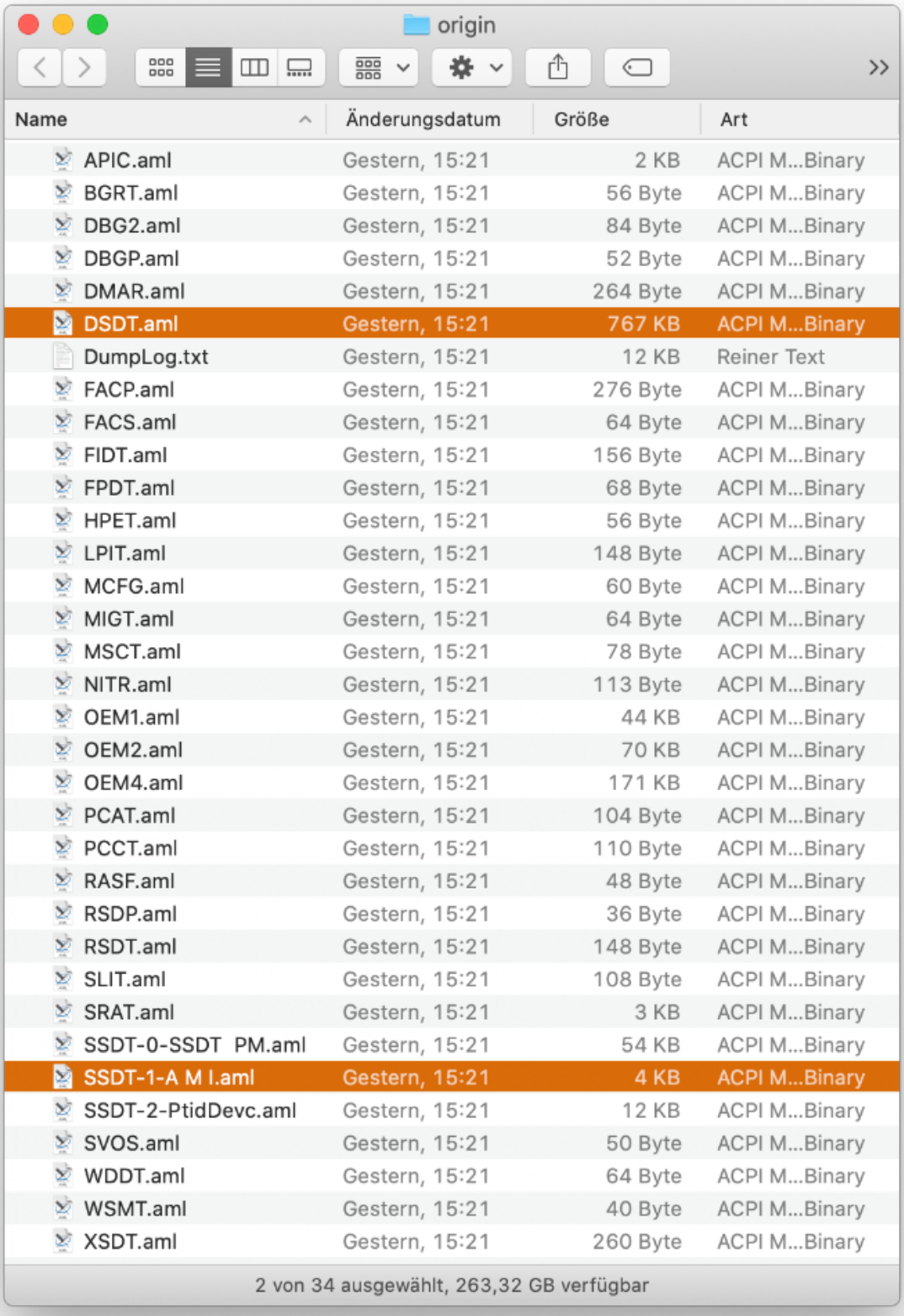
USB mittels SSDT deklarieren

Beitrag von „apfelnico“ vom 11. Oktober 2021, 13:24

Zuerst schauen wir uns die vorhandene „ACPI“ (Advanced Configuration and Power Interface) an. Darin vornehmlich die „DSDT“ (Differentiated System Description Table) und eine SSDT (Secondary System Description Table) für USB. Gerne nutze ich dafür einen Stick mit dem Bootloader „Clover“. Selbst wenn ihr den Bootloader „OpenCore“ nutzt, ist ein solcher Clover-Stick durchaus sinnvoll. Der muss gar nicht das vorhandene macOS starten können, es reicht völlig, wenn das Clover-Menü erreicht wird. Hier die Taste „F4“ drücken, und schon werden in „EFI\CLOVER\ACPI\origin“ die originalen ACPI-Tabellen geschrieben. Das ist nur eine von vielen Varianten, wie ihr an eure unveränderten ACPI-Tabellen kommt

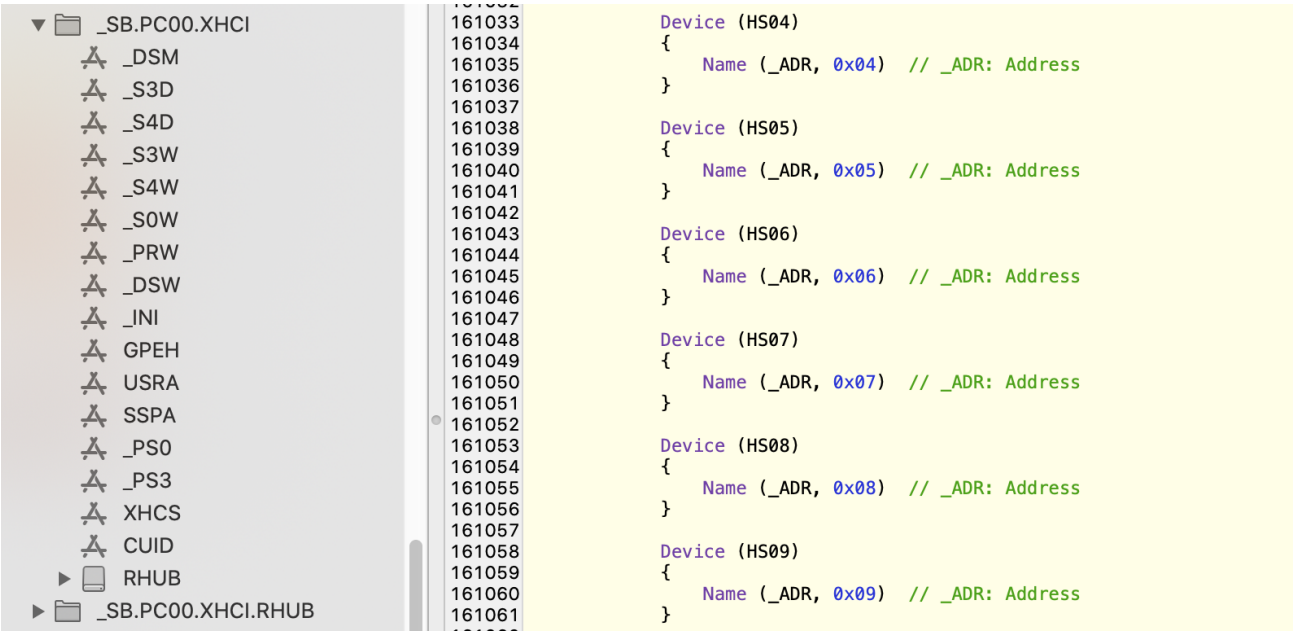

origin
Name
Änderungsdatum
Größe
Art
APIC.aml Gestern, 15:21 2 KB ACPI M...Binary
BGRT.aml Gestern, 15:21 56 Byte ACPI M...Binary
DBG2.aml Gestern, 15:21 84 Byte ACPI M...Binary
DBGP.aml Gestern, 15:21 52 Byte ACPI M...Binary
DMAR.aml Gestern, 15:21 264 Byte ACPI M...Binary
DSDT.aml Gestern, 15:21 767 KB ACPI M...Binary
DumpLog.txt Gestern, 15:21 12 KB Reiner Text
FACP.aml Gestern, 15:21 276 Byte ACPI M...Binary
FACS.aml Gestern, 15:21 64 Byte ACPI M...Binary
FIDT.aml Gestern, 15:21 156 Byte ACPI M...Binary
FPDT.aml Gestern, 15:21 68 Byte ACPI M...Binary
HPET.aml Gestern, 15:21 56 Byte ACPI M...Binary
LPIT.aml Gestern, 15:21 148 Byte ACPI M...Binary
MCFG.aml Gestern, 15:21 60 Byte ACPI M...Binary
MIGT.aml Gestern, 15:21 64 Byte ACPI M...Binary
MSCT.aml Gestern, 15:21 78 Byte ACPI M...Binary
NITR.aml Gestern, 15:21 113 Byte ACPI M...Binary
OEM1.aml Gestern, 15:21 44 KB ACPI M...Binary
OEM2.aml Gestern, 15:21 70 KB ACPI M...Binary
OEM4.aml Gestern, 15:21 171 KB ACPI M...Binary
PCAT.aml Gestern, 15:21 104 Byte ACPI M...Binary
PCCT.aml Gestern, 15:21 110 Byte ACPI M...Binary
RASF.aml Gestern, 15:21 48 Byte ACPI M...Binary
RSDP.aml Gestern, 15:21 36 Byte ACPI M...Binary
RSDT.aml Gestern, 15:21 148 Byte ACPI M...Binary
SLIT.aml Gestern, 15:21 108 Byte ACPI M...Binary
SRAT.aml Gestern, 15:21 3 KB ACPI M...Binary
SSDT-0-SSDT PM.aml Gestern, 15:21 54 KB ACPI M...Binary
SSDT-1-A M I.aml Gestern, 15:21 4 KB ACPI M...Binary
SSDT-2-PtidDevc.aml Gestern, 15:21 12 KB ACPI M...Binary
SVOS.aml Gestern, 15:21 50 Byte ACPI M...Binary
WDDT.aml Gestern, 15:21 64 Byte ACPI M...Binary
WSMT.aml Gestern, 15:21 40 Byte ACPI M...Binary
XSDT.aml Gestern, 15:21 260 Byte ACPI M...Binary
2 von 34 ausgewählt, 263,32 GB verfügbar

Die einzelnen Dateien lassen sich mit (dem unten verlinkten) „maciASL“ einsehen und bearbeiten. In der „DSDT“ findet man zumeist nur eine rudimentäre Darstellung von „XHCI“ (welches auch „XHC“ und ähnlich genannt sein kann). Hier ein erster Eintrag dazu, hier wurde das „Device“ angelegt und mit einer Adresse versehen.

```
Device (XHCI)
{
    Name (_ADR, 0x00140000)  // _ADR: Address
}
```

Auch im (ebenfalls unten verlinkten) „IORegistryExplorer“ wird das Device „XHCI“ mit der gleichen Adresse angezeigt, wie in der DSDT festgelegt: „XHCI@14“.

In der Folge in der DSDT wird „XHCI“ näher beschrieben, es werden Funktionen für Sleep und Wake nachgereicht und weitere Devices innerhalb von XHCI gebildet, wie „RHUB“ und die darin enthaltenen USB-Ports.



Am Beispiel eines X299 Systems, hier das „Asus Prime X299-Deluxe“, werden wir die vorhandenen USB-Ports näher beschreiben. Der Chipsatz X299 stellt 26 USB-Ports bereit, wie ebenfalls viele andere moderne Intel Chipsätze, somit sollte diese Anleitung

Allgemeingültigkeit besitzen. Alle diese Ports, ob nun physisch komplett vorhanden oder nicht, sind im „Grundgerüst“ der „DSDT“ angelegt. Diese lauten, wie wir schon teilweise auf den Screenshots gesehen haben:

HS01 ... HS14

USR1, USR2

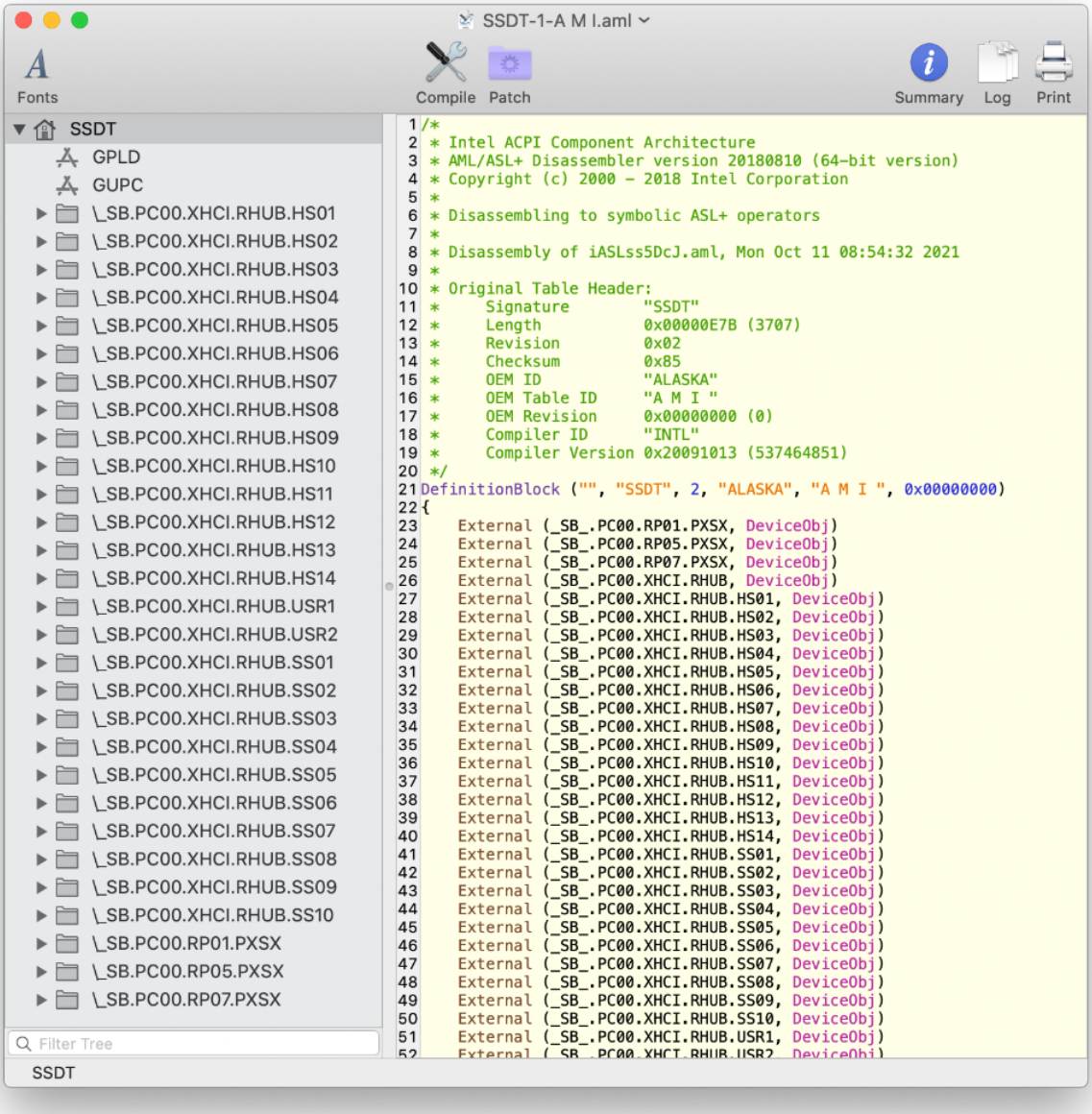
SS01 ... SS10

„HS“ steht für „High Speed“, 480mbit/s, USB2 mit abwärtskompatiblen USB1

„USR“ sind Platzhalter und werden nicht benutzt

„SS“ steht für „Super Speed“, 5gbit/s, USB3

Wie man hier in der „DSDT“ sieht, sind innerhalb der definierten 26 Ports keine weiteren Beschreibungen zu finden. Dazu schauen wir uns eine weitere „SSDT“ an. Fündig werden wir bei diesem System in der „SSDT-1-A M I“, hier sehen wir alle Ports von „XHCI“ näher beschrieben, zusätzlich noch drei USB3.1 von ASMedia, die uns vorerst nicht weiter interessieren.



Ganz interessant ist in der SSDT oben im „Header“ die Information „Length“. Diese wird hier in diesem Beispiel mit „0x00000E7B (3707)“ angegeben. Wir haben nun folgendes vor: wir werden diese SSDT kopieren, für unsere Zwecke ändern und ergänzen und per Bootloader zum richtigen Zeitpunkt laden lassen. Dazu müssen wir aber den Bootlader veranlassen, die originale SSDT nicht zu laden, ansonsten würde unsere SSDT nicht mehr geladen werden. Am Beispiel von „OpenCore“ setzen wir in der „config.plist“ unter „ACPI\Delete“ folgenden Eintrag:

Wurzel	Dictionary	8 Schlüssel/Wert-Paare
▼ ACPI	Dictionary	4 Schlüssel/Wert-Paare
▶ Add	Array	6 geordnete Elemente
▼ Delete	Array	2 geordnete Elemente
▼ 0	Dictionary	6 Schlüssel/Wert-Paare
All	Boolean	NO
Comment	String	Delete A M I
Enabled	Boolean	YES
OemTableId	Daten	0 Bytes:
TableLength	Zahl	3.707
TableSignature	Daten	4 Bytes: 53534454

Hier wird also unter „TableSignature“ mit dem Wert „53534454“ (Hexadezimal für den ASCII-String „SSDT“) und „TableLength“ mit dem von uns gefundenen Wert als Zahl „3707“ exakt unsere gewünschte SSDT ermittelt und „entfernt“ (nicht etwa aus dem BIOS gelöscht, sondern lediglich das Laden unterdrückt). Natürlich könnte man auch einfach über „OemTableID“ suchen lassen, aber hier schleichen sich gern mal Fehler ein. Gerade auch hier im Beispiel die „A M I “ hat noch ein Leerzeichen hinter dem „I“. Da ist die Angabe der Länge doch einfacher.

In der „config.plist“ über „ACPI\Add“ können wir unsere modifizierte SSDT, die wir selbstverständlich auch umbenennen können, wieder einfügen.

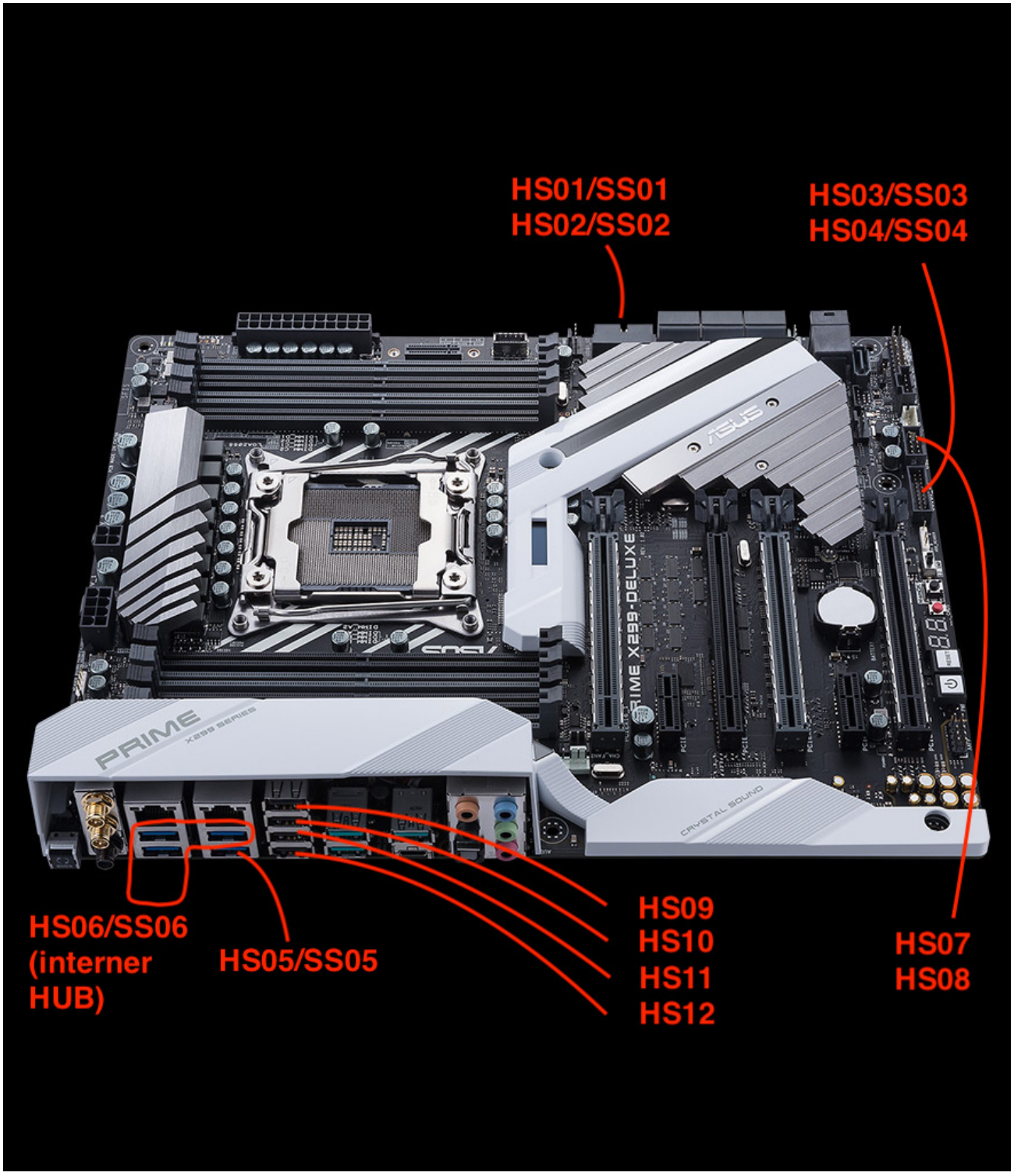
Schauen wir uns unsere SSDT nun genauer an. Wir finden hier Methoden für „USB Port Capabilities“ (Beschreibung von beispielsweise USB2, USB3, USB-C, intern …) und „Physical Location of Device“ (Beschreibung beispielsweise „hinten rechts“). Würden die Mainboardhersteller alles richtig machen, hätten wir hier kaum etwas zu tun, denn diese Standards(!) sind natürlich nicht Apple-spezifisch. Lediglich Apples „Port-Limit“ von 15 Ports je Controller ist für macOS spezifisch.

Grundsätzlich sind die beiden Methoden „_UPC“ und „_PLD“ für uns und macOS ausreichend, leider nur allgemein und nicht detailliert genug beschrieben. Hier setzen wir also an.

Wenn wir uns vorher schon Gedanken gemacht haben, welche unserer möglicherweise mehr als 15 vorhandenen Ports wir unter macOS verwenden wollen, um so besser. Ich möchte jetzt nicht noch erklären, wie wir herausfinden, welche namentlichen Ports sich hinter welchen physisch am Mainboard vorhandenen USB-Schnittstellen verbinden. Nur kurz soviel: An einer

USB3-Buchse laufen intern zwei Ports: ein USB2 und ein USB3 (HS, SS). Das sieht man auch direkt in der Buchse: auf der Zunge sind deutlich mehr Pins (oben, unten) als bei einer reinen USB2-Buchse. An den Pfostensteckern auf dem Mainboard (für USB-Ports am Gehäuse zum Beispiel), liegen ebenfalls mehr Ports an. Bei reinen USB2-Pfostensteckern werden je Stecker zwei USB2 übertragen, bei den USB3 sind es ebenfalls für zwei Buchsen, also inkl. Der jeweiligen USB2-Anteile hier schon vier Ports.

Schauen wir uns das am Beispiel vom „Asus Prime X299-Deluxe“ an:



Hier sehen wir an den beiden Pfostensteckern für USB3 liegen die ersten vier USB3 an (SS01 bis SS04 und die jeweils anteiligen USB2 mit HS01 bis HS04). Das sind schon mal acht Ports. Dann finden wir noch einen USB2-Pfostenstecker (HS07/HS08), das macht dann schon

insgesamt 10 Ports. Darüber hinaus finden wir hinten vier USB2 übereinander (HS09 bis HS12), das macht nun schon 14 USB Ports insgesamt. Schlussendlich finden wir noch mit SS05 und SS06 zwei weitere USB3-Ports, mit deren integrierten USB2-Ports (HS05 und HS06) wären das schon unzulässige 18 Ports. Und zusätzlich gibt es zwei „interne“ (also komplett intern verdrahtete) USB2-Ports, nämlich für das Board-eigene Bluetooth sowie für „AURA“ – die LED-Steuerung. Damit wären wir bei insgesamt 20 Ports, die dieses Board tatsächlich von den 26 möglichen zur Verfügung stellt. Zumindest für macOS werden wir uns also von fünf Ports trennen müssen. Übrigens: Wer genau aufgepasst hat, der wird etwas stutzen: Ich schrieb, hinten sind zwei weitere USB3 Ports. Zu sehen sind aber vier (mal von den grünen USB3.1 und USB-C von ASMedia abgesehen). Hier finden wir eine Besonderheit vom Board, dass die Ports HS06/SS06 erst intern auf einen Hub treffen, um dann auf drei Ports hinten weitergeleitet zu werden.

Merke: es reicht nicht nur die Anzahl der physischen Ports zu zählen um auf die tatsächliche Anzahl an benutzten Ports zu kommen, denn interne Hubs könnten diese schon drastisch reduzieren. Man sollte also immer über die genutzte Technik informiert sein. Wie finde ich nun heraus, welche derbeschriebenen Ports an der tatsächlichen Schnittstelle jeweils anliegen? USB2- und USB3-Gerät jeweils an allen Ports anhängen, in zum Beispiel „IORegistryExplorer“ sieht man dann, an welchen Port sich das jeweilige Gerät anhängt. Mehr nicht dazu, da gibt es auch schon etliche Abhandlungen darüber.

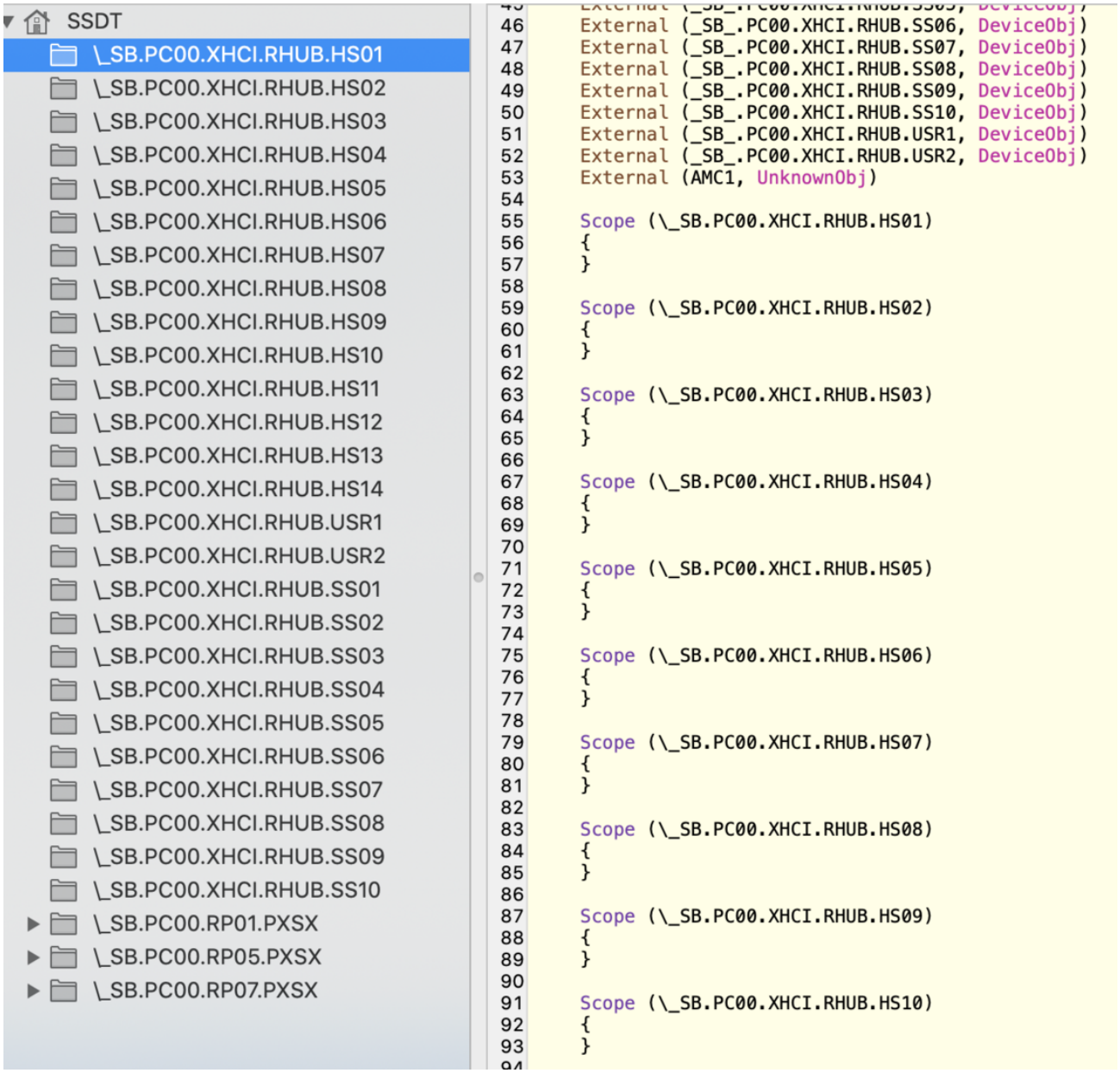
In meinem Fall hatte ich mich dazu entschlossen, je einen USB3 und USB2 Pfostenstecker nicht zu nutzen, da mein Gehäuse eh nur zwei USB3 anbietet. Somit fielen HS03/SS03 und HS04/SS04 sowie HS07/HS08 (vergleiche Bild) macOS zum Opfer und ich bin mit 14 Ports am XHCI im Limit von maximal 15 Ports.

Nun ändern wir unsere SSDT. Zuerst werfen wir die Methoden „GPLD“ und „GUPC“ raus, nebst „USSD“ und „UHSD“ (Name). Es schaut somit aus:

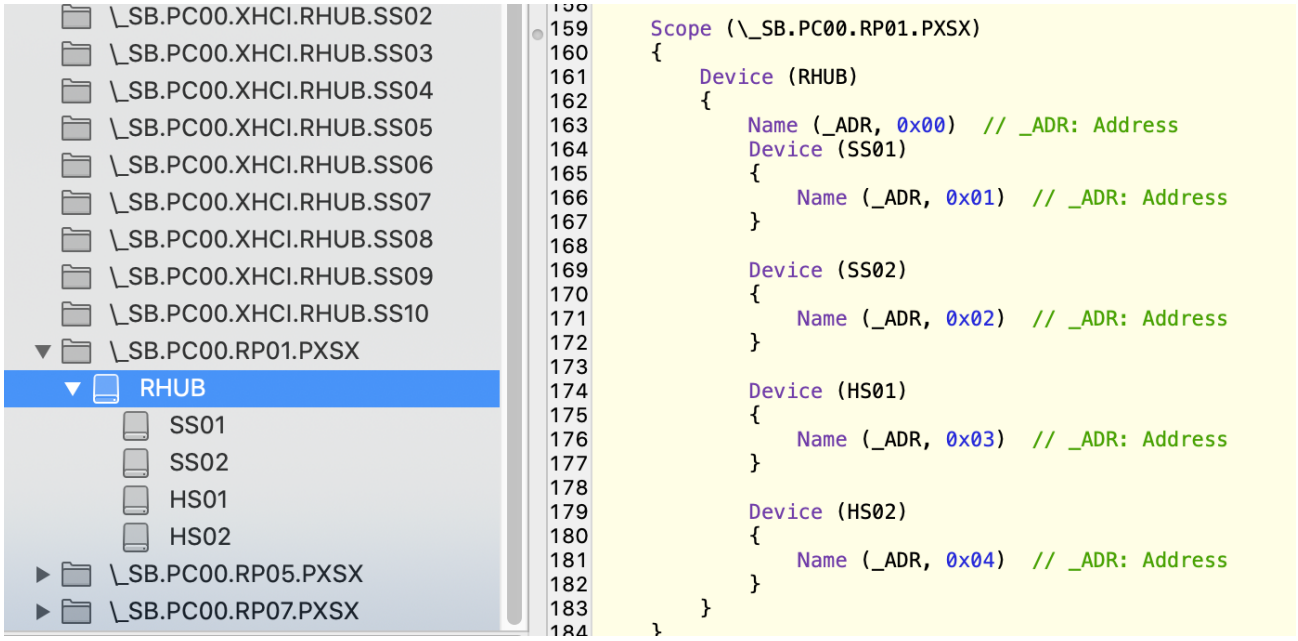
```
DefinitionBlock ("", "SSDT", 2, "ALASKA", "A M I ", 0x00000000)
{
    External (_SB_.PC00.RP01.PXSX, DeviceObj)
    External (_SB_.PC00.RP05.PXSX, DeviceObj)
    External (_SB_.PC00.RP07.PXSX, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.HS01, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.HS02, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.HS03, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.HS04, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.HS05, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.HS06, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.HS07, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.HS08, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.HS09, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.HS10, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.HS11, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.HS12, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.HS13, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.HS14, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.SS01, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.SS02, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.SS03, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.SS04, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.SS05, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.SS06, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.SS07, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.SS08, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.SS09, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.SS10, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.USR1, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.USR2, DeviceObj)
    External (AMC1, UnknownObj)

    Scope (\_SB.PC00.XHCI.RHUB.HS01)
    {
        Method (_UPC, 0, NotSerialized)  // _UPC: USB Port Capabilities
        {
            Return (GUPC (0x01))
        }

        Method (_PLD, 0, NotSerialized)  // _PLD: Physical Location of Device
        {
            Return (GPLD (DerefOf (UHSD [0x00]), 0x01))
        }
    }
```

Jetzt gibt es jede Menge Compiler-Warnungen, denn diese Methoden hatten ja Methode.

Nun entfernen wir auch die Einträge innerhalb aller Ports von XHCI.RHUB:



… und auch gleich noch aus den Ports der ASMedia. Hier lassen wir aber zumindest die Adressen drin, denn es sind neue Devices (erstmalig angelegt in der ACPI) und diese benötigen eine Adresse. Unsere bisherigen „Scope“ benötigen keine, da diese auf das jeweils gleichnamige „Device“ in der „DSDT“ referenzieren.



Nun ist die SSDT wieder fehlerfrei, allerdings sind auch damit noch keine weiteren Beschreibungen hinzugekommen. Das holen wir nun nach. In jeden Port (HS, SS) kopieren wir nun erstmal diesen Code, den wir im nächsten Durchgang dann noch individuell ändern:

Code

1. Name (_UPC, Package (0x04) // _UPC: USB Port Capabilities
2. {
3. 0xFF,
4. 0x03,
5. Zero,
6. Zero
7. })
8. Name (_PLD, Package (0x01) // _PLD: Physical Location of Device
9. {
10. ToPLD (
11. PLD_Revision = 0x1,
12. PLD_IgnoreColor = 0x1,
13. PLD_Red = 0x0,
14. PLD_Green = 0x0,
15. PLD_Blue = 0x0,
16. PLD_Width = 0x0,
17. PLD_Height = 0x0,

```
18. PLD_UserVisible = 0x1,
19. PLD_Dock = 0x0,
20. PLD_Lid = 0x0,
21. PLD_Panel = "UNKNOWN",
22. PLD_VerticalPosition = "UPPER",
23. PLD_HorizontalPosition = "LEFT",
24. PLD_Shape = "UNKNOWN",
25. PLD_GroupOrientation = 0x0,
26. PLD_GroupToken = 0x0,
27. PLD_GroupPosition = 0x0,
28. PLD_Bay = 0x0,
29. PLD_Ejectable = 0x0,
30. PLD_EjectRequired = 0x0,
31. PLD_CabinetNumber = 0x0,
32. PLD_CardCageNumber = 0x0,
33. PLD_Reference = 0x0,
34. PLD_Rotation = 0x0,
35. PLD_Order = 0x0,
36. PLD_VerticalOffset = 0x0,
37. PLD_HorizontalOffset = 0x0)
38.
39. })
```

Alles anzeigen

Hier haben wir sowohl "_UPC" wie auch "_PLD", die nicht nur macOS, sondern auch sämtliche anderen Systeme verstehen. Im Grunde waren die vorhandenen Einträge ähnlich, nur ungenau. Unter "_UPC" finden wir vier Einträge. Der erste sagt aus, ob dieser Port aktiv ist. "0xFF" steht für aktiv, "Zero" für nicht aktiv. Hiermit können wir also schon ganz genau steuern, ob dieser Port überhaupt genutzt werden soll. Wir lassen im ersten Durchgang diesen Wert auf aktiv. Aber wir können uns schon den zweiten Wert "0x03" anschauen und gegebenenfalls direkt ändern. Dieser Wert steht für die Art des Ports. Gültige Werte sind:

0x00 – USB2 (ausschliesslich unabhängige USB2 werden so deklariert)

0x03 – USB3 (auch zugehörige USB2, das heißt gleiche Buchse, werden so deklariert)

0x09 – USB-C (wenn unabhängig von der Drehung des USB-C-Steckers der _GLEICHE_ Port genutzt wird)

0x0A – USB-C (wenn je nach Drehrichtung des USB-C-Steckers ein weiterer Port genutzt wird)

0xFF – USB2 intern (zum Beispiel für Bluetooth)

Nun im zweiten Durchgang konzentrieren wir uns auf diejenigen Ports, die zwar in der SSDT aufgelistet vorhanden sind, aber physisch nicht am Mainboard. Diese bekommen als ersten und zweiten Wert "Zero" (oder "0x00", ist das gleiche). Wenn wir das geschafft haben, sind unsere Ports korrekt beschrieben, allerdings haben wir unser "Port Limit" noch nicht beachtet. Ich habe diese Ports (vorhanden, aber individuell wegen macOS gesperrt) absichtlich noch nicht weiter beachtet, da wir hier pfiffigerweise eine Abfrage nach "Darwin" durchführen. Nur bei Bestätigung durch macOS (Darwin – quelloffener Kern von OSX) werden diese Ports deaktiviert, unter Windows zum Beispiel bleiben die weiterhin offen. Wir fummeln in diesem Fall gar nicht weiter am ersten Wert von "_UPC" rum, sondern setzen eine zusätzliche Methode ein, nämlich "Status". Damit können wir grundsätzlich bestimmen, ob ein Device aktiv oder nicht sein soll, feinere dazwischenliegende Abstimmungen interessieren uns hierbei nicht:

Code

```
1. Method (_STA, 0, NotSerialized) // _STA: Status
2. {
3. If (_OSI ("Darwin"))
4. {
5. Return (Zero)
6. }
7. Else
8. {
9. Return (0x0F)
10. }
11. }
```

Alles anzeigen

Die Methode ist ganz simpel, bei Erkennung von macOS (Darwin) wird _STA "Zero" ausgegeben und das Device ist inaktiv, ansonsten voll aktiv mit den sonst geltenden Beschreibungen.

Das war jetzt letztendlich recht simpel, unten beigefügt sind sowohl originale wie auch modifizierte SSDT, so das jeder nachvollziehen kann, was sich geändert hat. Zusätzlich sind dann noch die drei USB3.1 ASMedia deklariert, auch hier schön zu sehen, dass es unterschiedlichste Konfigurationen gibt für die gleichen technischen Devices. Der erste an RP01 ist der interne USB-E (für Gehäuse USB-C). Hier wird per internem Kabel die Buchse auf

dem Mainboard mit dem USB-C am Gehäuse verbunden. Auffallend ist, dass hierbei die beiden SS01/SS02 (USB3.1-Anteil) komplett verdrahtet auf eine Buchse geschoben werden. Je nach Steckerdrehrichtung wird der eine oder andere Port genutzt. Entsprechend auch als "0x0A" deklariert. Während "HS01" (USB2-Anteil) immer auf der Buchse landet, egal wie der Stecker sitzt. Somit ist "HS02" zwar in der ACPI vorhanden, aber technisch nicht vorgesehen und somit auch per "Zero" deklariert.

An RP05 hängt der zweite ASMedia-Chip, der stellt zwei USB-A Buchsen (Grün) hinten am Mainboard bereit. Diese werden ganz normal als "USB3 deklariert, also "0x03".

Und zuletzt an RP07 der dritte ASMedia, hier auch hinten, allerdings als ein USB-A (Grün) und ein USB-C. Bei letzterem ist es egal, wie der Stecker angesteckt wird, es wird immer der gleiche Port benutzt. In diesem Fall also "0x09" eingestellt.

Ein letztes Special findet ihr unter HS13. Hieran hängt mein interner Bluetooth. Nach dieser Definition bleibt auch beim "DeepSleep" Bluetooth weiterhin aktiv, ich kann den Rechner per Maus oder Tastatur wecken und Bluetooth ist auch nach dem Wake selbstverständlich weiterhin aktiv. Dieses Problem hatte ich zuvor nicht, aber mit macOS Monterey war nach dem Sleep/Wake Bluetooth "tot". Das ist nun auch gefixt.

Viel Spaß beim anschauen der eigenen ACPI, und Verbessern der USB-Deklaration.