Kext as Kext can oder USB 3.0 ohne USBInjectAll

Beitrag von „Brumbaer“ vom 17. August 2017, 14:39

Ran ans Eingemachte.
Dann wenden wir das Gelernte mal an.

Ich habe hier ein Asus Strix Z270i Gaming Board.
Es wird als iMac18,3 betrieben, wegen Kaby Lake und so.

Ziel ist: Keine Änderungen an der DSDT, kein USB InjectAll.
USB 2.0 Ports funktionieren, deshalb kann man mit einem USB 2.0 Stick (oder einem USB 3.0 Stick an einem USB2.0 Anschluss oder hinter einem USB 2.0 Hub) macos installieren.

Wenn ich nur wüsst was drinnen ist.
Wenn man etwas über Geräte und Services wissen will, hilft *IORegistryExplorer* weiter.

Wenn man *IORegistryExplorer* startet, sieht man die *IOService (Plane)*. Das ist eine Auflistung der installierten Services.
Da Services häufig an Geräte gebunden sind, sieht es auf den ersten Blick wie eine Auflistung der Geräte aus.
Man kann sich auch die *IOACPIPlane* anzeigen lassen. Die zeigt uns dann die Geräte an, wie sie in der DSDT/SSDT verzeichnet sind.

XHCI Controller werden in der DSDT für gewöhnlich *XHC* nix oder irgendwas benannt. Gott sei Dank, sonst ging die Sucherei jedes Mal aufs Neue los.
Die Services verwenden ebenfalls den Namen aus der DSDT/SSDT und somit heißen die auch irgendwas mit XHC.

Auf der rechten Seite sehen wir, dass die DSDT ein Gerät namens *XHC* hat, das hat einen *RHUB* und dann 26 Ports.
Auf der linken Seite sehen wir keinen *RHUB* und nur 15 Ports.
Von der Problematik mit den 15 Ports hat jeder schon gehört. Es werden, wenn wie nicht im ersten Artikel beschrieben, die Ports explizit ausgewählt werden, die ersten 15 mit Services versehen.

Ein Blick auf die rechte Seite zeigt uns das sind HS01-HS14 und SS01 und tatsächlich genau diese Ports tauchen auf der linken Seite auf.
Der *RHUB* taucht auf der linken Seite nicht auf. USB Controller haben interne HUBs an denen die Ports angeschlossen sind.
Das ist sinngemäß ein eigenes Gerät bzw. eine eigene Funktionalität, aber Treiber-/Service-seitig ist dessen Behandlung Teil des XHC Treibers. Deshalb sieht man ihn auf der ACPI aber nicht auf der Service Seite.
Auf der rechten Seite sieht man hinter jedem Ports die Portnummer.
Auf der rechten Seite sieht man die erwarteten Ports plus USR1 und USR2.
Was sind USR1 und USR2.
Der Controller nummeriert die Ports durch.
Port 0x00 ist der Hub !!!!!!!!!
Port 0x01 bist Port 0x0E sind HS01 bis HS14
Port 0x11 bist Port 0x1A sind SS01 bis SS10.
Dem Ordnungsfanatiker fällt auf dass da eine Lücke ist, 0x0F und 0x10 sind nicht belegt.
0x0F ist nicht belegt, weil es nur 14 HS Ports gibt und 0x10 ist nicht belegt, da es der HUB für die SS Ports wäre.
Damit die Portnummern aber nicht frei sind und man beim "Durchzählen" nicht auf "Nichts" trifft, werden sie mit zwei Dummy-Ports belegt.

Was soll mir das sagen ?
Was wir anhand dieser Daten sagen können ist:
Der Gerätename des XHCI Controllers ist *XHC*.
Die *Portnummern* sind 0x01 bis 0x0E für HS Ports und 0x11 bis 0x1A für SS Ports.
Es wird ein Treiber für den Controller geladen, sonst würden die Ports nicht in der Service Plane auftauchen.
Es gibt keinen Eintrag in *AppleUSBXHCIPCI* für diese Mactyp/Gerätename Version, sonst wären nicht die ersten 15 Ports in der Service Liste, sondern die die der Original Mac hat.
HS02,5,7,9 und 11 haben Dreiecke, das heißt die Ports sind in Benutzung. Diese Ports werden auf jeden Fall von meinem Computer benutzt.
"Entfaltet" man das Port, sieht man welcher Service/Gerät an dem Port aktiv ist. Dann weiss man auch schon welche Buchse am Computer mit diesem Port verbunden ist.
Wir können sehen, dass Port HS11 mit dem internen BT Adapter verbunden ist.

Sonst noch was ?
Auf der linken Seite sehen wir zwei Einträge für XHC, also zwei Services. Warum ?
Bei unseren gestrigen Betrachtungen haben wir gesehen, dass der XHCI Treiber in Abhängigkeit von einem PCIDevice Treiber gestartet wird.
Da der PCIDevice Treiber zuerst "da war" steht es oberhalb des XHCI Treibers. Beide heißen XHC, weil das Gerät laut DSDT/SSDT XHC heißt.

Auf der rechten Seite sehen wir, dass der Treiber *AppleUSBXHCIPCI* ist und dass er aufgrund von *IOPCIClassMatch* geladen wurde.
Erinnern wir uns, das ist der Notfall-Treiber, wenn es keinen Controller-eigenen Treiber gibt.
Auf der linken Seite Finden wir den *class-code, vendor-id* und *device-id*(Produkt-Id). Das sind die Rohdaten aus den Controllerregistern, deshalb sind die Werte etwas verdreht.
Unter *name* finden wir eine lesbarere Form. 0x8086 steht für Intel und 0xa2af ist die *device-id* des XHCI Controllers des Z270 Chipsatzes.

Geradlinig ist anders
Voll interessant - na ja vielleicht nicht - aber wir werden es später brauchen.

AHDS ?
Treiber wird geladen, Ports sind da, nur nicht genug, meine Aufmerksamkeitsspanne ist am Ende, l. mich.
Na ja wenn das so ist, den USB-Port-Anzahlveränderungs-Patch rein und weiter bei "Was will man mehr ?".

Nö
Tja, die Aufgabe ohne Patch ist bei obiger Lösung nicht erfüllt. Außerdem ist mir der Patch suspekt. Also die Ports auf 15 begrenzen.

Port sucht Buchse
Dazu müssen wir herausfinden welche Ports auf Buchsen oder Header geführt sind.
Ein paar Ports haben wir schon gefunden, nämlich die die in der Service Plane mit einem Dreieck markiert waren.
Jetzt stecken wir nacheinander in alle anderen Ports ein USB 2.0 Gerät oder Hub und schauen bei welchem Port dann ein Dreieck erscheint.

Die Anschlüsse sind

- HS01 intern USB 3.0
- HS02 intern USB 3.0
- HS03 hinten außen rechts USB 3.0
- HS04 hinten außen rechts USB 3.0
- HS05 hinten außen links USB 3.0
- HS07 hinten Mitte USB 2.0 only

- HS08 hinten Mitte USB 2.0 only
- HS09 hinten Mitte USB 2.0 only
- HS10 hinten Mitte USB 2.0 only
- HS11 intern BCM Bluetooth Module

Das sind schon mal 10 HS Ports. Davon sind 5 Ports an USB 3.0 Buchsen geführt. Diese haben jedes zusätzlich noch ein SS Port. Macht zusammen 15. Wenn das kein Wink des Schicksals ist.

Die Frage ist wie bekommen wir raus welche SS Ports an den Buchsen/Headern liegen. Na ja wir tragen die ersten 5 ein und schauen welche Verbindung haben. Werfen die anderen raus probieren wieder mit dem nächsten Satz Ports.
Oder aber wir schauen in die Dokumentation und graben in der Erfahrungskiste.
Die Erfahrung sagt uns, das Asus gerne SS und HS Ports mit der selben Nummer an die selbe Buchse legt.
D.h. das wären dann SS01 bis SS05, weil HS01 bis HS05 an den USB 3.0 Buchsen/Header anliegen.
Asus benennt in seinen Handbüchern die Stecker/Buchsen nach den Port Nummern. USB3_12 Ist ein USB3.0 Header an dem SS01 und SS02 anliegen.
Ein kurzer Blick ins Handbuch bestätigt unsere Vermutung bezüglich SS01 bis SS05

Die komplette Port-/Buchsenliste sieht also so aus:

- SS01, HS01 intern USB 3.0
- SS02, HS02 intern USB 3.0
- SS03, HS03 hinten außen rechts USB 3.0
- SS04, HS04 hinten außen rechts USB 3.0
- SS05, HS05 hinten außen links USB 3.0
- HS07 hinten mitte USB 2.0 only
- HS08 hinten mitte USB 2.0 only
- HS09 hinten mitte USB 2.0 only
- HS10 hinten mitte USB 2.0 only
- HS11 intern BCM Bluetooth Module

Ein Blick zurück ohne Zorn
In der *AppleUSBXHCIPCI* wurden dem Treiber die Ports mit Hilfe von *AppleUSBMergeNub* und einer Parameterliste zugewiesen.
Es wird also ein Service aus einem Kext heraus gestartet. Der Service muss nicht im selben Kext stehen, denn *AppleUSBMergeNub* ist ja nicht Teil des *AppleUSBXHCIPCI*.

Also können wir ein Kext erzeugen, das AppleUSBMergeNub startet und dem Treiber unsere

Portliste übergibt. Wir müssen nichts patchen oder verändern, wir fügen einfach nur ein neues Kext hinzu.

Es ist ein Kext

Wir legen einen Ordner an, der unser Kext sein wird.
Der Ordner bekommt den schönen Namen BBStrixUSB.kext oder was auch immer ihr sonst bevorzugt. Hauptsache der Suffix ist kext.
Das Icon im Finder wird mit dem Umbenennen zum Kext Icon wechseln.
Ein Rechtclick auf das Icon und "**Paketinhalt zeigen**" öffnet den Ordner.
Wir werden keinen Contents Ordner verwenden und so legen wir auch keinen an.
Nun erzeugen wir die *Info.plist* Datei.
XCode hat unter **New -> File** eine Vorlage für Property Lists. Diese erstellt eine leere PList.
Es ist wichtig die Datei *Info.plist* zu nennen sonst funktioniert das Kext nicht. Außerdem schaltet der Name *Info.plist* im PList Editor PopUps für die Feldnamen frei. Wählt man den Feldnamen, wird auch der Datentyp gleich passend gewählt - praktisch.
Man kann nun die benötigten Felder hinzufügen. Man kann natürlich auch eine vorhandene Info.plist kopieren und rauswerfen, was man nicht braucht und editieren was nötig ist.
Wir fangen mit den allgemeinen Kext Feldern an.

Weniger geht nicht, oder ?

Das sieht im PList Editor so aus

Ich will nicht ausschließen, dass Clover auf ein paar dieser Felder verzichten kann, wer will kann es ja mal ausprobieren.
Der *Bundle Identifier* muss auf jeden Fall vorhanden sein. Diesen Namen darf sonst kein Kext tragen. Es ist üblich mit der eigenen umgedrehten Webdomain anzufangen und mit dem Projektnamen zu enden. Keine Leerschritte.
Bundle name, und die zwei *Bundle versions*, kann man ebenfalls frei vergeben. *Bundle version* muss allerdings das Format Zahl.Zahl.Zahl haben.
Den Rest übernimmt man wie er da steht.

Jetzt wird's persönlich

Wie wir wissen machen die *IOKitPersonalities* die Arbeit.
Also legen wir sie an. Wir nennen unsere Personality *Strix 270i Gaming*. Apple hätte sie wohl iMac18,3-XHC genannt.

Die Überprüfung soll starten, wenn ein XHCI Treiber geladen wird (*IOProviderClass*)

Der Rechner vom Typ iMac18,3 (*model*) ist.
Und unser Gerät *XHC* heisst (*IONameMatch*). In *AppleUSBXHCIPCI* steht an dieser Stelle *XHC1*, aber wir haben im IORegistryExplorer gesehen, dass unser Device *XHC* heisst. Statt das Device in der DSDT oder Clover umzubenennen, ändern wir hier den *IONameMatch*, da wir schon mal hier sind und dann auch nur an einer Stelle Änderungen vornehmen.
Unser Test hat einen *IOProbeScore* von 5000, um auch in Zukunft wichtiger als andere Einträge mit den selben Bedingungen zu sein.

Der Treiber der geladen werden soll ist, wie wir im Original gesehen haben, *AppleUSBMergeNub* im Kext mit dem *Bundle Identifier com.apple.driver.AppleUSBMergeNub*.
Jetzt müssen wir nur noch die Ports eintragen. Die gehören unter *IOProviderMergeProperties*.

Der erste Eintrag bei den *IOProviderMergeProperties* ist der *page-count*, der die höchste Port-Id enthält.
Danach kommt die Liste der Ports.
Die Daten des *port-count* Eintrages tragen wir später ein, wenn wir die höchste Portnummer kennen.
Um Platz zu sparen nur der relevante Ausschnitt

Wie ein Eintrag für ein Port aussieht haben wir ja gestern beim Betrachten von AppleUSBHCIPCI gesehen.
Für unsere erstes Port HS01 sieht das dann wie folgt aus

Unser erstes Port ist auch das erste Controller Port und ein HS Port deshalb heißt es HS01.
Es ist auf einen Header geführt, deshalb ist der *USB Connector* Typ 255
Die Port Id ist 0x01

Der am selben Stecker liegende SS Port ist der erste SS Port des Controllers und heißt deshalb SSP1 - weil es im iMac17,1 Eintrag auch so hieß. Wir könnten es auch SS01 oder XA4R nennen. Das erhöht den Wiedererkennungswert für Eingeweihte, verwirrt aber alle anderen, deshalb verwenden wir einen "Standardnamen".
Der USB Connector ist der selbe und die Port Id 0x11.

Das macht man für alle Ports.
Das Port mit der höchsten Id ist SSP5

Das liegt an einer USB 3.0 Typ A Buchse, deshalb ist der USB Connector 3. Die Port Id ist 0x15 das ist deshalb erwähnenswert, weil wir die höchste Port Id im port-count Eintrag brauchen.

Die ganze Datei sieht dann so aus, beachte den bei port-count eingetragenen Wert.

Feddich, wie en Reddich.
Yeah, lasst es Konfetti regnen.

In den EFI/CLOVER/kexts/Other Ordner der EFI Partition kopiert, neugestartet und IORegistryEditor zeigt uns

Was will man mehr ?
Z.B. das es funktioniert.
Wir stellen nämlich schnell fest, dass die USB 3.0 Einträge nicht funktionieren.
Wie wir aus dem IORegistryEditor wissen, wurde der "Notfall Treiber" für XHCI Geräte geladen. der scheint dann wohl nicht zu passen, zumindest nicht für USB 3.0.

Die Qual der Wahl
AppleUSBXHCIPCI hat eine Reihe von auf verschiedene Controller zugeschnittene Treiber. Welchen nehmen ?
Ausprobieren oder einen "educated guess" - gibt's da einen deutschen Ausdruck für ? - wagen.
Wie wir aus dem IORegistryEditor wissen, hat user USB Controller die Primary-Id 0xa2af8086.
Bei Durchsicht der Treiber Einträge in *AppleUSBXHCIPCI* finden wir einen für 0xa12f8086 - das Vorgängermodell.
Also probieren wir es doch mit dem.

Um den Treiber starten zu können brauchen wir wieder ein Kext. Na ja wir haben schon eins, also verwenden wir das.
Die Struktur für einen Treibereintrag haben wir gestern gesehen.

Also legen wir eine neue Personality an und nennen sie *AppleUSBXHCISPTB Z200* oder irgendwie anders.

Unser Treiber soll geladen werden, wenn ein PCIDevice entdeckt wird und es die Id 0xa2af8086 hat.
Den *IOProbeScore* setzen wir auf 5000, damit macht unser Eintrag einen auf wichtig.
Wir schreiben ja keinen eigenen XHCI Treiber, sondern nehmen einen aus dem *AppleUSBXHCIPCI* Kext. Das ist dem Kernel als *com.apple.driver.usb.AppleUSBXHCIPCI* (*CFBundleIdentifier*) bekannt.
Der Treiber den 0xa12f8086 (Vorgänger) verwendet ist (*IOClass*) *AppleUSBXHCISPT*. Also nehmen wir den auch.
Die letzten beiden Einträge übernehmen wir, da wir davon ausgehen, dass unser Controller alles kann, was der alte konnte.

Und war's das ?
In den Other Ordner, Neustart und
Alles geht, wie es soll.

Das war's

P.S.
Im Anhang findet ihr das Kext.