte

LSTE ist ein Feld in einem *Field*-Befehl der Operationregion *ECF2*. Die kennen wir inzwischen so gut, dass ich sie zum Sektumtrunk hätte einladen können. Wichtig daran ist, dass *LSTE* demzufolge im Speicherbereich *EmbeddedControl* liegt und der bei 16 Bit Zugriffen Schwierigkeiten macht.

Auf die Länge kommt es an

Aber *LSTE* ist nur 1 Bit lang und nicht 16. Als Arbeitshypothese behaupte ich, dass Apple nicht nur mit Feldern, die länger als 8 Bit sind Probleme hat, sondern mit allen Feldern, die nicht 8 Bit lang sind.

Wenn dem so wäre, müssten wir den Zugriff auf *LSTE* so ändern, dass ein ganzes Byte, statt nur eines Bits, gelesen wird und die anderen 7 Bits verworfen werden.

Wir könnten nun eine Methode schreiben, die ein Byte liest, so wie wir eine Methode

geschrieben haben, die ein Wort (16 Bit) byteweise liest (*RDWD*) - oder wir lesen statt eines Bytes einfach ein Wort und werfen 15 statt 7 Bit weg. Da letzteres weniger Schreibarbeit ist und wir schon wissen, dass die Methode zum 16 Bit Lesen funktioniert, wählen wir diese.

Wir brauchen für *RDWD* die Byteadresse an der das erste zu lesende Byte steht. Direkt vor *LSTE* steht *Offset(0x7F)* und schon haben wir die die Byteadresse von *LSTE*.

RDWD(0x7F) gibt einen 16 Bit Wert zurück indem *LSTE* enthalten ist. *LSTE* hat eine Länge von einem Bit und seine Deklaration folgt direkt hinter einem *Offset* Befehl. *LSTE* ist also das erste Bit (Bit 0) in dem Byte an der Adresse 0x7F. Wir lesen 16 Bit der Adresse 0x7F, aber da *LSTE* das erste Bit an dieser Adresse ist, ist es auch das erste Bit im Wort.

Maskenball

Um den Wert einer bestimmten Bitkombination aus einem Wert zu ermitteln, verwendet man ein Verfahren namens Maskierung. Die Maske entspricht der Binärzahl die man erhält, wenn man die interessanten Bits auf 1 setzt. Wir haben 16 Bit und uns interessiert Bit 0. Wie bei anderen Zahlen stehen links die höherwertigen Stellen, Bit 0 ist also ganz rechts 0b0000000000000001. Wandelt man das in eine Dezimalzahl erhält man 1.

Beim Maskieren führt man eine *Und*-Verknüpfung zwischen Maske und Wert aus. Alle Bits, die in der Maske eine 0 haben, werden zu Nullen und die anderen Bits nehmen den selben Zustand wie im Ausgangswert an.

And (RDWD(0x7F), 1) liest die 16Bit aus dem Controller und löscht alle Bits außer dem einen (die Maske für Bit 0 ist 1), das uns interessiert. Das Ergebnis des *And*-Befehls ist 0 oder 1, ganz so als ob wir nur das eine Bit gelesen hätten.

Wir ändern unsere `_LID` Methode in

Code

```
1. Method (_LID, 0, NotSerialized) // _LID: Lid Status
2. {
3. If (LEqual (And (RDWD(0x7F), 1), One))
4. {
5. Return (Zero)
6. }
7. Else
8. {
9. Return (One)
10. }
11. }
```

Alles anzeigen

DSDT.aml geändert, gespeichert und neugestartet. Und ... vergessen den Patch in der *Config.plist* abzuschalten ... Auf ein Neues, Neustart, im Clover Menü, Optionen, *ACPI*, *Patch* abgeschaltet und Tadaah, bootet, kein Crash, kein Crash, kein Crash.

Klappe schließen und der Bildschirm wird dunkel. Kein Sekt mehr da, so ein Pech, Böller sind auch alle, also nur ruhige gelassene Heiterkeit statt ausgelassener Feierstimmung.

Ach ja, da wir wissen, dass es funktioniert, können wir den eben erwähnten Patch in Clover abschalten oder entfernen.

Irgendwas ist immer

Und irgendwo zwischen Verblüffung und heiterer Gelassenheit stelle ich fest, dass die NVME SSD Schwierigkeiten macht und manchmal hängen Maus und Tastatur beim Systemstart. System auf USB Medium hat scheinbar keine Probleme, wenn denn Maus und Tastatur gehen. Habe ich wohl was an der DSDT verbastelt. Also alte und neue DSDT verglichen.

Alles zu seiner Zeit

DSDTs und *SSDTs* liegen in zwei Formaten vor. Als *aml* oder als *dsl* Datei. *Aml* Dateien enthalten ausführbaren, maschinenlesbaren Code. *Dsl* Dateien enthalten nicht ausführbaren, menschenlesbaren Code. Wenn man eine Datei im *dsl* Format im patched Ordner ablegt, passiert nichts, denn sie enthält keinen ausführbaren Code.

Wenn man eine *aml* Datei in *MaciAsl* öffnet, erzeugt es intern eine *dsl* Datei, zeigt sie an und lässt sie uns editieren und beim Speichern erzeugt sie aus der internen *dsl* wieder eine *aml* Datei. Das erweckt den Eindruck, dass eine *aml* Datei und eine *dsl* Datei für Menschen gleich leicht zu lesen seien. Dem ist nicht so.

Wenn man zwei *DSDTs* oder *SSDTs* vergleicht, sollte man das mit den *dsl* Versionen tun, denn was Unterschiede in den *aml* Versionen bedeuten, erkennt nur der Geübte. Deshalb vor dem Vergleich die *aml* Dateien mit *MaciAsl* in *dsl* Dateien umwandeln. *Dsl* Dateien sind normale Textdateien man kann somit die Text-Vergleichs-App seiner Wahl verwenden.

Wenn man keine hat, kann man *FileMerge* verwenden. *FileMerge* findet man in *Xcode* im *XCode Menü* unter *Open Developer Tool*. *FileMerge* dient dem zusammenfügen von Dateien, zeigt aber auch die Unterschiede an.

Natürlich habe ich erwartet, die von Hand gemachten Änderungen zu sehen. Aber da gibt es auch auch unerwartete Änderungen.



Theorie ...

Irgendwo im *BIOS* ist eine *DSDT* gespeichert und diese wird dann an *Clover* und von dort an *macos* übergeben. Ändert man was an den *BIOS* Einstellungen, wird eine neue *DSDT* erstellt. Speichert man eine eigene *DSDT* im patched Ordner, ersetzt *Clover* die *BIOS-DSDT* durch die eigene. Das ist kein Problem, solange die Änderung einer *BIOS* Einstellung nicht eine Änderung in der *DSDT* bewirkt. Denn diese Änderung wird nicht automatisch in die eigene *DSDT* übernommen. Dann passen verwendete *DSDT* und [BIOS Einstellungen](#) nicht mehr zueinander.

... und Praxis

Das ist das Konzept. Die Realisation kann im Detail anders aussehen. Z.B. spricht nichts dagegen, dass das *BIOS* die *DSDT* bei jedem Start komplett neu erstellt. Was bleibt ist das Problem, dass wenn sich etwas in der vom *BIOS* bereitgestellten *DSDT* ändert, die Änderung nicht in der gepatchten *DSDT* wider gespiegelt wird.

Hier haben wir solch einen Fall. die Adresse der Operationregion *GNVS* unterscheidet sich. D.h. sie kann sich ändern und dann stimmen die Adressen aus der *BIOS DSDT*, die sagt wo *GNVS* tatsächlich gespeichert ist, und der gepatchten *DSDT*, die sagt wo *GNVS* gespeichert wurde als die Kopie der *DSDT* gemacht wurde, nicht mehr überein und die Zugriffe auf *GNVS* liefern Müll.

Die Schreiber von *Clover* wissen um das Problem und deshalb gibt es die Option *FixRegions* in den *ACPI Fixes*.

"Haaaahh, Haaaahh ! Nimm das du Schurke !", denke ich mir und setze die Option. Neustart, Crash. So leicht lassen sich Schurken wohl nicht aufhalten.

Plan B

Das Problem lässt sich lösen indem man die *DSDT* nicht patched, sondern alle Änderungen in einer neuen *SSDT* zusammenfasst und dort von der *BIOS DSDT* aufrufen lässt.

Aber um in der *DSDT* etwas in der *SSDT* aufrufen zu können, muss ich die *DSDT* patchen, "Katze, Schwanz, und so".

Die Lösung sind Clover DSDT Patches. Diese werden auf die aktuelle *DSDT* angewandt. Ohne *DSDT.aml* in *patched* Ordner wird das die zum BIOS passende sein. Mit *Clover* nennt man dann die Methoden um, so dass sie nicht mehr in der *DSDT* gefunden werden. Dann woanders gesucht und hoffentlich in unserer *SSDT* gefunden werden.

Klappe, die dritte

Nehmen wir als Beispiel die *_LID* Routine.

Wir packen sie in eine *SSDT*.

Das Gerüst einer *SSDT* sieht wie folgt aus:

Code

1. DefinitionBlock ("", "SSDT", 2, "VonMir", "Brumbaer", 0x00000001)
2. {
3. }

MEINER und *Brumbaer* sind zwei Strings zur Identifikation des Autors und der Tabelle.

Der Autor darf sechs Zeichen lang sein, der Tabellename acht.

Da *Brumbaer* 8 Buchstaben lang ist, habe ich ihn als Tabellennamen verwendet. Kein guter Stil. Aber solange man nicht zwei *SSDTs* mit dem Tabellennamen *Brumbaer* anlegt, geht das.

Die 1 am Ende ist die Versionsnummer, die 2 davor schaltet in den 64Bit Mode.

Kopiert man die *_LID* Routine in die neue *SSDT* rein, erhält man:

Code

1. DefinitionBlock ("", "SSDT", 2, "VonMir", "Brumbaer", 0x00000001)
2. {
3. Method (_LID, 0, NotSerialized) // _LID: Lid Status
4. {
5. If (LEqual (And (RDWD (0x7F), One), One))
6. {
7. Return (Zero)
8. }
9. Else
10. {
11. Return (One)
12. }
13. }

14. }

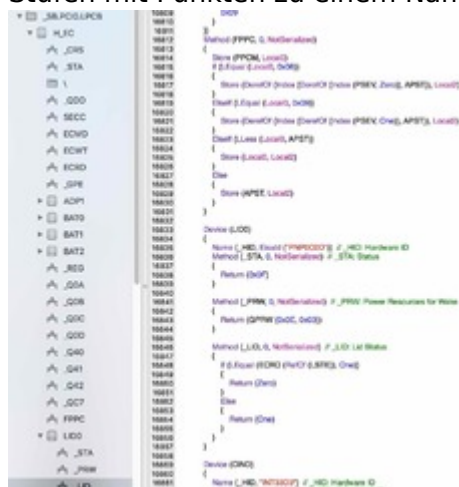
Alles anzeigen

Alles ein Frage des Kontextes

Wenn man von Herrn Meier redet, weiß keiner welcher Herr Meier gemeint ist. Es sei denn man erwähnt Herrn Meier in einem bestimmten Kontext, wie in einer Geschichte über das Büro. Oder man man gibt mehr Details: Herr Meier im Büro, Herr Meier, wohnhaft in

So geht es uns mit unserer `_LID` Methode. Die `_LID` Routine gehört "in" den momentanen Kontext auf Englisch Scope. Wenn wir wollen, dass die `_LID` Routine zu unserem `LIDO` Gerät gehört, müssen wir entweder den Kontext dorthin verschieben oder `_LID` mit Wohnort, PLZ, Straße und Hausnummer angeben. Wir entschlossen uns den Kontext, den Scope, zu verschieben. Der Befehl dazu heißt `Scope`, Zufälle gibt's. Der `Scope`-Befehl hat einen Parameter, der angibt wohin der `Scope` verlagert werden soll. `Scope(LIDO)` klingt plausibel, ist aber nur korrekt, wenn der `Scope` vorher schon auf dem Objekt stand zudem `LIDO` gehört. Um das wasserdicht zu machen, geben wir als Parameter für den `Scope`-Befehl, die komplette Adresse inklusive Plz. und Hausnummer an. Allerdings müssen wir sie erst finden.

In der Seitenleiste zeigt `MaciAsl` einen Objekt-Baum. Wählt man ein Objekt (`_LID`) an, wird es im Objekt-Baum markiert. Wenn man nun im Objekt-Baum nach oben wandert und die einzelnen Stufen mit Punkten zu einem Namen verknüpft bekommt man:



`_SB.PCI0.LPCB.H_EC.LIDO._LID`

`_LID` ist die Methode und wir suchen das Objekt zu dem sie gehört bzw. die neue gehören soll. Also ist das Objekt `_SB.PCI0.LPCB.H_EC.LIDO`

Code

1. DefinitionBlock ("", "SSDT", 2, "VonMir", "Brumbaer", 0x00000001)
2. {

```

3.
4.
5. Scope (_SB.PCI0.LPCB.H_EC.LID0)
6. {
7. Method (_LID, 0, NotSerialized) // _LID: Lid Status
8. {
9. If (LEqual (And (RDWD (0x7F), One), One))
10. {
11. Return (Zero)
12. }
13. Else
14. {
15. Return (One)
16. }
17. }
18. }
19.
20.
21. }

```

Alles anzeigen

Sofortiges Kompilieren führt zu sofortigen Fehlermeldungen.



`_SB.PCI0.LPCB.H_EC.LID0` ist kein Standardobjekt laut ACPI Spezifikation, und deshalb kennt der Compiler es nicht.

Für die Methode `RDWD` gilt das Gleiche.

Code

```

1. DefinitionBlock ("", "SSDT", 2, "MEINER", "Brumbaer", 0x00000001)
2. {
3. External (_SB_.PCI0.LPCB.H_EC.LID0, DeviceObj)
4. External (_SB_.PCI0.LPCB.H_EC.RDWD, MethodObj)
5.
6.

```

```

7. Scope (_SB.PCI0.LPCB.H_EC.LID0)
8. {
9. Method (_LID, 0, NotSerialized) // _LID: Lid Status
10. {
11. If (LEqual (And (RDWD (0x7F), One), One))
12. {
13. Return (Zero)
14. }
15. Else
16. {
17. Return (One)
18. }
19. }
20. }
21. }

```

Alles anzeigen

Die erste *External*-Anweisung sagt dem Compiler, dass das Objekt `_SB_.PCI0.LPCB.H_EC.LID0` ein Gerät (Device) ist und dass es irgendwo anders definiert wurde.

Die zweite *External*-Anweisung sagt dem Compiler, dass das Objekt `_SB_.PCI0.LPCB.H_EC.RDWD` eine Methode ist und dass sie irgendwo anders definiert wurde. Der Pfad vor dem `RDWD` (`_SB_.PCI0.LPCB.H_EC`) ist komplett in dem Pfad des Scope-Befehls (`_SB.PCI0.LPCB.H_EC.LID0`) enthalten, deshalb genügt es `RDWD` in der `_LID` Routine zu schreiben.

Wäre das nicht der Fall hätte man

Code

1. Schnipp
2. If (LEqual (And (_SB_.PCI0.LPCB.H_EC.RDWD (0x7F), One), One))
3. Schnapp

schreiben müssen. Über Scope ließe sich mehr sagen, aber ich verweise interessierte an die ACPI Spezifikation.

Jetzt müssen wir noch dafür sorgen, dass unsere neue `_LID` Methode auch verwendet wird.

Raus aus die Kartoffeln, rin in die Kartoffeln

In der *DSDT* ist ja schon eine `_LID` Methode und die muss weg, damit unsere neuere, bessere

und schönere Methode verwendet wird. Deshalb verwende wir einen Clover DSDT Patch, um die `_LID` Routine in der `DSDT` umzubenennen, denn `Clover DSDT Patches` können nicht löschen. Den Patch haben wir im dritten Artikel gemacht und in diesem Artikel weiter oben, wieder rausgeworfen. Jetzt holen wir ihn wieder rein.

Für mich sieht ein `_LID` aus, wie das andere

Die `_LID` Routine gibt es nur einmal. Was mache ich denn, wenn es eine Routine mehrmals gibt, wie `_DSM` oder `_CRS` ?

Es gibt in der `Clover DSDT Patches` Tabelle eine Spalte `TgtBridge`. Dort kann man eine Kennung eines Gerätes eintragen auf das der Patch begrenzt werden soll. Auch wieder als Hexzahlen. `TgtBridge` ähnelt dem `Scope`-Befehl. Dummerweise ist die Funktionalität eingeschränkt, da das nur funktioniert, wenn das zu ersetzende Etwas innerhalb des `Device`-Befehls der `TgtBridge` steht. Leider werden `Scope`-Befehle ignoriert.

Bei den Battery Patches bietet es sich an die gesamten `_BST` und `_BIF` Methoden auszulagern. Wichtig ist immer, das man nichts auslagert, was sich wechselnde `OperationRegion`-Befehle enthält.

Und siehe da, alles funktioniert noch und die NVME und Maus/Tastatur-Probleme beim Start sind weg.

Eine Folge hab ich noch.

Beitrag von „Dr.Stein“ vom 8. Februar 2018, 16:22

Ich freu mich sehr auf die Taschenbuchausgabe für 9,95€ 😊

Beitrag von „McRudolfo“ vom 8. Februar 2018, 17:24

Ich würde auch die gebundene Ausgabe nehmen - für gewöhnlich erscheint diese auch früher 😊

Beitrag von „jboeren“ vom 8. Februar 2018, 17:27

Ich möchte nur die Ausgabe mit Autogramm!

Beitrag von „gerox“ vom 8. Februar 2018, 17:35

eine Freude zu lesen

kurz und knapp : Werner würde sagen " Gooooil--"....

Beitrag von „grt“ vom 9. Februar 2018, 12:05

[Zitat von Dr.Stein](#)

Ich freue mich sehr auf die Taschenbuchausgabe

meine Buchbindewerkstatt steht zur Verfügung... 😊

Beitrag von „Bernd.H“ vom 9. Februar 2018, 12:53

Ich würde sagen Zeitverschwendung (oder du hast ja nichts anderes weiter vor, da kann man das ja tun) das Laptop wird nie wie ein anderes vergleichbares Hack-Book laufen, bei der fast 80%igen inkompatiblen Hardware.

Beitrag von „umax1980“ vom 9. Februar 2018, 13:09

ich würde sagen: warten wir das doch mal ab

Beitrag von „Bernd.H“ vom 9. Februar 2018, 15:35

ja freilich, kann ja jeder selber tun was er möchte.. war doch nur meine persönliche Meinung dazu.

Beitrag von „apfelnico“ vom 9. Februar 2018, 15:54

Wäre dafür, die letzten vier Einträge inkl. meinem hier zu löschen. Ich werde richtig sauer, wenn ein so schöner Artikel mit so xxxxxxxxxx und respektlosen Gesülze, was nichts zum Thema beiträgt, verunstaltet wird. Hoffentlich folgen noch weitere Parts von Brumbaer. Richtig großes Kino!

Beitrag von „Bernd.H“ vom 9. Februar 2018, 16:13

[apfelnico](#).. meinst ALLE Post sollten zensiert werden, wir sind nicht in der Ex-DDR, hier ist freie Meinungsäußerung möglich !

Beitrag von „apfelnico“ vom 9. Februar 2018, 16:23

Hatte ich befürchtet, dass der Mist weitergeht. Schau dir noch mal deinen "Beitrag" zum Thema an. Ich denke das "respektlos" recht passend ist. Mit Zensur tue ich mich schwer, gerade weil ich die DDR auch miterlebt habe. Freie Meinungsäußerung ist ein hohes Gut, ist aber auch keine Pflichtübung.

Beitrag von „Bernd.H“ vom 9. Februar 2018, 16:32

[apfelnico](#).. man sollte doch nicht immer bei jeder Kleinigkeit die dir persönlich nicht passt streiten.. immer die Ruhe bewahren.

Es war doch nur eine Meinung, das man so viel Energie UND Zeit reinbringt in ein Laptop, was aller Wahrscheinlichkeit viel inkompat. Hardware hat,

eben aus rein technischer Interesse, versucht es zum laufen zu bekommen. Ich hätte die Zeit, 1. für das lange schreiben hier und 2. für die vielen Tests nicht.

Und bitte lies das doch dann nicht, was dich NICHT interessiert und gut ist und alle leben in Ruhe.

Und abwarten wie [@umax1980](#) schrieb.. ich verfolge den Thread auch nur aus technischer Neugier mit und tippe darauf, das Brumbaer irgend wann das Sache aufgibt.

Beitrag von „Brumbaer“ vom 9. Februar 2018, 19:20

Wer bin denn du ?

Über den Mac ist als Informationsquelle wirklich hilfreich. Schon auf der Titelseite nennt es einem z.B. den Prozessortyp. Da Apple keine 8250U verbaut könnte man mit einem *Unbekannt* rechnen, aber weit gefehlt, "*Intel Core i5*" sei der Name des Prozessors. So sei es, das langt mir.

Wenn's scheitert macht

Ein Durchklicken der Optionen liefert keine bunten, aber dafür die erwarteten Ergebnisse. Das heißt unter *PCI* erfahre ich, dass der Miix keine PCI Geräte hat. Hätt' man mir das 'mal vorhergesagt.

Mit *PropertyInjector* kann man der Anzeige auf die Sprünge helfen. WLAN und iGPU habe ich eingetragen, damit man was sieht. Alles nur Kosmetik.

Rest in peace

Da war noch was - mit dem Ruhezustand. Habe ich nur vergessen, weil ich die ganze Zeit über mit Bildschirmfreigabe gearbeitet habe. Nach dem Wake gehen die internen USB Geräte nicht. Die üblichen Verdächtigen hatte ich schon durch, also mal das Internet fragen. Die Ausbeute war eher mager und die Lösung, falls es denn eine gab, war ein Verändern von Optionen der CPU PLL Spannung. Probiere ich gleich an dem Tag aus, an dem Lenovo ein BIOS zur Verfügung stellt, das den Namen verdient und Optionen zur CPU PLL Spannung anbietet. Ich befürchte die Chancen dafür stehen nicht gut, nicht einmal wenn man eine Wiedergeburt in Betracht zieht.

Die Geräte sind noch in der Registry zu sehen. Warum geht dann nichts ?

Kahlschlag, Holz weg, Holzweg

Ich tippe auf ein Problem, das darauf zurückzuführen ist, dass USB Gerät und USB Controller aus der selben Spannungsquelle versorgt werden. Ich hatte mit einem extern mit Spannung versorgtem USB Gerät am externen Port getestet und das funktionierte. Nun mit einem vom Rechner mit Spannung versorgtem USB Gerät am externen Port getestet und funktioniert - auch. "Mist". Nicht im Sinne eines landwirtschaftlichen Abfallproduktes, sondern als Ausdruck eines Gefühles, das sich bei Gelegenheiten bei denen sich etwas nicht so entwickelt wie man es erwartet, einstellt.

Also gut, zweiter Test: BT. BT scheint zu funktionieren, obwohl es intern verbaut ist, wie Touchpad und Tastatur.

Leichte Schläge auf den Hinterkopf fördern das Kommunikationsvermögen

Es scheint also was persönliches zu sein zwischen Touchpad, Tastatur und mir, allerdings wollen T&T nicht damit rausrücken was sie wollen. Genau genommen wollen sie mit gar nichts rausrücken. Wenn wir aus tausend Folgen NCIS etwas gelernt haben, dann dass in solchen Situationen leichte Schläge auf den Hinterkopf helfen.

In Computersprache nennt man leichte Schläge auf den Hinterkopf Reset, schwere Schläge übrigens auch, da gilt es das richtige Maß zu finden. IORegistryEditor sagt uns, dass T&T Instanzen von IOUSBDevice (neumacosisch IOUSBHostDevice) sind. IOUSB(Host)Device hat eine Methode, die die Kommunikation zurücksetzt, dafür gedacht neu anzufangen, wenn die Platte hängt. Das ist nicht brutal, tut nicht weh und sorgt dafür, dass das Device noch eine Chance bekommt es richtig zu machen. Es sei an dieser Stelle erwähnt, dass ein IOUSBDevice durch Resets weder körperlich noch seelisch Schäden erleidet und dass beim Schreiben dieses Artikels keinem Device, egal von welcher Art, körperlicher oder seelischer Schaden zugefügt wurde.

Im Zweifelsfall ein kext

Um ein USBDevice in den Genuss eines Resets kommen zu lassen, braucht man eine App oder ein Kext. Ein Kext ist praktischer, weil man es nicht starten muss und es unauffällig seinen Dienst tut. Zuerst ein Testkext namens Sleeper geschrieben, um rauszubekommen, wie ein Kext Sleep und Wake Events erkennt.

Dann die Erkennung der USB Devices hinzugefügt und beim Wake für diese ein Reset eingefügt. Die Möglichkeit den Reset zu verzögern vorgesehen und das war's auch schon.

Wort mit vier Buchstaben, dass oben schon mal verwendet wurde. Ich wollte das doch in ein eigenes Kext packen. jetzt heiß das Ding Sleeper und macht ganz was anderes, Wecker wäre viel treffender.

Wort mit vier Buchstaben, dass oben noch nicht verwendet wurde und Gleichgültigkeit ausdrückt.

Verzögerung auf 10 Sekunden eingestellt, Other Ordner, Neustart, Ruhezustand, Wake, T&T gehen nicht, ein paar Sekunden später, T&T gehen. Ein Fremdwort mit vier Buchstaben und Bezug zu niedrigen Temperaturen.

Zwei Seiten einer Medaille

Da wir gerade von Ruhezustand sprechen. Die Optionen für den Ruhezustand im *Energie sparen* Kontrollfeld hängen vom gewählten Platform Plugin ab. Für gewöhnlich lässt man das das System wählen. Bei einer nicht unterstützten CPU wird das das *ACPI_SMC_PlatformPlugin* sein. Bei einer unterstützten Intel CPU wird es das *X86PlatformPlugin* sein.

Da wir eine X86 CPU haben macht es Sinn das *X86PlatformPlugin* zu verwenden. Wird es aber nicht, weil macos die CPU nicht kennt. MacOS entscheidet anhand der *plugin-type* Eigenschaft seiner ersten CPU welches Platform Plugin verwendet wird.

Ist der *plugin-type* 1, so ist es das *X86PlatformPlugin*. Wir können den *plugin-type* über das *PropertyInjector* Kext oder die *config.plist* setzen. Im *Clover Configurator* unter *ACPI*, ein Häkchen zum Einschalten und ein Feld für den Wert. Ein Wort mit vier Buchstaben, "fortbewegen".

Zeig mal was du kannst

Zeit für ein paar Benchmarks.

LuxBall GPU: 1764, LuxBall CPU: 1415.

Geekbench Single: 4242, Multi: 12625.

Cinebench OpenGL: 29FPS, CPU: 500.

Würgemale



IntelPowerGadget zeigt uns bei allen Tests, dass bessere Ergebnisse durch Throttling verhindert werden. Man sieht in dem Screenshot sehr schön wo das Throttling einsetzt. Man kann im BI verschiedene Stromsparmodi einstellen. Die oben angegebenen Werte sind für "höchste Performance".

Träume, Wünsche, Hoffnungen

Wenn ich einen Hack aufsetzte, habe ich bestimmte Erwartungen. Sachen die laufen müssen und einige Dinge, die ich zu schätzen weiß aber nicht unbedingt brauche.

Einige Sachen sind offensichtlich und nur in der Liste damit ich sie abhaken kann 😊

Andere Sachen wie Shiki, tauchen bei mir nicht auf oder finde ich nicht essentiell wie 32 State PM, da ich sie nicht brauche, obwohl sie für andere rechnernotwendig sind.

Und wieder andere Dinge tauchen nicht in der Liste auf, weil ich sie vergessen habe.

Nur weil etwas nicht wichtig ist, bedeutet es nicht, dass ich nicht u.U. viel Zeit investiere, um es zum Laufen zu bekommen; aus Interesse halt.

Wichtig für alle Rechner

- Prozessor ✓
- Prozessor rudimentäre 2 State PM notfalls über C-States ✓
- Grafikkbeschleunigung ✓
- Speicher ✓
- Sound ✓
- USB ✓
- NVMe ✓
- SATA - hat Miix nicht

- LAN - hat Miix nicht
- iTunes/Appstore ✓

Gerne gesehen bei allen Rechnern.

- Prozessor PM z.B. XCPM ✓ ersetzt die rudimentäre PM
- WiFi ✓
- BT ✓
- Micro ✓
- AirDrop ✓
- Continuity ✓
- Messages ✓
- Ruhezustand ✓

Bei Laptops kommen zu den wichtigen Dingen folgende hinzu bzw. werden von gern gesehen zu wichtig:

- Prozessor PM ✓
- WiFi ✓
- BT ✓
- Eingebaute Tastatur ✓
- Eingebautes Touchpad ~ Keine Gesten außer Scrollen
- Batterieanzeige ✓
- Ruhezustand ✓

Zusätzlich bei Laptops gern gesehen:

- Einstellbare Hintergrundbeleuchtung ✓
- Tastaturbeleuchtung ✓
- Klappenerkennung ✓

Speziell für den Miix kämen zusätzlich in die erste Kategorie:

- LTE
- Touchscreen

Und in die zweite:

- Kameras
- Fingerabdruckscanner,
- Accelerometer

- Was auch immer

Es ist doch gar nicht schlimm ...

Wenn man sich von den ganzen Problembeschreibungen löst und realisiert, dass es auch für jedes beschriebene Problem eine Lösung gab und sich die Häkchen in der Liste anschaut, dann sind das eine ganze Menge. Genau genommen haben alle für einen Laptop gewählten Features ein Häkchen. Das Touchpad funktioniert zu meiner Zufriedenheit. Es unterstützt halt keine Gesten außer Scrollen und wird über das Mauskontrollfeld gesteuert. Für mich nicht schlimm, aber auch nicht ideal.

Selbst Handoff funktioniert, übrigens ohne 2FA.

Das HackBook scheint fertig, ob es versteckte Fehler gibt oder ich ein Feature vergessen habe, wird die Zeit zeigen.

... kann es aber noch werden.

Das Dumme ist, dass ich den Miix nicht als Laptop gekauft habe, sondern als 2 in 1. Präziser als MacBook und iPad Ersatz. Und dazu fehlen mir Touchscreen und LTE.

Ich weiß, dass es sich um einen über I2C angebotenen Touchscreen von Wacom und ein über USB angebotenes LTE Modul handelt.

Spannenderweise funktioniert die Netzwerkkomponente des LTE Moduls mit macos eigenen Treibern. Ich muss das Modem vermutlich nur noch dazu bringen anzurufen.

Es gibt ein VoodooI2C Kext mit Support Kexten, die wohl auch Touchscreens unterstützen. Ob und wie einfach die funktionieren werde ich bald wissen.

Man wird sehen ob ich aus einem HackBook einen 2 in 1 machen kann.

Die Zeit verfliegt, wenn man sich amüsiert

Der bisherige Zeitverbrauch für das Projekt ist schwer abzuschätzen. Einige Problemlösungen waren schneller erarbeitet als beschrieben und bei anderen war es genau andersherum. Das langwierigste Einzelproblem war das _LID Problem zu erkennen. Im Nachhinein ist es ganz einfach und hätte in 20 Minuten entdeckt sein können, aber es hat letztendlich über eine Woche gedauert.

Die Probleme habe ich nicht so geradlinig nacheinander abgehandelt wie beschrieben, sie und ihre Lösungen haben sich zum Teil gegenseitig beeinflusst. Aber die Artikel sind wirr genug ohne die Probleme, die die Beschreibung von Problemen, die in anderen Problemen Als ich die Artikel geschrieben habe, waren die meisten Taten schon vollbracht. Man kann also nicht von den Erscheinungsdaten auf die Komplexität der einzelnen Probleme schließen. Ich habe die Gelegenheit genutzt beim Beschreiben die einzelnen Schritte nach zu vollziehen, um zu sehen, dass ich auch nichts ausgelassen habe.

Ruhezustand

Jetzt begeben sich erst mal in selbigen und lasse das Projekt eine Woche ruhen und widme mich dann dem Touchscreen, schau mal, dann seh mal scho.

Vielen Dank fürs Lesen.

Ein Wort des Abschieds mit vier Buchstaben.

Beitrag von „griven“ vom 9. Februar 2018, 21:26

Und wieder einmal großartig geschrieben und nebenbei den Beweis erbracht das mit dem notwendigen Hintergrundwissen und einer gewissen Bereitschaft die Dinge auch anzugehen sich selbst ein scheinbar aussichtsloser Kandidat in ein HackBook verwandeln lässt das in vielen Disziplinen angeblich sehr kompatiblen Geräten in nichts nachsteht bzw. diese sogar übertrifft. Ich bin mir ziemlich sicher das Du den Touchscreen auch noch dazu überreden kannst seinen Zweck zu erfüllen 😁

Was ich aber besonders bewundere ist Deine Bereitschaft Dich solchen Problemen zu stellen und Deinen Elan solche Projekte auch umzusetzen. Egal ob es ein MIXX ist oder eine Tardis oder die Bärenbrüder alle Deine Projekte sind durchdacht und liebevoll ausgeführt. Da wo es an fertigen Lösungen hapert werden eben welche geschaffen und genau das ist der Spirit den es braucht im kleinen wie im großen. Ohne Leute die einen solchen Willen und eine solche Einstellung haben würde es keine Hackintoshes geben denn auch da haben die Leute gesagt macOS auf einem PC, nee das geht nicht das ist nicht kompatibel, das ist Zeitverschwendung.

Beitrag von „DSM2“ vom 10. Februar 2018, 08:18

[Zitat von Bernd.H](#)

ich würde sagen Zeitverschwendung (oder du hast ja nichts anderes weiter vor, da kann man das ja tun) das Laptop wird nie wie ein anderes vergleichbares Hack-Book laufen, bei der fast 80% igen inkompatiblen Hardware.

Wo ist das Ding den bitte zu 80% inkompatibel?

Generell ist es sehr wohl möglich aus nem Notebook ein anständiges Hackbook zu machen. Das beweist der [@Brumbaer](#) hier super, auch wenn das Miix zickiger ist als mein Dell.

Im Fall von meinem Dell 7773 zum Beispiel

Funktioniert alles bis auf Kamera und Kartenleser. Liegt aber auch nicht in meinem Interesse diese in Betrieb zu nehmen, da eh nicht benötigt.

Ich bin mir sicher Brumbaer kriegt das schon geschaukelt.

Bei solchen Kommentaren von dir wie "Zeitverschwendung" krieg ich die Krise... Weil du es nicht hibekommen hättest, kriegt es niemand hin oder was ?

EDIT: Gut das zuklappen habe ich noch vergessen, sprich das Display bleibt an, wenn ich es zuklappe. Aktuell ebenfalls nebensächlich, kann ich in den Angriff nehmen sobald sich der Umzugs Stress gelegt hat...

Beitrag von „andreas_55“ vom 10. Februar 2018, 08:59

[Zitat von Brumbaer](#)

Vielen Dank fürs Lesen.

Vielen Dank fürs Schreiben.

Beitrag von „Bernd.H“ vom 10. Februar 2018, 14:29

[@DSM2](#) ach ja das und dies und Jenes „interessiert mich nicht“ alles klar, dann brauch es nicht zu Funktionieren... ja war gut zum lachen diese Einstellung. Also auch viele Komponenten inkompatibel... ja aber brauch man dann ja nicht. Oh oh was sind das für Aussagen.

und den Satz „weil du es nicht hinbekommst kriegen andere es auch nicht hin“ hättest du dir ersparen können, denn so ein exotisches Ding Miix hätte ich bei Kenntnis über die Hardwarekomponenten NIE für ein Hackbook ausgewählt.

Beitrag von „derHackfan“ vom 10. Februar 2018, 14:37

[@Brumbaer](#) vielen lieben Dank für deine Mühe uns teilhaben zu lassen an deinem Projekt.

Beitrag von „grt“ vom 10. Februar 2018, 14:57

ich freu mich auch jedesmal, wenn [@Brumbaer](#) eine fortsetzung schreibt.
bin besonders gespannt auf die touchfunktion - und auf den stammtisch wenn wir uns das geräterchen live ankieken können.

Beitrag von „al6042“ vom 10. Februar 2018, 15:24

Leute....
kommt wieder runter... der Thread ist zu gut, um hier unnötigen Ärger zu fabrizieren...

Beitrag von „al6042“ vom 10. Februar 2018, 16:11

Beide Beiträge wurden soeben gelöscht...

Macht einer von Euch beiden noch mal eine Text zu diesem Vorfall hinterher, ist er für die nächsten Wochen gebannt...

Beitrag von „Brumbaer“ vom 9. April 2018, 17:16

Man hat mich gefragt, wie es weiterging, wie nicht anders zu erwarten mühsam:

Zwei für den Preis von einem

Der eine, der Laptop, läuft. Zeit für den zweiten. Das wichtigste Tablet-Feature ist der Touchscreen.

Da stelle me uns e mal ganz dumm

Ein Touchscreen dient vornehmlich als Ersatz für eine Maus abzüglich der ständigen Bedrohung des Käsebestandes.

Damit der Touchscreen als Eingabegerät funktioniert braucht man einen Treiber.

Der Touchscreen schickt irgendwelche Signale über eine Schnittstelle an den Treiber und der fügt große Ohren, einen langen Schwanz und ein Gespür für Käse hinzu und schon denkt das System es hätte eine Maus.

Man kann allerdings auch Touchscreens bauen, die ein Mauskostüm tragen, so dass die Katze (aka Maustreiber) denkt, da sei eine Maus.

Welcher Fall liegt hier wohl vor ?

Kein Anschluss unter dieser Nummer

Ich hatte beim Kauf des Miix gehofft, der Touchscreen sei über USB angeschlossen. Spätestens seit dem Erstellen des USB Kextes ist klar, dass dem nicht so ist. Es gibt zwar zwei USB Eingabegeräte, aber dabei handelt es sich um Tastatur und Maus.

Wo bist du, mein Schatzzzzzz ?

Dummerweise

Dummerweise ist der einfachste Weg Windows zu starten und im Gerätemanager nach dem Gerät zu suchen.

Dummerweise habe ich Windows überschrieben, als ich verzweifelt gehofft habe meine NVMe

sei inkompatibel und alles würde mit der Original NVMe sofort und ohne Probleme funktionieren.

Da ich bekanntermaßen fast alles habe, habe ich auch ein Windows 10 - auf einer 2,5" SSD, fertig installiert und einsatzbereit.

Dummerweise hat der Miix keinen SATA Anschluss.

Da ich bekanntermaßen fast alles habe, habe ich auch einen USB SATA Adapter.

Dummerweise komme ich nicht darum herum, Windows starten zu müssen.

FensterIn tut man nicht nur in Bayern

Also setzen wir uns bequem hin, sammeln uns, atmen gleichmäßig und ruhig, suchen unsere Mitte, und nehmen unseren ganzen Mut zusammen. Die Engländer kennen das Konzept "Mut aus der Flasche", wenn es denn nicht anders geht, so sei in diesem Falle auch dies gestattet.

So gewappnet starten wir Windows. Die Augen zu Schlitzen verengt, konzentrieren wir uns auf das Such-Text-Feld und geben Geräte Manager ein und starten diesen.

Die ersten beginnen nun zu vermuten, dass sie nicht vom Teufel geholt werden, nur weil sie Windows gestartet haben. Den anderen geben wir noch 5 Minuten, sich an den Gedanken zu gewöhnen und von denen, die der Teufel doch geholt hat, verabschieden wir uns in stiller Trauer.

Wir schauen uns die List der Geräte an und finden Mäuse und andere Zeigegeräte. Es müsste schon mit dem Teufel zugehen, wenn wir unseren Touchscreen dort nicht finden. Kein beruhigender Gedanke für die die immer noch in Erwartung von ihm geholt zu werden, hin und wieder ängstlich über ihre Schulter schauen.

Da gibt es zwei Geräte, beide haben das selbe Icon und beide heißen HID-konforme Maus. Welches wir uns zuerst anschauen ist egal, denn sollten wir finden, was wir suchen, wird es immer beim zweiten Gerät sein.

Das erste Gerät ist über USB angeschlossen also nichts für uns, hol's der Teufel

Das andere Gerät ist ein I2C HID-Gerät und unter Details/Hardware-IDs kommt unter anderem WCOM513B zum Vorschein.

"WCOM hallo, WCOM", und da steht auch "VID_056A". "VID_056A hallo, VID_056A".

Es ist ein Gerät der Firma Wacom (Vendor Id 056A) und über I2C angeschlossen und ein HID Gerät.

Wacom ist der wohl bekannteste Hersteller von Touchscreens und Digitizern, also ist das vermutlich unsere Touchscreen.

Es handelt sich um ein HID Gerät, d.h. die Befehle, die an die Hardware geschickt werden, folgen dem HID Standard.

Alles toll soweit, aber bekanntermaßen steckt der Teufel im Detail. In diesem Falle I2C. Wie bekommt man den zum Laufen ? Problem für später. Bevor der Teufel sich aus dem Detail befreit und uns doch noch holt, beenden wir erst mal Windows.

Puhh, überstanden. Windows ist doch gar nicht so schlimm. Mag aber auch am Weihwasser liegen, mit dem ich den Luftbefeuchter befüllt habe.

I2C, ein alter Bekannter, aus Controller Tagen. Ein einfaches Zwei-Leiter-Bussystem für Kommunikation innerhalb eines Gerätes. Ursprünglich für das Kommunizieren von ICs untereinander in Fernsehern von Philips entwickelt. Unsere Atmel/Arduino Freunde kennen I2C aus Lizenzgründen als TWI.

Am Atmel-Controller ist das Programmieren des I2C Controllers kein Problem, aber wie sieht es unter macos aus ?

Google ist prima, Google ist ne Wucht, mit Google macht das suchen Spass

Ich persönlich finde "Suchen" überbewertet, "Finden" ist mir wichtiger.

Und siehe da Google findet was und zwar VoodooI2C, einen I2C Treiber. Und er kommt nicht allein, sondern bringt Kexte zur Unterstützung von HID Geräten mit.

Das ist ja schon fast magisch, wie das passt.

Fauler Zauber

VoodooI2C.kext und VoodooI2CHID.kext in den System Ordner und ... nix.

Na ja, fast nix. Im IORegistryExplorer kann man sehen, dass zumindest der VoodooI2C Treiber installiert wird. Bei den Geräten I2C0 und I2C1.

Laut Google gibt es ein paar Threads in amerikanischen Foren zu dem Thema. Schnelles Überfliegen - ggf. Nachtflugverbot beachten - führt zu folgender Zusammenfassung: RTFM.

Also das FM aufgerufen.

VoodooI2C is for "advanced tinkerers"

"Fortgeschrittene Fummler", manche Dinge übersetzt man besser nicht.

Es geht los mit einer Checkliste:

Prozessortyp, Haswell oder neuer - Check

Maschine kommt mit Win7 oder neuer - Check

Unterstützter I2C Controller - Check

I2C Gerät - Check

OS X 10.10 oder neuer - Check

Clover Bootloader - Check

Alle 5 Bedingungen erfüllt ? - Äh sind aber 6 Bedingungen.

Und gleich geht's los mit der Fummelei, DSDT Fummelei.

Die 5 Bedingungen, die 6 sind, sind symptomatisch für den Stand der Anleitung.

Es gab immer mal Änderungen, aber die Anleitung hat nicht alle mitgemacht. Im Endeffekt kein echtes Problem, aber etwas das seinen Weg in die Flüche findet, wenn gar nichts funktioniert.

Und die Krönung ist, dass die Probleme gar nichts damit zu tun hatten, sondern mit einer fehlerhaften Datei. Nachdem ich die Software erneut heruntergeladen hatte, funktionierte es laut Anleitung unter Berücksichtigung der Änderungen.

Die Lösung besteht laut Anleitung aus einer Handvoll Patches (240+ Zeilen Patchcode) in der DSDT. Klassischer Rundum-Schlag-Patch.

Es geht auch anders

Die Miix Lösung berührt die DSDT nicht, sondern kommt mit einem Eintrag in der config.plist (`_OSI` in `XOSI` umbenennen) und einem Zusatz in der SSDT, die ich für das Batteriemangement angelegt hatte, aus:

Code

```
1. Scope (\)
2. {
3. Name (_SB.PCI0.I2C1.TPL1.SDM1, Zero)
4.
5.
6. Method (XOSI, 1, NotSerialized)
7. {
8. If (LEqual (Arg0, "Windows 2015"))
9. {
10. Return (LOr (_OSI ("Darwin"), _OSI (Arg0)))
11. }
12. Else
13. {
14. Return (_OSI (Arg0))
15. }
```

16. }

17. }

Alles anzeigen

Der `_OSI` Befehl fragt beim Betriebssystem an, ob ein bestimmter Text ihm was sagt. Das wird meist zur Abfrage des Betriebssystems benutzt. Es gibt keinen Befehl der fragt "Welches Betriebssystem bist du denn?". Stattdessen wird gefragt "Kennst du den Text `Windows2015`?". Wenn die Antwort ja lautet handelt es sich um Windows 10. Der Mechanismus ist für die Abfrage des Betriebssystems etwas umständlich, aber äußerst flexibel und kann auch noch für andere Dinge verwendet werden.

Der Betriebssystempatch des Originals ersetzt `_OSI` ("`Windows2015`") durch `LOR` (`_OSI` ("`Darwin`"), `_OSI` ("`Windows2015`"). Das bedeutet jedesmal, wenn gefragt wird "Bist du Windows 10", wird ein "oder macos" angehängt. Dadurch werden macos und Windows 10 von der DSDT gleichbehandelt. Wenn man ein altes "OS X" benutzt kann es sein, dass man es besser nicht mit Windows 10, sondern einem älteren Windows verbindet.

Die gewählte Variante macht das Gleiche, braucht aber mehr Worte. Ich habe sie trotzdem gewählt, denn ich kann diese Variante ohne DSDT.aml verwenden. Die `XOSI` Methode wird in einer SSDT gespeichert bzw. einer existierenden SSDT hinzugefügt und durch einen ACPI Patch in der `config.plist`, werden die `_OSI` Aufrufe auf `XOSI` umgeleitet.

Min to the Max

Die wahre Patchorgie von 241 Zeilen Patchcode wird in der Miix Variante zu

Code

1. `Name (_SB.PCI0.I2C1.TPL1.SDM1, Zero)`

Die Zeile wird in der SSDT mit dem `XOSI` Gedöns - oder irgendeiner anderen oder einer eigenen SSDT - gespeichert.

Warum genügt die `SDM1` Zeile ?

Wenn man sich die Ziele der Original Patcherei ansieht (in der Dokumentation oder einer gepatchten DSDT), erkennt man, dass das alles schon in der DSDT stand, es wurde nur nicht verwendet.

DSDTs sind häufig so ausgelegt, dass sie verschiedene Konfigurationen enthalten und anhand

von Betriebssystem und verschiedenen Objekten/Flags entscheiden welche Konfiguration verwendet werden soll.

Hat man ein Windows 10 und hat das SDM1 Objekt den Wert 0, so baut die DSDT genau die Hardwarebeschreibung für I2C zusammen, die wir brauchen.

Dass macos als Windows 10 erkannt wird, haben wir mit der XOSI Methode sichergestellt und SDM1 ?

SDM1 ist ein Feld in GNVS:

Code

1. OperationRegion (GNVS, SystemMemory, 0x7FF69018, 0x074E)
2. Field (GNVS, AnyAcc, Lock, Preserve)
3. {
4. OSYS, 16,
5. SMIF, 8,
- 6.
- 7.
8. Schnipp
- 9.
- 10.
11. EGPV, 8,
12. TBDT, 32,
13. ATLB, 32,
14. SDM0, 8,
15. SDM1, 8,
16. SDM2, 8,
17. SDM3, 8,
- 18.
19. Schnapp

Alles anzeigen

GNVS steht vermutlich für Global Non Volatile Storage und bietet Infos für was weiß ich nicht alles.

Man könnte nun SDM1 auf 0 setzen und unser I2C Bus sollte funktionieren.

Aber wie beim blinden Rumgepatche weiß man nicht, wo das überall Auswirkungen zeigt. Deshalb wollen wir SDM1 in GNVS nicht verändern.

Weltweite Kette oder Kneipe an der Ecke

Die Überprüfungen von SDM1 innerhalb vom TPL1 folgen alle dem selben Muster:

Code

1. If (LEqual (SDM1, Zero))

Unsere Abfrage befindet sich in einer Methode des Gerätes TPL1, das Teil von I2C1, das Teil von PCI0, das Teil von _SB, das Teil von / ist.

Wenn der Interpreter nun auf SDM1 trifft, versucht es dessen Wert zu bestimmen.

Dazu sucht es in der näheren Umgebung (sprich in TPL1) nach einem SDM1, findet aber keins. Also schaut der Interpreter in I2C1 und dann in PCI0 und in _SB und findet nirgendwo eins.

In / (die oberste Ebene der Objekthierarchie) findet der Interpreter letztendlich das SDM1 in GNV5 und verwendet dessen Wert. SDM1 in GNV5 ist in der obersten Hierarchiestufe und kann immer und überall gefunden werden. Wegen des überall, heißt es globales Objekt.

Wenn wir nun dafür sorgen, dass ein anderes SDM1, das nur von TPL1 gesehen gefunden werden kann, schon früher gefunden wird, dann können wir dessen Wert ändern, ohne dass Methoden außerhalb von TPL1 beeinflusst werden.

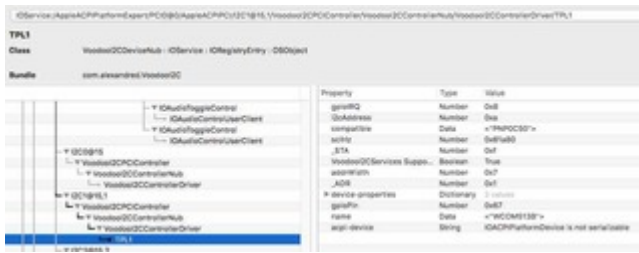
Also erzeugen wir ein SDM1 in TPL1. Das kann ohne volle Angabe der Adresse von anderen Methoden und Objekten nicht gefunden werden. Da das Objekt bisher nicht existierte, gibt es auch keine Zugriffe mit voller Adresse auf auf SDM1 in TPL1, kann also auch keine Probleme außerhalb erzeugen.

Mit Name (SDM1, Zero) erzeugen wir das Objekt mit dem Namen SDM1 und dem Wert 0. Da unser SDM1 zu TPL1 gehören soll muss die komplette Adresse angegeben werden:

Code

1. Name (_SB.PCI0.I2C1.TPL1.SDM1, Zero)

Da unser SDM1 in TPL1 liegt wird es ohne Angabe der Adresse nur bei Aufrufen aus TPL1 - aus der näheren Umgebung - heraus gefunden. Man nennt so etwas ein lokales Objekt. Im Gegensatz zu einem globalen Objekt, dass von überall aus gefunden werden kann.



Man könnte die Lösung auch mit einem Scope Befehl und dem Name Befehl realisieren. Wie das dann aussehen würde mag sich jeder selbst überlegen.

Und schon erscheint der VoodooI2C Treiber mit seinen Untertreibern und dem TPL1 Gerät im IORegistryExplorer.

Lass uns miteinander reden

Jetzt haben wir einen Treiber der die I2C Schnittstelle unterstützt und es macos erlaubt mit dem Touchscreen zu reden.

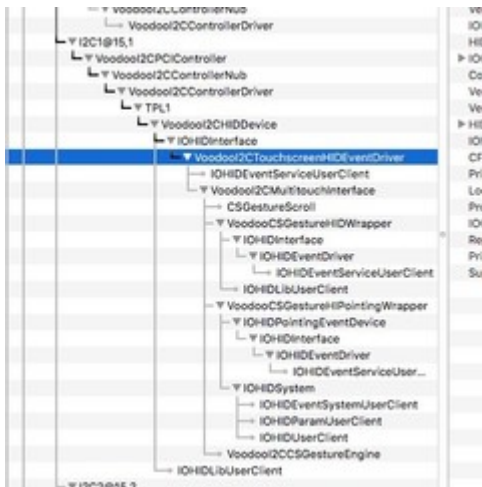
Jetzt müssen sie sich nur noch verstehen.

Windows hat uns verraten, dass der Touchscreen HID kompatibel ist.

Macos hat HID Treiber, es fehlt nur etwas, was den macos HID Treiber und unseren I2C Treiber verbindet und dabei die Daten passend aufbereitet.

Das VoodooI2C Paket bietet mehrere solche Treiber an. VoodooI2CHID.kext klingt vielversprechend - HID ohne Schnörkel.

Und siehe da, VoodooI2CHID hängt sich an das Gerät und stellt Verbindung mit dem macos' HID Treiber her.



VoodooI2CHID unterstützt mit Hilfe des macos HID Treibers:

Klicken durch Zeigen auf eine Stelle

Drag durch Klicken auf eine Stelle und Bewegen des Fingers

RechstKlick durch Klicken und Gedrückt-Halten.

Scrollen mit zwei Fingern.

Doppelklick funktioniert etwas unzuverlässig, da es schwer ist zweimal kurz hintereinander die selbe Stelle zu treffen. Das hat weniger mit einem ruhigen Händchen, als der Größe und Weichheit der Fingerkuppe zu tun.

Damit kann ich erst einmal leben. Es zeigt mir dass man den Touchscreen ansprechen kann, dass er eine "bekannte" Schnittstelle zum System hat. Und somit kann man mit entsprechendem Aufwand die Funktionalität hinzufügen die man braucht bzw. zu brauchen glaubt.

Aber vielleicht muss man nicht einmal mehr Aufwand reinstecken, denn es gibt noch weitere Module und Optionen für Gesten und sogar für Stifte - die Frage ist ob die auch funktionieren. Die grundsätzliche Funktion des Touchscreens ist hergestellt, erst einmal weiter zum nächsten Grundsatzproblem.

Beitrag von „cooper“ vom 9. April 2018, 17:32

Hurra !..... meine Abendlektüre ist daHurra... Hurra.. Hurra !

Danke . 👍

Beitrag von „grt“ vom 9. April 2018, 17:52

[@Brumbaer](#) hast du mal betrieb mit wacomstiften probiert?

Beitrag von „Brumbaer“ vom 9. April 2018, 18:05

[@grt](#)

Der Miix wird mit Stift ausgeliefert. Dieser funktioniert mit dem oben erwähnten Treiber nur als dünner Finger.

Es gibt Voodoo-Treiber mit Pen Support, aber ein kurzer Test hat keine Pen spezifischen Funktionen gebracht. Woran das liegt kann ich noch nicht sagen.

Dass die generelle Touch Unterstützung funktioniert langt mir im Moment, denn die Funktionen, die ich brauche kann ich darauf aufbauend notfalls selbst realisieren. Das ist dann nur noch mühsam und aufwändig, aber nicht unmöglich.

Jetzt kommt erst mal das LTE-Modem dran. Denn das ist das letzte KO Kriterium. Wenn ich das nicht zum Laufen bekomme ist, werde ich auch keine Zeit mehr in die Vervollkommnung des Restes investieren.

Beitrag von „burzlbaum“ vom 10. April 2018, 13:24

Toll geschrieben und einfach beeindruckend was mit dem nötigen technischen Fachwissens so alles machbar ist!

bin allerdings zwischendurch bei „Es geht auch anders,, inhaltlich ausgestiegen xD

Beitrag von „Arkturus“ vom 10. April 2018, 21:12

technisch unversiert bin ich einfach nur fasziniert 👍

[@Brumbaer](#) in zwei Zeilen soviel Humor einzubauen ist m.E. unübertroffen, Erich Kästner, Karl Valentin, Loriot, Brumbaer, nichts mehr ...

Beitrag von „umax1980“ vom 10. April 2018, 21:18

[@Brumbaer](#) hat sich dein Schreibstil schon zu Schulzeiten so entwickelt? Da hatten die Deutschlehrer ja richtig Spaß....

Beitrag von „derHackfan“ vom 10. April 2018, 21:24

Es war ein mal ... da hat er in seiner Bärenhöhle von Winter zu Winter seine Schreibkunst weiterentwickelt, dann eines Tages im Frühling ist es aus ihm heraus gesprudelt und die Menschheit war so weit es anzunehmen, ... und wenn er nicht gestorben ist dann entwickelt er noch Heute. 😄

Beitrag von „Brumbaer“ vom 11. April 2018, 01:39

[Zitat von umax1980](#)

[@Brumbaer](#) hat sich dein Schreibstil schon zu Schulzeiten so entwickelt? Da hatten die

Deutschlehrer ja richtig Spaß....

Nein, in der Schule waren meine Schwerpunkte Mathe und Physik.

Beitrag von „Brumbaer“ vom 12. April 2018, 11:13

Nach Hause telefonieren

Wollen wir das nicht alle ?

Na ja, vielleicht, aber ohne leuchtenden Finger, es sei denn wir wären süd-koreanischen Haustierklone.

Eine ständige Internetverbindung all-überall ist was tolles. Ich habe mich an das LTE Modem im iPad so gewöhnt, dass kein Gerät ohne LTE Modem das iPad ersetzen kann.

Das eingebaute LTE Modem war einer der Gründe für die Wahl des Miix.

Alles ganz einfach

Wo habe ich das schon gehört ?

Egal - IORegistryExplorer zeigt unter den USB Geräten das Modem an.

Typ *L831-EAU* Hersteller *Fibocom* - und jetzt kommt's - *IORegistryExplorer* zeigt an, dass "Freude schöner Götter Funken", Treiber geladen werden.



Ein Gefühl, wie bei km 42 eines Marathonlaufes. Nicht, dass ich jemals einen Marathon gelaufen wäre oder es in Betracht ziehen würde oder es in Betracht ziehen würdew es in Betracht zu ziehen, aber so muss es sich anfühlen, körperlich und emotional erschöpft, aber das Ziel in Sicht, ein letztes Mobilisieren der Kräfte und dann ins Ziel.

Habe ich auch verdient nach dem Stress mit dem Miix. Einen Tee gekocht, eine Box von nem iPad gesucht, den SIM-Karten-Schlitten-Auswurf-Stecker - Stecker und Auswurf passt nicht ? Doch, doch. Man muss was reinstecken, damit was raus kommt, wie beim Kaugummi-

Automaten. Liebe Kinder, Kaugummi-Automaten sind ... googelt's halt.

Ist denn schon Ostern

Wo war ich ? Ach ja das Dingens gepackt und die SIM Karte aus dem iPad genommen. Jetzt muss man nur noch den Steckplatz am Miix finden. Wenn das Teil mit ausgeklapptem Ständer vor einem steht, sieht man ihn erst mal nicht. Ich sehe eh besser mit den Händen, also die Seiten abgetastet - nichts. Also gut Gerät von der Tastatur abgezogen, Ständer rangeklappt. Miix nach links, rechts, oben und unten gedreht und nichts. Wenn das so weiter geht, muss ich ins Handbuch schauen. "Nicht mit mir", denke ich, "ich der Mann, der ein Bit am Geruch erkennt, den noch kein Byte gebissen hat, der morgens mit dem Datenbus zur Arbeit fährt und abends mit dem Adressbus nach Hause kommt, der soll in ein Handbuch schauen ? Haaahhh, niemals". Die Schnellstartanleitung tut's auch.

Es ist auch ganz einfach, man klappt den Ständer auf und dann sieht man dahinter zwar nicht den SIM Karten Slot, aber immerhin das Logo des Slots. Also mal getastet und gleich gefunden. Ich sach ja, ein Fachmann wie ich, findet so einen Slot ja sofort. Der braucht kein Handbuch. Die Karte geht etwas schwer rein. Vermutlich weil sie aus einer größeren Karte gestanzt wurde und etwas dicker ist, als die neuen Nanos. Selbstoptimierung macht auch vor SIM Karten nicht halt. Wahrscheinlich ernähren die sich auch nur von veganem Strom.

Fasching ist draußen

Auf den Straßen herrscht buntes Treiben, aber hier tut sich nichts. Also Windows gestartet und das Modem getestet und was soll ich sagen ? Es läuft - zumindest das Modem. Zurück zu MacOS. Offensichtlich wird ein Ethernettreiber geladen, es kommen nur keine Daten, vermutlich besteht keine LTE-Verbindung.

Mühsam

Die Suche nach Informationen über das Modem führt Widersprüchliches zu Tage. In der Produktliste taucht es erst gar nicht auf und die Handbücher, die Google findet, beschreiben unterschiedliche Versionen.

Wahrscheinlich heißt es Handbuch, weil man trotz des Buches, alles von Hand rauskriegen muss.

Klassisch

Klassischerweise hat ein Modem zwei Komponenten. Die eine für die reine Datenübertragung und die andere für den Verbindungsaufbau. Die für Datenübertragung tut beim Mac so als wäre sie ein *Ethernetcontroller* und die andere als sei sie ein serielles *AT Modem*. Den *AT Befehlssatz* gibt es seid Ewigkeiten. Inzwischen wurde er natürlich, um ein paar Features

z.B. für den Verbindungsaufbau über Handy Netz erweitert. Bei den Erweiterungen kocht zum Teil jeder Hersteller seinen eigenen Buchstabensalat.

Bei der Suche nach Informationen über das *L831-EAU* bin ich auch über ein Handbuch zu dessen Interpretation des AT Chipsatzes gestolpert.

Desinteresse

Schaut man sich obigen Auszug aus dem *IORegistryExplorer* an, sieht man

- *HS03*. Das ist der USB Port - interessiert uns nicht.
- *L831-EAU@14300000* ist das USB-Gerät. Gaaanz toll - aber interessiert uns nicht. Dann sehen wir zwei Treiber.
- *AppleUSBHostCompositeDevice* ist ein Treiber für USB Geräte, die mehrere Funktionen in sich vereinen. Welche Funktion ein USB Gerät hat geben seine *Klasse*, *Unterklasse* und *Protokoll* an. Das Tripel sagt zum Beispiel *Drucker*. Und für diese Funktion gibt es dann ein *Interface*. Manchmal hat ein Gerät aber mehr als eine Funktion, z.B. *Drucker und Scanner*. Dann hat man ein *Composite Device* und Geräte Klasse, Unterklasse und Protokoll verweisen bezüglich der Funktion auf die Interfaces. Und es gibt ein Interface für die Druckfunktion und eins für die Scanfunktion, jeweils mit passender Klasse, Unterklasse und Protokoll. *AppleUSBHostCompositeDevice* verwaltet solche Mehrfach-Funktionsgeräte und ist im laufenden Betrieb eher unsichtbar. Kurzum interessiert hier nicht.
- *AppleUSBHostLegacyClient* stellt die Kompatibilität mit älteren Programmen her. Interessiert hier auch nicht.

Desdesinteresse

Auf der selben Ebene gibt es noch *Data (OFF)@1* und Fibocom *L831-EAU (NCM)@0*. Dies sind die zwei Interfaces mit ihren Treibern. Das Data Interface lädt einen Ethernettreiber. Das sieht gut aus, aber das andere, zeigt keinen seriellen Treiber, also klappt die normale Methode mit AT Befehlen über eine serielle Schnittstelle nicht.

Bestandsaufnahme

Da stellt sich uns die Frage, "was gibt's zu Mittag ?". Nein, also doch schon, aber eigentlich, wollte ich auf "Hat das Modem eine serielle Schnittstelle und wenn nein was hat es stattdessen ?" hinaus.

Statt nur das Ergebnis zu präsentieren, zeige ich im Folgenden wie das Ergebnis zu Stande kam. Wenn man schon weiß wie es geht, ist es in einer Minute erledigt. Dank der Erklärung dauert es deutlich länger. Die, die USB Deskriptoren und der Aufbau von USB Geräten nicht interessiert oder ihn schon kennen, können gerne zu *Was sagt uns das ?* springen.

USB Stocherer

Wieder mal eine schöne Übersetzung. *USB Prober* ist eine App und Teil des *Apple USBFamily Log Paketes*. Es soll Hardware Entwickler beim Anbinden von USB Geräten und der USB Treiber Entwicklung unterstützen. Dass die neueste Version zum System 10.9.4 gehört, sagt viel über den aktuellen Stand der Unterstützung von Hardware Entwicklern. Ob die Unterstützung für große Firmen besser ist, weiß ich nicht.

Ok, genug gejammert, der Miix liefert genug Gründe zum Jammern, da brauch ich nicht noch Apple für.

Ach ja, das Paket findet man auf der Apple Webseite unter <https://developer.apple.com/download/more/>. Im Suchfeld USB eingeben. Ob man es mit jeder Art von Entwickleraccount runter laden kann oder ob es noch sonst wo zum Download angeboten wird weiß ich nicht.

Was haben wir denn da ?

Die USB Spezifikation legt fest, dass jedes USB Gerät über *Descriptors* definiert wird. Ein Descriptor beschreibt die Eigenschaften einer Komponente und ihre Unterkomponenten. Die Deskriptoren bilden so eine Hierarchie in Form einer Baumstruktur. Die Struktur der Deskriptoren spiegelt die Struktur der Komponenten eines USB Gerätes wieder. Am Ende des Textes ist eine Grafik, die die Organisation der Deskriptoren nicht als Baum, sondern als Schachtelbild zeigt.

USB Prober liest die Deskriptoren und bereitet sie in von Menschen lesbarer Form auf:

```
High Speed device # 5 (0x1430000) ..... Miscellaneous/Common Class device: "L831-EAU"
  *Port Information: ..... B:001x
  *Number of Endpoints (Includes EPR): ..... 4
  *Total Endpoints for Configuration 1 (current): ..... 4
  *Device Descriptor
    Descriptor Version Number: ..... 0x0200
    Device Class: ..... 239 (Miscellaneous)
    Device Subclass: ..... 2 (Common Class)
    Device Protocol: ..... 1 (Interface Association)
    Device MaxPacketSize: ..... 64
    Device VendorID/ProductID: ..... 0x2CE7/0x0002 (unknown vendor)
    Device Version Number: ..... 0x3729
    Number of Configurations: ..... 1
    Manufacturer String: ..... 1 "FISIXCM"
    Product String: ..... 2 "L831-EAU"
    Serial Number String: ..... 3 "004099020640000"
  *Configuration Descriptor (current config)
  *Device Qualifier Descriptor
  *Other Speed Configuration Descriptor
  *High Speed device # 6 (0x1430000) ..... "USB2.0 Hub"
  *Super Speed device # 7 (0x1430000) ..... Composite device: "USB Storage"
```

In der ersten Zeile erfahren wir, dass es sich um das *fünfte USB Gerät* handelt, über *USB 2.0* (High Speed) kommuniziert und am Port 3 (die 3 in *0x1430000*) hängt, der Klasse *Verschiedenes* angehört und das Gerät *L831-EAU* heißt.

Uns interessiert das Gerät, nicht wo am Computer es angeschlossen ist, deshalb ignorieren wir die Port Informationen.

Das Ende ist nah

Endpoints

sind die Punkte an denen die *Pipes* anknüpfen. *Pipes* sind die Datenübertragungs-Kanäle. Die Daten fließen durch die *Pipes* von einem *Endpoint* zum Computer oder vom Computer zu einem *Endpoint*.

Es geht nicht ohne

Die Geräteverwaltung hat einen eigenen Endpoint, den *Endpoint 0*. Das ist der *Control Endpoint*. Computer und USB kommunizieren nach einem strengen Protokoll. Der Computer sendet einen 8 Byte langen *Request*. Danach können noch Daten folgen, die vom Computer oder dem USB-Gerät stammen können. Es gibt vordefinierte *Requests* für alles Mögliche und man kann auch eigene *Requests* definieren. Zu den vordefinierten *Requests* gehören zum Beispiel das Wählen einer Konfiguration und das Abfragen der Deskriptoren. Ohne diese Fähigkeiten geht nichts, deshalb muss jedes USB Gerät einen Endpoint 0, einen *Control Endpoint* haben.

Aber es geht auch anders

Der *Control Endpoint* wird mit dem Gerät selbst und dessen Verwaltung assoziiert und sein Übertragungsprotokoll ist nicht wirklich effizient.

Deshalb können *Interfaces* (kommt gleich) eigene Endpoints definieren.

Diese Endpoints können nur gelesen oder beschrieben werden, aber im Gegensatz zum Endpoint 0 nicht beides.

Soll ein Interface Daten empfangen und senden können, muss es zwei Endpoints bekommen, einen zum Senden und einen zum Empfangen.

Endpoints werden durchnummeriert. Da die 0 schon vom Control Endpoint belegt ist, beginnen die Endpoints, die beschrieben werden, bei der 0x01 und Endpoints, die gelesen werden, bei 0x81.

Die Datenübertragung erfolgt nicht mit Requests, sondern in einem von drei anderen Modi:

- *Bulk* - Byte folgt auf Byte, Computer kontrolliert, z.B. Daten von oder zu einer Festplatte
- *Interrupt* - Das Gerät schickt Daten unaufgefordert an den Computer. Falls das Gerät dem Computer was sagen muss. Z.B. bei einem Überwachungsgerät oder wenn das Gerät was macht und sich dann meldet wenn es fertig ist. In Wahrheit fragt der Computer regelmäßig nach, ob das Gerät Daten hat, die es loswerden will. Das passiert aber im verborgenen und sieht von außen so aus als wäre da ein Interrupt am Werke.
- *Isochronous* - Daten werden als kontinuierlicher Datenstrom übertragen. Z.B. von einer Videocamera. Wie bei einem Hop-on Bus, kann man jederzeit ein- oder aussteigen.
- Wie gesagt ein USB-Gerät muss über keinen solchen Endpoint verfügen, aber es muss

einen Control-Endpoint besitzen.

Der L831-EAU hat laut USB Prober in der momentan ausgewählten Konfiguration 4 Endpoints inkl. des Endpoints 0. Der Endpoint wird vom Gerät zur Verfügung gestellt. Die anderen 3 Endpoints müssen von einem oder mehreren Interfaces zur Verfügung gestellt werden.

Die Anzahl und Art der Interfaces und damit die Anzahl und Art der Endpoints hängt von der jeweiligen Konfiguration (*Configuration*) ab. Es ist selten, dass ein Gerät mehr als eine Konfiguration unterstützt. USB Prober weist auch nur einen *Configuration Descriptor* aus. D.h. unser Gerät hat auch nur eine Konfiguration.

Die schauen wir uns später an.

Der *Device Descriptor* erzählt uns Allgemeines über unser USB Gerät. Die meisten der Einträge sollten selbst erklärend sein. *Number of Configurations* bestätigt uns, dass es nur eine Konfiguration gibt.

Device Class, Subclass und Protocol verdienen besondere Erwähnung. USB verwendet Klassen, Unterklassen und Protokolle, um Geräte in Gruppen einzuteilen. Alle Geräte einer Gruppe haben bestimmte Eigenschaften. Typische Gruppen sind *Drucker, Audio/Video* oder *Drahtlos*. Unter http://www.usb.org/developers/defined_class/#BaseClassEFh findet man eine Auflistung. In unserem Fall ist die Klasse 239. Das ist wenig hilfreich, steht die 239 doch für *Verschiedenes*. Unterklasse 2 und Protokoll 1 bedeuten "es gibt einen *Interface Association Descriptor*, der mehr über die Geräte Funktion aussagt". Dieser ist Teil der Konfiguration, wir werden ihn deshalb später als Teil des *Configuration Descriptors* wiederfinden.

Das Leben der anderen

Den *Configuration Descriptor* schieben wir noch mal nach hinten. Und so kommen wir zu *Device Qualifier Descriptor* und *Other Speed Configuration Descriptor*. Beide gibt es nur bei *High Speed (USB 2.0)* Geräten. *High Speed* Geräte unterstützen zwei Geschwindigkeiten: *Full-Speed (USB 1.0)* und *High-Speed (USB 2.0)*. In der ersten Zeile stand, dass im Moment High Speed verwendet wird. Nun ist es möglich, dass sich die Deskriptoren im High- und Full-Speed Modus unterscheiden. Diese beiden Deskriptoren beschreiben das Gerät im "anderen", dem gerade nicht verwendeten Modus. Es ist uns egal was im Full-Speed Modus wäre, wir werden ihn nie verwenden, deshalb ignorieren wir die beiden Deskriptoren.

Last, not least

Jetzt können wir uns nicht länger vor dem Configuration Descriptor drücken.

```
*Configuration Descriptor (current config)
  * Length (and contents):                163
  * Number of Interfaces:                 2
  * Configuration Value:                  1
  * Attributes:                           0x00 (self-powered, remote wakeup)
  * MaxPower:                             100 mA
  * Interface Association
    * First Interface                     0
    * Interface Count                      2
    * Function Class                       2 (Communications-Control)
    * Function Subclass                    13
    * Interface Protocol                   0
    * Function String                      4 "Fibocom LE31-EAU"
  * Interface #0 - Communications-Control ..... "Fibocom LE31-EAU (NMO)"
  * Interface #0 - Communications-Control (F1) ..... "Fibocom LE31-EAU (NMO)"
  * Interface #1 - Communications-DATA/Unknown Com Class Model ..... "Data (DF)"
  * Interface #1 - Communications-DATA/Unknown Com Class Model (F1) .. "Data (NMO)"
  * Interface #1 - Communications-DATA/Unknown Com Class Model (F2) .. "Data (NMO)"
```

Der *Configuration Descriptor* trägt selbst wenig zur Erleuchtung bei, aber seine Unterdeskriptoren haben es in sich.

Bei einem USB Gerät stellen die *Interfaces* die Funktionen zur Verfügung. Bei einer Drucker, Scanner, Fax Kombi würde man je ein *Interface* für die Druck, Scan- und Faxfunktion anlegen. Man kann auch Alles irgendwie in einem *Interface* verwurschteln, aber vorgesehen ist eine Aufteilung der Funktionen auf verschiedene *Interfaces*.

Der *Interface Association Descriptor* gibt ergänzende Informationen über die Konfiguration und das Zusammenspiel der *Interfaces*. Er ist nur notwendig, wenn im Device Descriptor auf ihn verwiesen wird.

Das erste Interface (*First Interface*) ist das Interface mit der Nummer 0. Insgesamt (*Interface Count*) gibt es zwei Interfaces. Der eine oder andere mag sich erinnern, dass der Device Descriptor sich nicht auf eine Funktion festnageln lassen wollte und auf den *Interface Association Descriptor* verwiesen hat. Und der sagt jetzt Klasse 2 (Kommunikationsgerät) , Unterklasse 13 (NCM - Network Control Model) und Protokoll 0 (No encapsulated commands / responses.).

Also ein Modem, dass der NCM Unterklasse angehört und keine encapsulated commands/responses unterstützt.

Die fünf, die zwei waren

Anzahl der Interfaces ist laut Configuration Descriptor 2, aber wir sehen 5 Interfaces. Na ja es sind nur 2 Interfaces, Interface 0 und Interface 1, von denen gibt es aber 2 bzw. 3 Varianten. Wozu denn das, warum hat man nicht einfach zusätzliche Konfigurationen angelegt ?

Eine Konfiguration, kann den Charakter des Gerätes komplett ändern und wenn man zwischen zwei Konfigurationen wechselt ist es so als ob man das Gerät abzieht und ein anderes Gerät anschließt. Alles auf Anfang. Schaltet man bei der Druck-, Scan-, Fax-Kombi die Konfiguration um, um z.B. die Drucker Emulation zu wechseln, brechen auch laufende Scans und Faxübertragungen ab und auch Scanner und Fax, werden zurückgesetzt.

Das Umschalten auf ein *Alternatives-Interface*, betrifft nur dieses Interface und lässt den Rest des Gerätes unberührt. Das erlaubt ein relative glattes Umschalten einzelner Funktionen im

laufenden Betrieb.

```
▼ Interface #0 - Communications-Control ..... "Fibocom L831-EAU (NCM)"
Alternate Setting: 0
Number of Endpoints: 1
Interface Class: 2 ((Communications-Control))
Interface Subclass: 13
Interface Protocol: 0
  ► Comm Class Header Functional Descriptor
  ► Comm Class Union Functional Descriptor
  ► Comm Class Reserved Functional Descriptor (36)
  ► Comm Class Ethernet Networking Functional Descriptor
  ► Endpoint 0x81 - Interrupt Input
▼ Interface #0 - Communications-Control (#1) ..... "Fibocom L831-EAU (MBIM)"
Alternate Setting: 1
Number of Endpoints: 1
Interface Class: 2 ((Communications-Control))
Interface Subclass: 14
Interface Protocol: 0
  ► Comm Class Header Functional Descriptor
  ► Comm Class Union Functional Descriptor
  ► Comm Class Reserved Functional Descriptor (27)
  ► Comm Class Reserved Functional Descriptor (28)
  ► Endpoint 0x81 - Interrupt Input
```

Die zwei *Alternativen* für das Interface 0 sind sich ziemlich ähnlich. *Alternate Setting* gibt an um welche Alternative es sich handelt. Alternative 0 ist die, die nach einem Neustart automatisch gewählt wird. Beide Alternativen haben einen *Interrupt Endpoint* mit der Nummer 0x81. Da die Nummer mit 8 beginnt handelt es sich um einen Endpoint der gelesen wird. *Interface Class*, *Subclass* und *Protocol* definieren wieder die Art des Interfaces. Bei der ersten Alternative handelt es sich um die uns aus dem *Interface Association Descriptor* vertrauten Werte. Alternative 1 ist ein Kommunikationsgerät, Unterklasse MBIM (Mobile Broadband Interface Model) und Protokoll MBIM-Protokoll.

Die Gruppenzugehörigkeit beeinflusst die verfügbaren *Functional Descriptors*. Diese beschreiben verschiedene Aspekte dieser speziellen Interface Art und interessieren uns in diesem Moment nicht.

Lange Rede stark gekürzt. Interface 0 kontrolliert die Kommunikation nach NCM oder MBIM. Was auch immer das sein mag.

```
▼ Interface #1 - Communications-Data/Unknown Comm Class Model ..... "Data (DF)"
Alternate Setting: 0
Number of Endpoints: 0
Interface Class: 10 ((Communications-Data))
Interface Subclass: 0 (Unknown Comm Class Model)
Interface Protocol: 1
▼ Interface #1 - Communications-Data/Unknown Comm Class Model (#1) ... "Data (ND)"
Alternate Setting: 1
Number of Endpoints: 2
Interface Class: 10 ((Communications-Data))
Interface Subclass: 0 (Unknown Comm Class Model)
Interface Protocol: 1
  ► Endpoint 0x82 - Bulk Input
    Address: 0x82 (IN)
    Attributes: 0x82 (Bulk)
    Max Packet Size: 512
    Polling Interval: 0 (Endpoint never NAKs)
  ► Endpoint 0x82 - Bulk Output
▼ Interface #1 - Communications-Data/Unknown Comm Class Model (#2) ... "Data (MDW)"
Alternate Setting: 2
Number of Endpoints: 2
Interface Class: 10 ((Communications-Data))
Interface Subclass: 0 (Unknown Comm Class Model)
Interface Protocol: 2
  ► Endpoint 0x82 - Bulk Input
  ► Endpoint 0x82 - Bulk Output
```

Interface 1 hat 3 Varianten.

Alternative 0 hat keine Endpoints, macht demzufolge Datenübertragungs-technisch nichts, also *Datenübertragung aus*. Die anderen beiden Varianten haben jeweils einen *Lese-* und *Schreib-Endpoint* vom Typ *Bulk*. Also massenweise Daten rein und raus - macht Sinn.

Alle drei gehören der Datenübertragungsklasse, Abteilung Daten an. Interface 0 war Abteilung

Kontrolle. In der Abteilung Daten sind keine Unterklassen definiert, weshalb Subclass bei allen dreien auf 0 steht. Das Protokoll der ersten beiden Alternativen ist *Networktransport* Block bei der letzten Alternative *MBIM*.

Interface 3 bietet also drei Alternativen, Aus, NCM Daten und MBIM Daten.

Einen hab ich noch

Der Vollständigkeit halber schauen wir uns noch einen *Endpoint Descriptor* an. Wir sehen seine Nummer (*Address*), den Übertragungsmodus (*Attributes*), die maximale Größe eines Paketes (*Max. Packet Size*) und wie oft nach Daten geschaut werden soll, falls es sich um einen Interrupt Endpoint handelt (*Polling Interval*).

Da die Endpoint Descriptors zu den Interfaces gehören, gibt es keinen Deskriptor für den Control Endpoint der gehört ja zum Gerät. Auf der anderen Seite, seine Adresse ist immer 0, sein Übertragungsmodus immer Control und abgefragt, wird er auch nicht. Also bleibt nur die maximale Paketgröße. Die findet sich tatsächlich oben im *Device Descriptor* als *Device MaxPacketSize*. Alles gut.

Sag mir wo die Texte stehen, wo sind sie geblieben ?

Texte haben unterschiedliche Längen und können in verschiedenen Sprachen vorliegen. Damit die einzelnen Deskriptoren eine feste Größe haben und man mit wenig Aufwand mehrere Sprachen unterstützen kann, werden die Texte nicht in den Deskriptoren gespeichert. Stattdessen wird nur mittels einer Nummer (*index*) auf einen String Deskriptor verwiesen. Dieser String Descriptor enthält dann den Text.

USB Prober ersetzt die Nummern in den Deskriptoren freundlicherweise gleich durch die entsprechenden Texte.

Was sagt uns das ?

Das Gerät dient der Datenübertragung und unterstützt die Datenprotokolle NCM und MBIM.

Zu jedem gibt es eine passendes Daten- und ein passendes Kontrollinterface. Beide Kontrollinterfaces haben jeweils nur einen Interrupt-Lese-Endpoint. Das ist also bestimmt keine serielle Schnittstelle.

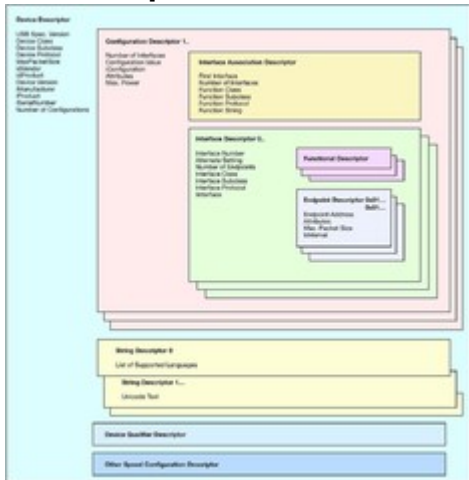
Also kein Umschalten der Konfiguration oder Umschalten auf ein alternatives Interface und schon ist alles wie wir es gerne hätten. Wir werden mit den Standardmethoden und AT Befehlen, nicht weiterkommen.

Also zurück zum Anfang. Im IORegistry Explorer stand ja auch, dass kein serieller Treiber für das Kommunikations-Kontroll-Interface geladen wurde, sondern AppleUSBNCMControl.

Denen, die finden für diese Erkenntnis hat sich der Exkurs in die Welt der Deskriptoren nicht gelohnt, sei gesagt: They will be back.

Wenn das km 42 ist, dann liegt das Ziel hinter einer Kurve, denn zu sehen ist es noch nicht.

PS Descriptor



Beitrag von „Monchi_87“ vom 12. April 2018, 11:52

Ich liebe diesen Schreibstil.

Beitrag von „umax1980“ vom 12. April 2018, 12:03

Den Einwurf mit den Kaugummi-Automaten muss ich noch mal erwähnen, neben dem ganzen fachlichen vielleicht der wichtigste Hinweis....

Beitrag von „burzlbaum“ vom 12. April 2018, 15:47

Zitat von made my day

Wenn das so weiter geht, muss ich ins Handbuch schauen. "Nicht mit mir", denke ich, "ich der Mann, der ein Bit am Geruch erkennt, den noch kein Byte gebissen hat, der morgens mit dem Datenbus zur Arbeit fährt und abends mit dem Adressbus nach Hause kommt, der soll in ein Handbuch schauen ? Haaahhh, niemals"

toi toi toi weiterhin

Beitrag von „burzlbaum“ vom 17. April 2018, 13:11

Hallo Brumbaer!

Ich habe eine Verständnisfrage zum Thema Touchscreen.

[Zitat von Brumbaer](#)

VoodooI2CHID unterstützt mit Hilfe des macos HID Treibers:

Klicken durch Zeigen auf eine Stelle

Drag durch Klicken auf eine Stelle und Bewegen des Fingers

RechstKlick durch Klicken und Gedrückt-Halten.

Scrollen mit zwei Fingern.

Doppelklick funktioniert etwas unzuverlässig, da es schwer ist zweimal kurz hintereinander die selbe Stelle zu treffen.

Bei mir klappt aktuell nur Klick und Drag wirklich zuverlässig. Nutze den ApplePS2Controller.kext und hatte auch mal Voodoo getestet, aber da war das Touchpad dann etwas träge und den Touchscreen habe ich gar nicht getestet. Wenn ich jetzt die VoodooI2CHID testen möchte. Reicht es die ApplePS2... zu ersetzen oder ist zusätzlich eine andere Voodoo Kext nötig? Oder kann man das so Pauschal gar nicht beantworten?

Falls du solche Fragen nicht in deinem Thread möchtest, bitte ich um Entschuldigung und Verschieben.

Danke 😊

Beitrag von „anonymous_writer“ vom 17. April 2018, 13:15

Bin zwar nicht [@Brumbaer](#) und er ist hoffentlich nicht böse wenn ich antworte.

Du benötigst immer denn VoodooI2C.kext und denn VoodooI2CHID.kext.

Hier die Entwicklerseite dazu mit Hilfsanleitung.

<https://github.com/alexandred/VoodooI2C>

Beitrag von „andreas_55“ vom 20. April 2018, 07:06

[Zitat von Brumbaer](#)

Selbstoptimierung macht auch vor SIM Karten nicht halt. Wahrscheinlich ernähren die sich auch nur von veganem Strom.

Pffeww..... 😏

Mensch Brumbaer, siehst Du nicht, dass ich gerade Kaffee trinke? Beinahe die Tastatur unter Kaffee gesetzt. 👍

Beitrag von „Brumbaer“ vom 26. April 2018, 02:11

AppleUSBNCMControl

Ist das eine Drohung oder eine Versprechen, Erlösung oder Verdammnis, Peek oder Cloppenburg ?

Bei dem Namen liegt die Verbindung nahe, dass der Treiber in `AppleUSBNCMIrgendwas.kext` zu finden ist.

Es stellt sich heraus es gibt tatsächlich ein Kext und sein Irgendwas ist Nichts.

"AppleUSBNCMNichts.kext, sehr lustig, ha, ha, ha, was haben wir wieder gelacht".

Das `AppleUSBNCM.kext` hat Alles was man erwartet u.a. eine Info.plist und eine Programmdatei names `AppleUSBNCM`.

Persönchen

In der Info.plist schauen wir uns die `IOKitPersonalities` an, denn die sind der Weg zum Herzen des Kextes.

IOKitPersonalities	Dictionary	(3 items)
AppleUSBNCMControl	Dictionary	(5 items)
IOClass	String	AppleUSBNCMControl
IOProviderClass	String	IOUSBHostInterface
bInterfaceClass	Number	2
bInterfaceProtocol	String	*
bInterfaceSubClass	Number	13
AppleUSBNCMData0	Dictionary	(5 items)
IOClass	String	AppleUSBNCMData
IOProviderClass	String	IOUSBHostInterface
bInterfaceClass	Number	10
bInterfaceProtocol	Number	1
bInterfaceSubClass	Number	0
AppleUSBNCMData13	Dictionary	(5 items)
IOClass	String	AppleUSBNCMData
IOProviderClass	String	IOUSBHostInterface
bInterfaceClass	Number	10
bInterfaceProtocol	String	1
bInterfaceSubClass	Number	13

Es gibt drei Persönchen, die einen von zwei Treibern laden. Entweder `AppleUSBNCMControl` oder `AppleUSBData`.

Bei allen ist die `IOProviderClass IOUSBHostInterface`. D.h. sie hängen sich an ein USB-interface. An welches legen `bInterfaceClass`, `bInterfaceSubClass` und `bInterfaceProtocol` fest. Sie entsprechen der Klasse, Unterklasse und dem Protokoll im USB-Interface-Descriptor.

Wer mit wem

Nach einem Neustart wird die Konfiguration des USB Gerätes auf 1 gesetzt und alle Interface Alternativen 0 ausgewählt. Davon ausgehend, dass niemand was daran geändert hat, hätten wir bei L831-EAU

Interface 0 mit Klasse 2, Unterklasse 13 und Protokoll 0.

Interface 1 mit Klasse 10, Unterklasse 0 und Protokoll 1.

`AppleUSBNCMControl` testet auf Klasse 2, Unterklasse 13 und Protokoll * (egal) und hängt sich

deshalb an Interface 0.

AppleUSBNCMData0 testet auf Klasse 10, Unterklasse 0 und Protokoll 1 und hängt sich deshalb an Interface 1.

AppleUSBNCMData13 testet auf Klasse 10, Unterklasse 13 und Protokoll 1 und hängt sich deshalb an kein Interface, denn wir haben keins mit solch einem Tripel.

Dazu hätte wir die *Info.plist* nicht aufmachen müssen, das zeigt uns auch der *IORegistryExplorer*. Allerdings kann uns der *IORegistryExplorer* nicht zeigen, was nicht geladen wurde.

Der Eintrag für *AppleUSBNCMControl* hat kein *CFBundleIdentifier* Feld, also ist der Treiber in der Programmdatei dieses Kextes gespeichert.

Not Cool Man

könnte sein, mit NCM ist an dieser Stelle aber vermutlich *Network Control Model* gemeint. NCM ist eine Unterklasse der USB CDC (*Communication Device Class*) Klasse. Auf USB.org kann man die entsprechenden Spezifikationen downloaden.

In Kürze NCM unterscheidet Daten und Kontroll-Funktionen (Interfaces). Die Daten Funktionen empfangen und senden Pakete. Diese Pakete bestehen aus den Paketen, so wie sie über das Ethernet verschickt werden würden, ergänzt um Informationen, die dem Modem mitteilen, was und wieviel es da bekommt. Wenn man Pakete in Pakete packt, so nennt man das einbetten.

Wie man sich einbettet, so

Der Kontroll-Kanal kann ein *Embedded Request and Response Protocol* unterstützen.

Man schickt einen besonderen *Request* an den *Control-Endpoint* (An letztes Mal erinnern: Control Endpoint, Requests - Befehl mit optionalen Daten vom oder zum USB Gerät, schon wieder vergessen ?). Die Daten des *Requests* sind aber keine Daten, sondern ein Befehl, eine Aufforderung (*Request*) an das Modem. D.h. der *Request* an das Modem ist in den Daten des *Requests* an das USB-Gerät eingebettet.

Hat das Modem den Befehl ausgeführt, so schickt es von einem *Interrupt-Endpoint* eine Nachricht. Empfängt der Treiber die Nachricht schickt er einen *Request* an den *Control-Endpoint*, er möchte doch mit den Daten - meist die Antwort (*Response*) auf einen *Request* an das Modem - rausrücken. Was der dann auch tut indem er sie in die Daten des *Requests* einbettet. Über den *Interrupt-Endpoint* wird auch eine Nachricht geschickt, wenn das Modem etwas, z.B. dass sich die Empfangsstärke geändert hat, mitzuteilen hat.

Wir erinnern uns an letztes mal, die Interface Deskriptoren für Interface 0 ? Nur einen Endpoint, einen Lese-Interrupt-Endpoint. Also genau das was man für dieses Verfahren braucht. Dann wollen wir das Ding doch mal eintüten oder besser einbetten.

Schwierige Entscheidung

Nun kann man entweder einen neuen Treiber schreiben oder das was Apple anbietet verwenden. Wäre schon schön, wenn man das nicht neu schreiben müsste, Mac Treiber werden in C++ geschrieben und die Treiber sind C++ Klassen, die von der Klasse IOService erben.

Im Idealfall hat man eine Headerdatei mit der Klassendefinition. Natürlich gibt es keine Headerfiles oder sonst eine Dokumentation zu AppleUSBNCM. Apple stellt allerdings die Quellen zu AppleUSBCDC der Superklasse von AppleUSBNCM, zur Verfügung. Allerdings in der Version 4.5. Inzwischen sind wir bei 5.0 und die verwendeten USB Interface- und Device-Klassen haben sich geändert.

Aber wir haben ja eine ausführbare Datei, die unsere Klasse enthält. Man kann nun aus der ausführbaren Datei die Informationen gewinnen, die man zur Erstellung einer Headerdatei benötigt.

Einige recht einfach, andere mit Aufwand.

Denen, die interessiert wie das geht, sei gesagt, dass das den Rahmen dieser Beschreibung sprengen würde, es ist genug Stoff für eine eigene Artikelserie. Es sei nur gesagt, dass Hopper eine große Hilfe dabei ist und dass er einen Python Interpreter enthält mit dem einige Abläufe automatisieren kann.

Bei meinen Reisen durch AppleUSBNCM kam ich an eine Stelle an der ich einen Funktionsnamen gebraucht hätte, den Hopper auch brav ausgab, der aber nicht über Python abzufragen war. Eine Anfrage beim Entwickler bescherte mir 3 Tage später eine Hopper Version mit der das möglich war - wow, besser, WOW.

So nicht

Nachdem die AppleUSBNCM und AppleUSBCDC (Superklasse von AppleUSBNCM) soweit analysiert waren, dass ich wusste, was sie warum taten, versuchte ich eine Kommunikation mit dem Modem herzustellen.

Ich konnte zwar die Embedded Commands senden, aber es gab nie eine Rückmeldung vom Interrupt Endpoint.

Also habe ich einen neuen Treiber geschrieben. für den Fall, dass ich bei der Analyse von AppleUSBNCM etwas übersehen habe. Nein, habe ich wohl nicht.

Also nochmals in die USB Deskriptoren geschaut, ob es denn einen Hinweis auf eine andere Zugriffsvariante gibt, und siehe da: Es gibt keinen.

Wer lesen kann, ist klar im Vorteil

Wir hatten gesehen, dass unser Control Interface (USB Interface 0, Alternative 0) vom Typ Klasse 2, Unterklasse 13 und Protokoll 0 war.

Klasse war Kommunikation, Unterklasse NCM und Protokoll laut Spec *No encapsulated commands / responses*

Tufftäääh, Tufftäääh, Tufftäääh. Da steht laut und deutlich, na eher kontrastreich und gut leserlich *No encapsulated commands / responses*.

Das Ding kann gar keine Embedded Commands/Responses also wird der Embedded Commands/Responses Treiber (AppleUSBNCMControl) zwar geladen, aber das Gerät unterstützt die Funktion nicht. Kann also nicht gehen. Hätte ich auch gleich lesen können. Obwohl, ich nehme an, dass ich es gelesen hatte, aber es ist halt nicht hängengeblieben.

Es ist ärgerlich, die Zeit für die Analyse des Apple Treibers und das Schreiben eines eigenen Treibers verschwendet zu haben. Aber eigentlich spielt es keine Rolle, denn irgendwann würde ich es ja doch tun müssen.

Wat nu, sprach die Kuh

Na ja wenn man nicht weiter weiß schaut man, wie es die machen, die wissen wie es geht. In unserem Fall sind das die Windowers.

Es gibt Tools mit denen kann man die Kommunikation zwischen Computer und Geräten überwachen. Auf dem Mac gibt es z.B. Wireshark um sich Netzwerkpakete anzuschauen und zu analysieren.

Trüffelschweine

Hier geht es darum ein Tool zu finden, das den USB Verkehr unter Windows überwacht und ausgibt. Solche Tools nennt man oft Sniffer, also USB Sniffer gegoogelt. Und man findet auch ein paar. Sie sind nicht billig, aber meist gibt es eine Demo Version - einen Monat Prüfung bevor man sich ewig bindet. Ein Monat sollte langen.

Verschiedene ausprobiert und alle bringen Windows beim Booten zum Absturz. Na super. "Na, super" mit sarkastischem Unterton sagt man ja nicht mehr. Heute sagt man ja: Na, Diesel.

Die Sniffer arbeiten nach dem selben Prinzip, sie hängen einen Schnüffel-Treiber in den USB Treiberstapel. Während der Schnüffeltreiber sich in den USB Stack wühlt, wie das Trüffelschwein seine Nase in die feuchte Erde, funktioniert USB nicht. Dummerweise ist mein Windows auf einer USB Platte und das Trüffelschwein zieht dem Windows die Platte weg und Windows landet mit dem Schwein Nase an Nase im Dreck.

Neues Fenster

Das Miix hat für Massenspeicher genau einen M.2 Slot. Das macht die Wahl des Ortes für die Windows-Installation leicht, mit auf die M.2. An einem runden Tisch ist immer noch in Platz, alle müssen nur etwas zusammenrücken. Dankbarer-weise geht das auch bei M.2 SSDs, obwohl die nicht rund sind. Das Festplattendienstprogramm geöffnet und 128GB von der HFS+ Partition abgezwickelt.

Wie pflegen die Helden zu sagen, "heute ist ein guter Tag, mal wieder die EFI Partition auf

einen USB Stick zu kopieren", denn "Kopierst du in der Zeit, bootest du in der Not".

Dann Windows von einer USB Stick installiert. Die Seriennummer des Miix hat das Windows ungefragt übernommen.

Und Windows geht einwandfrei, Sniffer installiert, geht auch. Nur macOS geht nicht mehr. Nicht mal an Clover komme ich ran. Na, Diesel.

Eiskalt

Da läuft's einem kalt den Rücken runter. Aber mit *BOOTICE* bekommt man schnell warme Füße. Mit *BOOTICE* kann man die EFI Booteinträge manipulieren. Das Programm hat ein etwas altmodisches Aussehen, erfüllt aber seinen Zweck. Man kann alle notwendigen Schritte mit *BOOTICE* erledigen.

Zuerst mountet man die EFI Partition (Reiter Physical Disk, Button Parts Management. Partition anwählen, Button *Assign Drive Letter*).

Dann fügt man einen Eintrag für `\EFI\BOOT\BOOTX64.efi` hinzu (Reiter *UEFI*, Button *Edit boot entries*, Button *Add*, Datei auswählen).

Dann bewegt man den Eintrag an den Anfang der Liste (Button *Up*), und speichert die Änderung (Button *Save current boot entry*).

Programm beenden, Neustart, fettig.

Und das nächste mal wird geschnüffelt.

Beitrag von „Brumbaer“ vom 7. Mai 2018, 00:06

Windows-Phone

Irgendwie schon, aber Windows-Modem trifft es besser.

Verbindung mit dem Netz hergestellt, den Schnüffler an der Nase gezogen, damit er anfängt seinem Namen Ehre zu machen, und schon "geht dat los".

Wer zu spät kommt, den bestraft das Leben. In diesem Falle ist es zu spät für Screenshots, denn die Lizenz für die Testversion des Sniffers ist abgelaufen. Deshalb gleich weiter zu den Ergebnissen.

- Eintracht - Hamburg 3:0

- Hannover - Hertha 3:1
- USB - Modem Encapsulated commands und responses.

Das Ergebnis ist überraschend, man hätte meinen sollen, dass Hertha gewinnt und dass das mit dem *Encapsulated commands and responses* nicht funktioniert. Es gibt aber einen Grund warum die *EC&R* hier funktionieren.

MBIM

steht nicht für eine mobile Glocke, sondern für *Mobile Broadband Interface Model*. Der eine oder der andere (mehr als zwei werden es kaum sein) wird sich vielleicht erinnern, dass es unterschiedliche USB-Interface-Deskriptoren in Abhängigkeit vom *Alternate Setting* gab. Eine Variante trug *NCM* im Namen, die andere *MBIM*.

MBIM ist eine Spezifikation zur Kommunikation mit USB-Breitband-Modems. Sie unterscheidet sich in für uns zwei wichtigen Dingen von NCM.

- Das Modem muss über Modem Encapsulated commands und responses gesteuert werden können.
- Die Daten entsprechen nicht Ethernet-Paketen, sondern Internet-Paketen. Es gibt Internet-Pakete mit 4 Byte langen IPs (IPv4) und welche mit 16 Byte langen IPs (IPv6), das soll uns an der Stelle aber nicht belasten.

À la carte

Was hätten wir den gerne ?

Da schauen wir erst mal in die Karte, was mich dazu bringt eine Prepaid Karte für den Miix zu kaufen, denn die Bastelei wird wohl noch etwas dauern und mein iPad braucht seine Karte.

Prepaid ist ok und schon habe ich so eine neue, schlanke und ranke Karte. Wenn das Miix in Betrieb geht, kann ich sie durch die andere Karte ersetzen.

Ziel ist es das L831-EAU auf MBIM umzuschalten und dann erst die Kommunikation zu starten.

In Memoriam



Das ist die "Treiber" Struktur. Als erstes knöpfen wir uns *AppleUSBNCMControl* vor. Einen

Ersatz-Treiber hatte ich ja schon geschrieben, nur hat er nicht funktioniert.

Eigentlich sollte er funktionieren, also nur die Umschaltung auf MBIM hinzugefügt. Neue Sim Karte rein, Neustart und das Modem antwortet.

Wieder einmal eine Spec runtergeladen und nachgeschaut, welche Befehle das Modem laut Spec unterstützen muss. Hatte ich erwähnt, dass Vorschau keine gute Wahl zum Anschauen von großen PDFs ist. Falls nicht: **Vorschau ist eine miese Wahl zum Anschauen großer PDFs.**

Die Liste mit den unter Windows übertragenen Befehlen abgeglichen und eine Initialisierungssequenz abgeschickt.

Sieht ganz gut aus, bis mit der Sim Karte kommuniziert werden soll, dann überträgt das Modem nur eine Fehlermeldung.

Die Fahrscheine bitte

Das Modem findet keine Sim Karte. Ich finde sie hingegen sofort, sie steckt im Kartenslot. Also Windows gebootet und auch dort wird die Karte nicht gefunden. Ich wollte schon auf Windows schimpfen, bis mir einfiel, dass es mit meinem Treiber auch nicht geht. Karte rein, Karte raus, kein Unterschied. Lustig, die Karte rastet ein und um sie auszuwerfen drückt man sie erst tiefer rein. Das war bei der anderen Karte nicht so. Da sieht man mal wieder, dass diese halbe Hemden nicht die Fusstapfen von uns King Size Katzen füllen können. Die andere Karte hat wohl die Kontakt Pins ein wenig zurückgebogen und nun kommen die Pins nicht mehr an die Kontakte der schlanken Karte. Ein Fetzen Papier behebt das Problem, bis ich dazu komme, den Miix zu öffnen und die Pins zurück zu biegen.

Contact

Lange darauf hingearbeitet und trotzdem kommt der Erstkontakt mit dem Handy-Netz irgendwie überraschend.

Nun besteht die Hoffnung die bestehende Verbindung in den NCM Modus rüberzuretten, damit ich nicht auch noch den Daten-Treiber neu schreiben muss.

Geht nicht. Hätte ich die Spec aufmerksamer gelesen hätte ich es erst gar nicht probiert, das steht nämlich drin, dass es nicht geht. Wenn man zwischen den Modi umschaltet, wird alles zurückgesetzt und Control Interface auf MBIM und Daten Interface auf NCM geht schon gar nicht.

IOEthernetController

Macos ist gutgläubig. Wenn es aussieht wie ein Ethernet-Controller und redet wie ein Ethernet-Controller, dann denkt macos es ist ein Ethernet-Controller und lädt die zugehörigen Treiber für die Einbindung in das System.

Die Klasse *IOEthernetController* ist leidlich dokumentiert und es gibt ein paar Beispieldateien.

Das lässt einen den Treiber schneller schreiben, fördert aber nicht unbedingt das Verständnis. *IOEthernetController* dient letztendlich dazu Ethernet-Daten-Pakete zu empfangen und zu verschicken. Wie vorher erwähnt, hat bzw. will MBIM, aber Internet-Daten-Pakete. Es mag sein, dass es möglich ist *IOEthernetController* zu umgehen, aber ich habe leider keine Möglichkeit gefunden.

MBIM und *IOEthernetController* sind eher eine von den Ehen bei denen man von vorne herein vermutet, dass es sich um eine Zweckehe handelt.

Was will uns der Künstler damit sagen ?

Laut OSI-Schichten-Modell ist die Internet-Datenpaket-Übertragung Teil des Layers 3, des *Network Layers*. Die Ethernet-Datenpaket-Übertragung Teil des Layers 2, des *Data Link Layers*. Das macht es nicht klarer, oder ?

Im Layer 3 wird definiert, was von wo nach wo soll. In unserem Falle sollen von der IP 123.456.789.6 zur IP 123.456.789.77 übertragen werden. Wie die Daten, Adressen und Hilfsinformationen formatiert werden sollen, gibt die jeweilige Spezifikation vor. In unserem Fall als IPv4- oder IPv6-Paket.

Der Layer 2 präzisiert, wie das geschehen soll. In unserem Fall laut Ethernet-Protokoll. Dummerweise überträgt Ethernet aber keine Daten zwischen Geräten mit IPs. Das Ethernetprotokoll verwendet die *Ethernet Hardware Address (EHA)* statt der IP um die Geräte zu identifizieren. Ein anderer Name für *Ethernet Hardware Address* ist *MAC Adresse*. Das *MAC* kommt von einem Sublayer des *Data Link Layers* indem die Adresse Verwendung findet.

Soll ein Internet-Paket über Ethernet übertragen werden, muss man zuerst die *EHA* finden, die zur jeweiligen *IP* gehört und dann das Internet-Paket als Payload (Daten) in ein Ethernet-Paket packen. Dazu wird dem Internet-Paket ein Header vorangestellt und eine Prüfsumme hintenangestellt.

Früher war Alles besser

Früher bei NCM war das alles super. Denn NCM arbeitet mit Ethernet-Paketen. Bekommt unser Pseudo Ethernet-Controller ein Ethernet-Paket, so muss er es nur an das Modem weiterschicken.

Aber MBIM hat mit Ethernet und dessen Paketen nichts am Hut. Versucht ein Treiber die *EHA* zu einer *IP* zu finden, so antwortet normalerweise der Router. Da der Router aber nichts von unserem USB Gerät weiß, hält er sich raus. Also habe ich dem Treiber eine Funktion spendiert, die Anfragen nach der *EHA* zur *IP* des Modems beantwortet. Der Interessierte schickt eine Anfrage über das Ethernet. Diese erfolgt laut *ARP (Address Resolution Protocol)* Protokoll. Und der Treiber schickt eine Antwort laut des selben Protokolls. Jetzt kann jedes Ethernet Gerät das Modem anhand dessen *IP* finden, schon mal sehr hilfreich.

Zahlenspiele

Die IP des Modems wird vom Netzwerkprovider zugewiesen, aber im Netzwerk-Kontrollfeld erscheint die IP nicht. Es gibt laut ARP Protokoll eine Möglichkeit, seine IP in die Welt herauszuschreien, damit alle Geräte die IP zu einer EHA oder umgekehrt kennen. Leider lässt sich das Netzwerk-Kontrollfeld von dem Geschrei nicht einschüchtern und ignoriert diese ARP-Pakete.

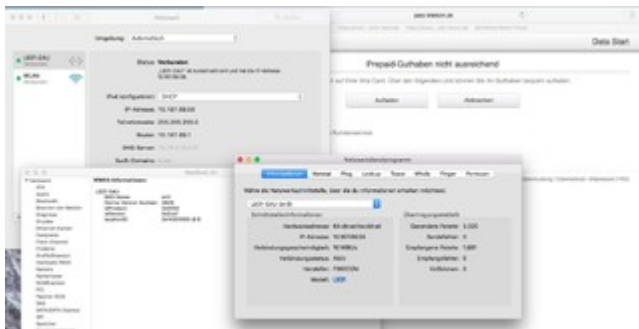
Das Netzwerk-Kontrollfeld sieht allerdings die Option vor die IP Adresse über DHCP zu beziehen. Also hat der Treiber eine weitere Funktion bekommen. Er mimt einen DHCP Server und gibt die IP Adresse, Gateway Adresse und DNS Server Adresse des Modems an das Netzwerk-Kontrollfeld weiter. Meine Güte bei den vielen Rollen, die der Treiber übernimmt, bekommt er noch eine Identitätskrise.

Jetzt wo ich das schreibe frage ich mich, ob es nicht einfacher gewesen wäre, im Netzwerk-Kontrollfeld eine beliebige IP zu setzen und den Treiber eine Adressumwandlung vornehmen zu lassen. Die guten Ideen hat man immer zu spät.

Geht's ?

Na ja, kommt ein Paket von macos muss man das Internet-Paket aus den Ethernet-Paketen herauslösen und an das Modem schicken. Im umgekehrten Fall muss man ein Ethernet-Paket aus den Internet-Paketen machen und an macos weiterleiten.

Es lässt viel Spielraum für Fehler, aber mit Geduld, Spucke, den Specs und Wireshark, geht's dann doch.



Und jetzt ?

Gute Frage. Das Modem funktioniert.

Es kommt als Ethernetkarte daher, weshalb es immer vor dem WLAN bevorzugt wird. Eine automatische Umschaltung erfolgt nicht. Will man es nicht muss man es ausschalten. Da gilt es herauszufinden, wie man es als WWAN kennzeichnet so dass es auch das Netzwerk-Kontrollfeld als solches erkennt.

Aber das ist eher nervig als spannend.

Punkt ist, dass Modem und Touch funktionieren, aber noch Raum für Verbesserungen lassen. Aber wie sooft, ist der interessante Teil, das Zum-Laufen-Bringen, das Falten-Rausbügeln ist dann mühselig.

Ich könnte mir natürlich auch erst anschauen ob man die Kameras zum Laufen bringt.

Ich denke ich werde eine Pause einlegen und dann Touch und Modem überarbeiten, und dann erst die Kameras anschauen, falls mir bis dahin nichts besseres einfällt oder mir ein interessanterer 2 in 1 unterkommt.

Denn irgendwie ist der Funke zum Miix nicht wirklich übergesprungen. Vielleicht kennen wir uns einfach zu gut.

Beitrag von „griven“ vom 7. Mai 2018, 00:23

Ich finde es nach wie vor extrem krass wie tief Du in der Materie steckst und was Du daraus machst. Einfach nur alle Daumen hoch dafür 😄

Beitrag von „apfelnico“ vom 7. Mai 2018, 09:18

Du schreibst großartig, und manches verstehe ich sogar. Möglicherweise kann ich hier helfen:

[Zitat von Brumbaer](#)

Es kommt als Ethernetkarte daher, weshalb es immer vor dem WLAN bevorzugt wird.

Gehe in die Systemeinstellungen, dort auf Netzwerk. Hier jetzt unten auf das "Zahnradchen", dort auf "Reihenfolge der Dienste festlegen ...".

Beitrag von „Brumbaer“ vom 7. Mai 2018, 10:02

[@apfelnico](#)

Danke, ich hatte das Menü bestimmt 100 mal offen, aber die Option nie wahrgenommen.

Beitrag von „GucciGucciGu“ vom 21. Mai 2018, 14:22

Alter Fuchs, bist du gscheit ...

Vor so viel Wissen und Arbeit kann man nur den Hut ziehen.

Herrlich wie du deine Artikel schreibst und deine Lösungen erläuterst. Bei manchen Sachen wie z.B. der ACPI Spezifikation steig ich aus fehlendem Hintergrundwissen leider aus. Dafür bin ich bei anderen Sachen wie bei der letzten MBIM Geschichte voll dabei. 👍

Wie genau du die Treiber schreibst/modifizierst, wäre manchmal noch interessant.

Beitrag von „Brumbaer“ vom 22. Mai 2018, 19:43

Programmiersprache für macos Treiber ist C++. Die IDE natürlich XCode.

Die Apple eigene Dokumentation gibt genug Hinweise darauf, wie man einen Mac-Treiber schreibt, bzw. wie er aufgebaut ist und welche Anforderungen er erfüllen muss.

Zusätzlich gibt es Bücher darüber und über macos Interna.

Falls man noch nicht programmieren kann, ist ein Treiber womöglich als Einstieg nicht die beste Wahl.

Beitrag von „mauritzius“ vom 11. Juni 2018, 15:55

Ich habe mich gerade wegen des Fibocom-Moduls registriert: Ich habe einen T470 und das letzte fehlende Stück ist das Fibocom L831-EAU Modul, das du hier offenbar auch zum Reden bewegen möchtest.

Bisher habe ich aber einige andere Versuche gemacht: Im AT-Command-Manual steht, dass das Modem mehrere Betriebsarten unterstützt. Ich habe bisher alle Modi durchprobiert, meines verhält sich folgendermaßen:

- 0: 3AT + 3NCM
- 1: 1AT + MBIM
- 2: 3AT, 3NCM und MBIM
- 3: MBIM

Ich wollte eines der AT-Interfaces über CellPhoneHelper.kext ansprechen, aber das Data-Interface bleibt bei mir immer ohne entsprechenden Data-Treiber. Es sieht so aus, als bindet der ACM-Treiber nicht korrekt zum Interface.

Bist du in der Zwischenzeit schon einen Schritt weiter gekommen?

Beitrag von „LovelsHackintosh“ vom 2. Juli 2018, 20:40

Also Respekt was du hier draus gemacht hast !! ich selbst sitze an einem HP 2570p mit Touch Screen aber muss noch ein Withlist Bios finden um den MSata al auch den Halfsize msata Steckplatz frei zu machen. zusätzlich muss ich die DsDt für die Batterie bearbeiten und versuche das Touch Display zum laufen zu bringen.

Aber wie tief du eingedrungen bist ist der Wahnsinn !