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

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 janz 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 gehoffte, 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 Schatzzzzz ?

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 Batteriemanagement 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 de 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.

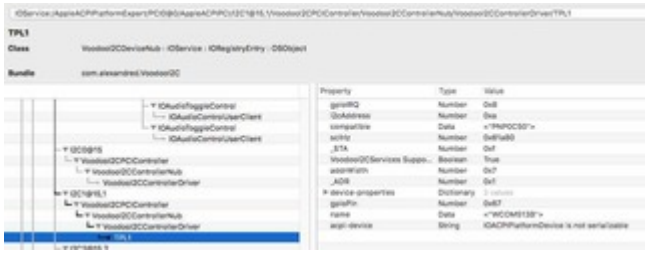
Also schaut der Interpreter in I2C1 und dann in PCI0 und in _SB und findet nirgendwo eins.
In / (die oberste Ebene der Objekthierarchie) findet der Interpreter letztendlich das SDM1 in GNVS und verwendet dessen Wert. SDM1 in GNVS ist in der obersten Hierarchiestufe und kann immer und überall gefunden werden. Wegen des überall, heißt es globales Objekt.

Wenn wir nun dafür sorgen, dass ein anderes SDM1, dass 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 VodooI2C 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, VoddooI2CHID 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.