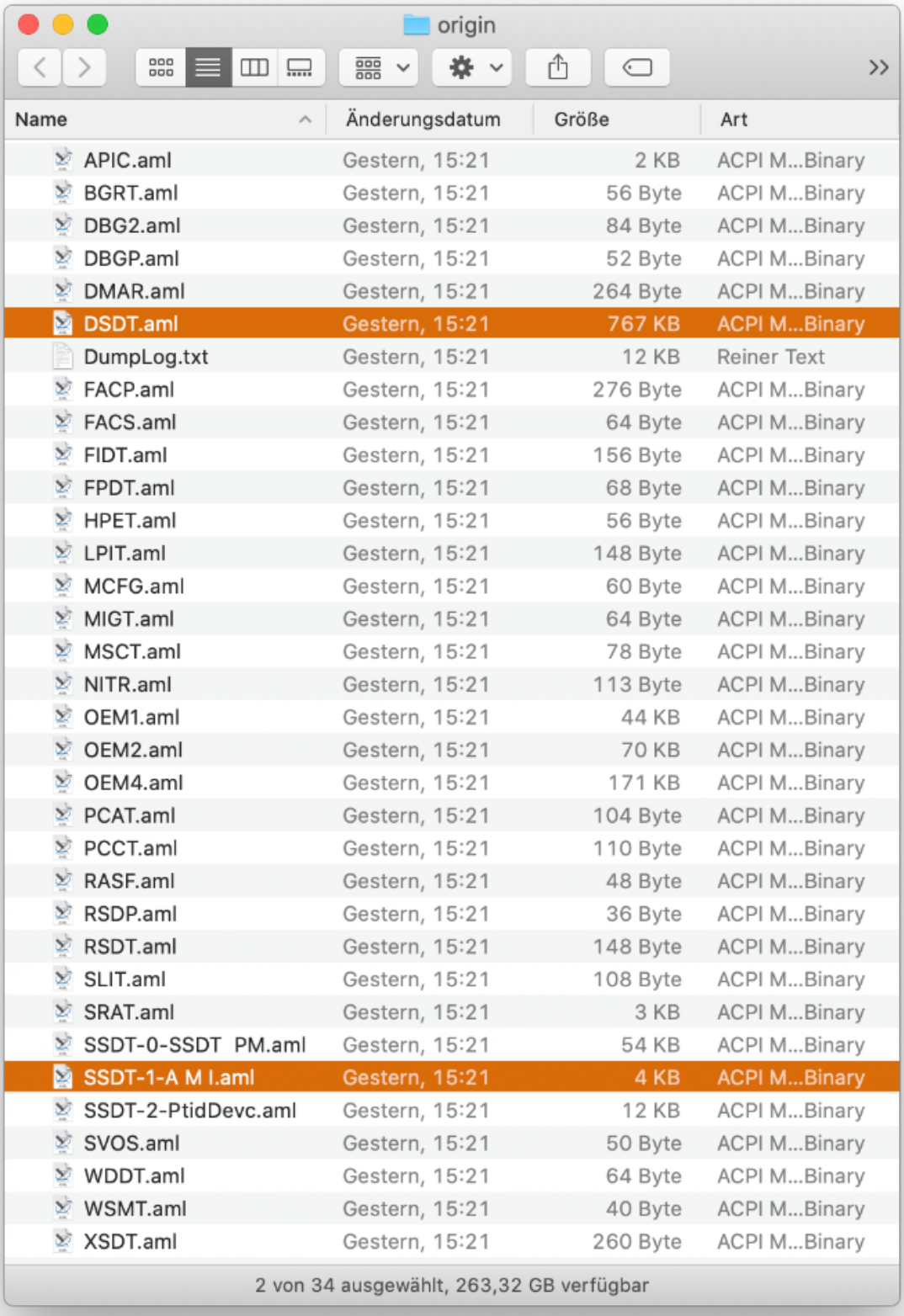
USB mittels SSDT deklarieren

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

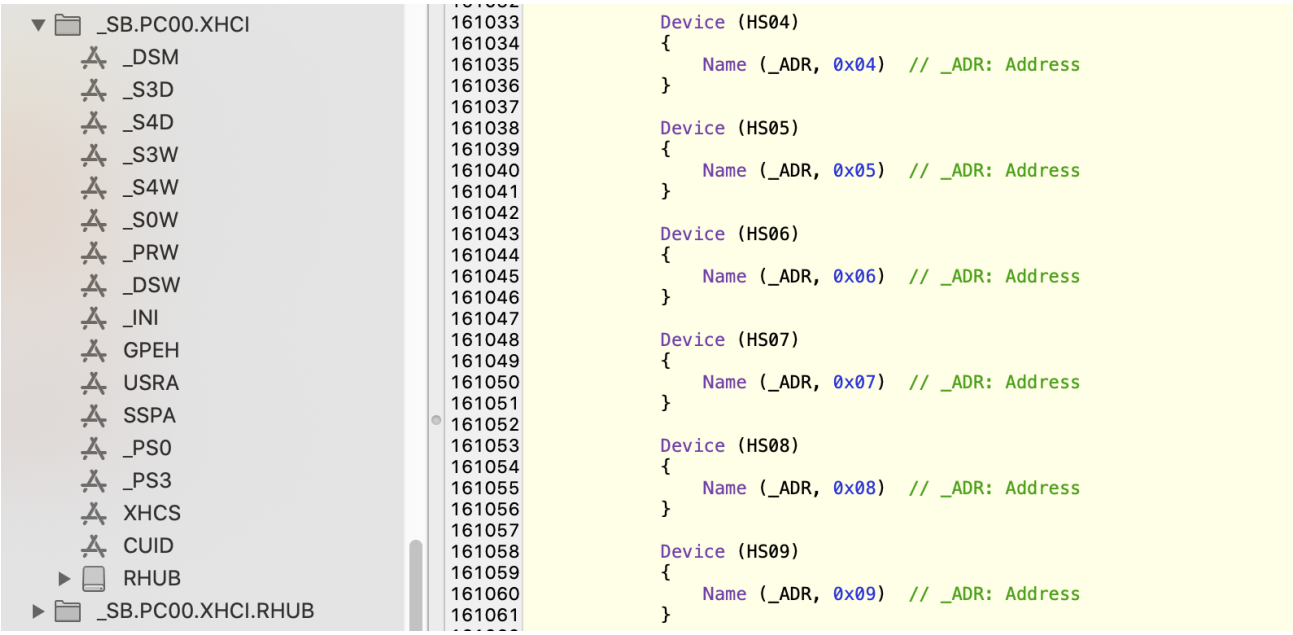

origin
Name
Änderungsdatum
Größe
Art
APIC.aml Gestern, 15:21 2 KB ACPI M...Binary
BGRT.aml Gestern, 15:21 56 Byte ACPI M...Binary
DBG2.aml Gestern, 15:21 84 Byte ACPI M...Binary
DBGP.aml Gestern, 15:21 52 Byte ACPI M...Binary
DMAR.aml Gestern, 15:21 264 Byte ACPI M...Binary
DSDT.aml Gestern, 15:21 767 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

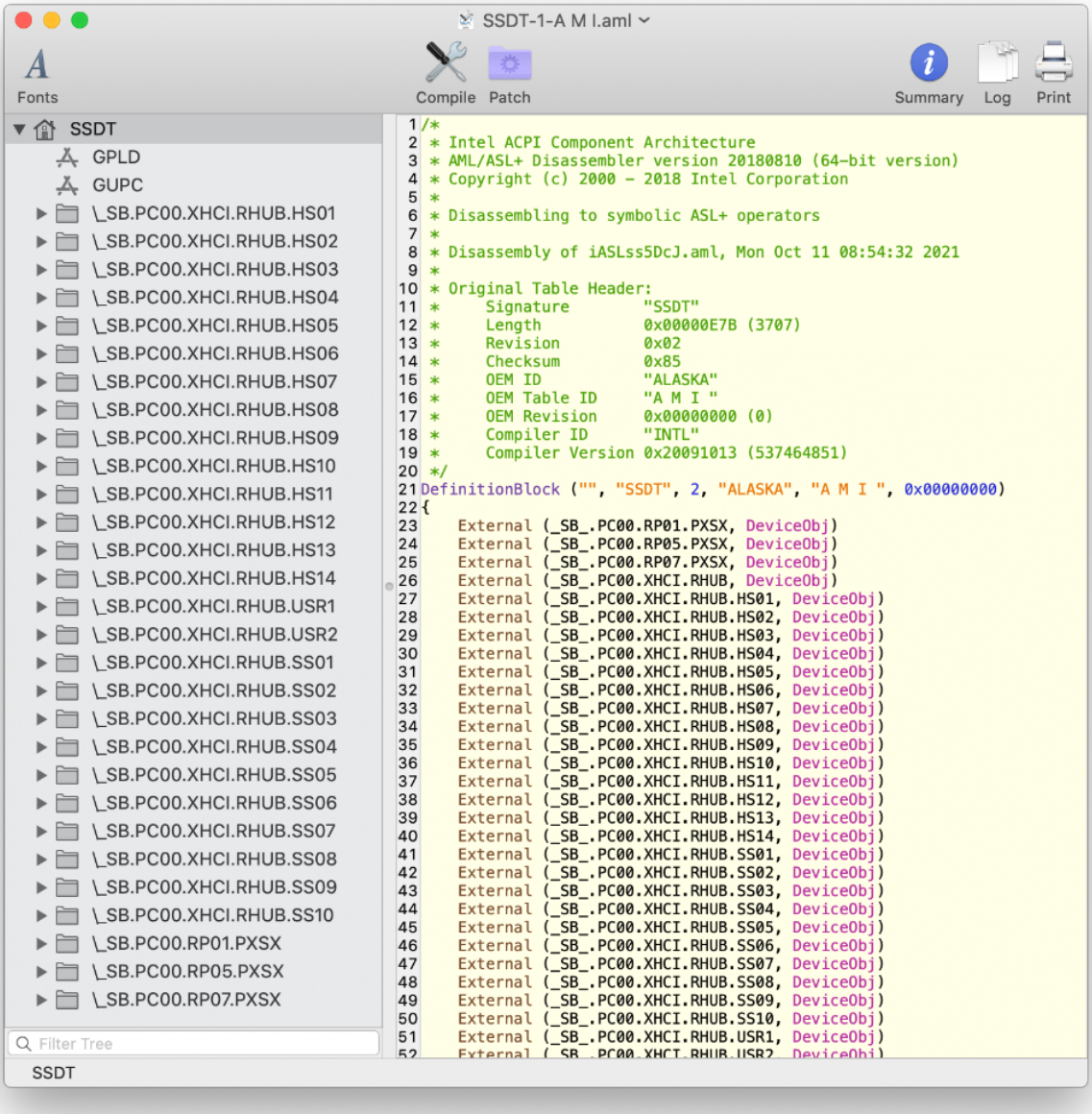
SS01 ... SS10

„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.

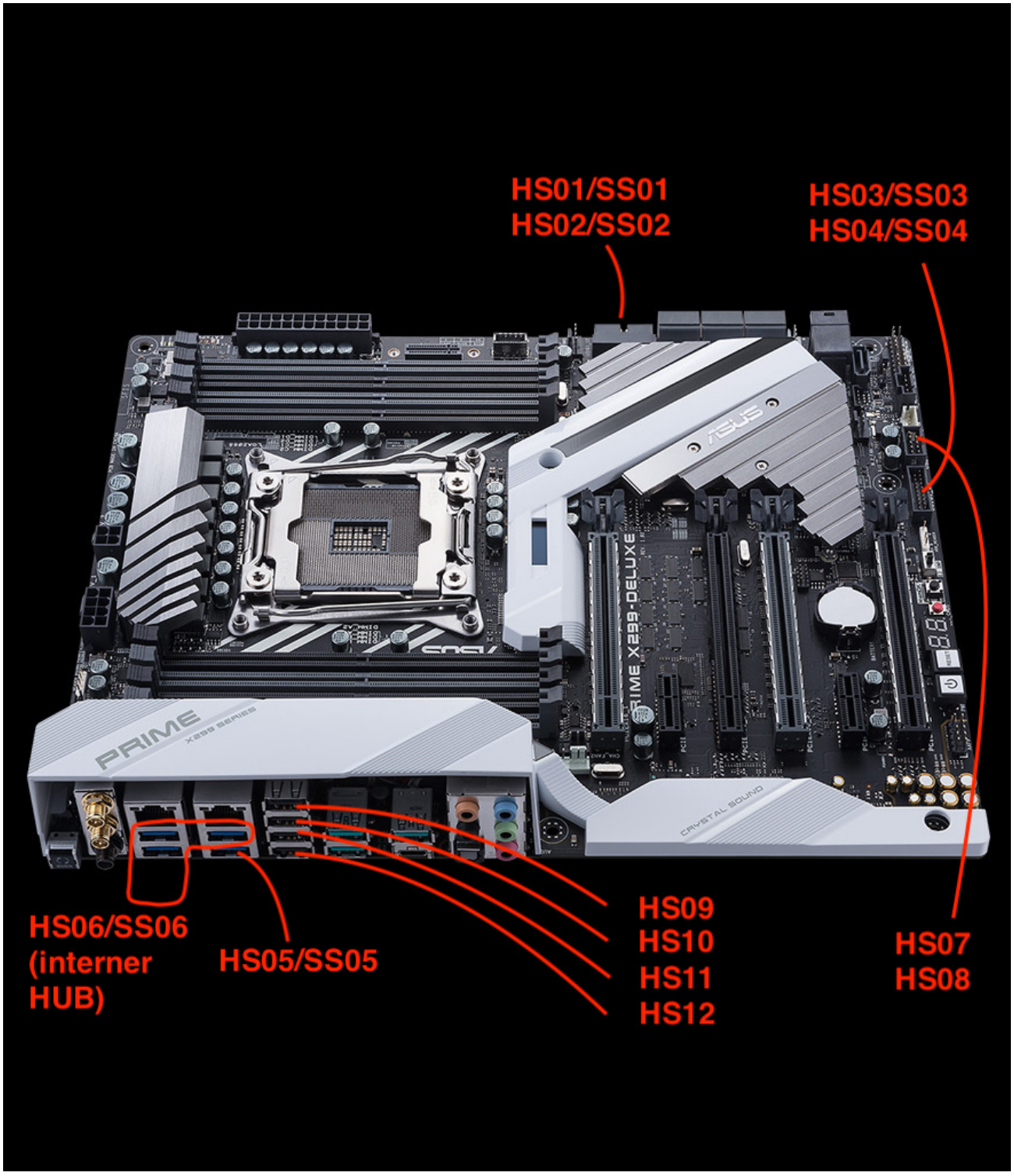
Schauen wir uns unsere SSDT nun genauer an. Wir finden hier Methoden für „USB Port Capabilities“ (Beschreibung von beispielsweise USB2, USB3, USB-C, intern …) und „Physical Location of Device“ (Beschreibung beispielsweise „hinten rechts“). Würden die Mainboardhersteller alles richtig machen, hätten wir 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.

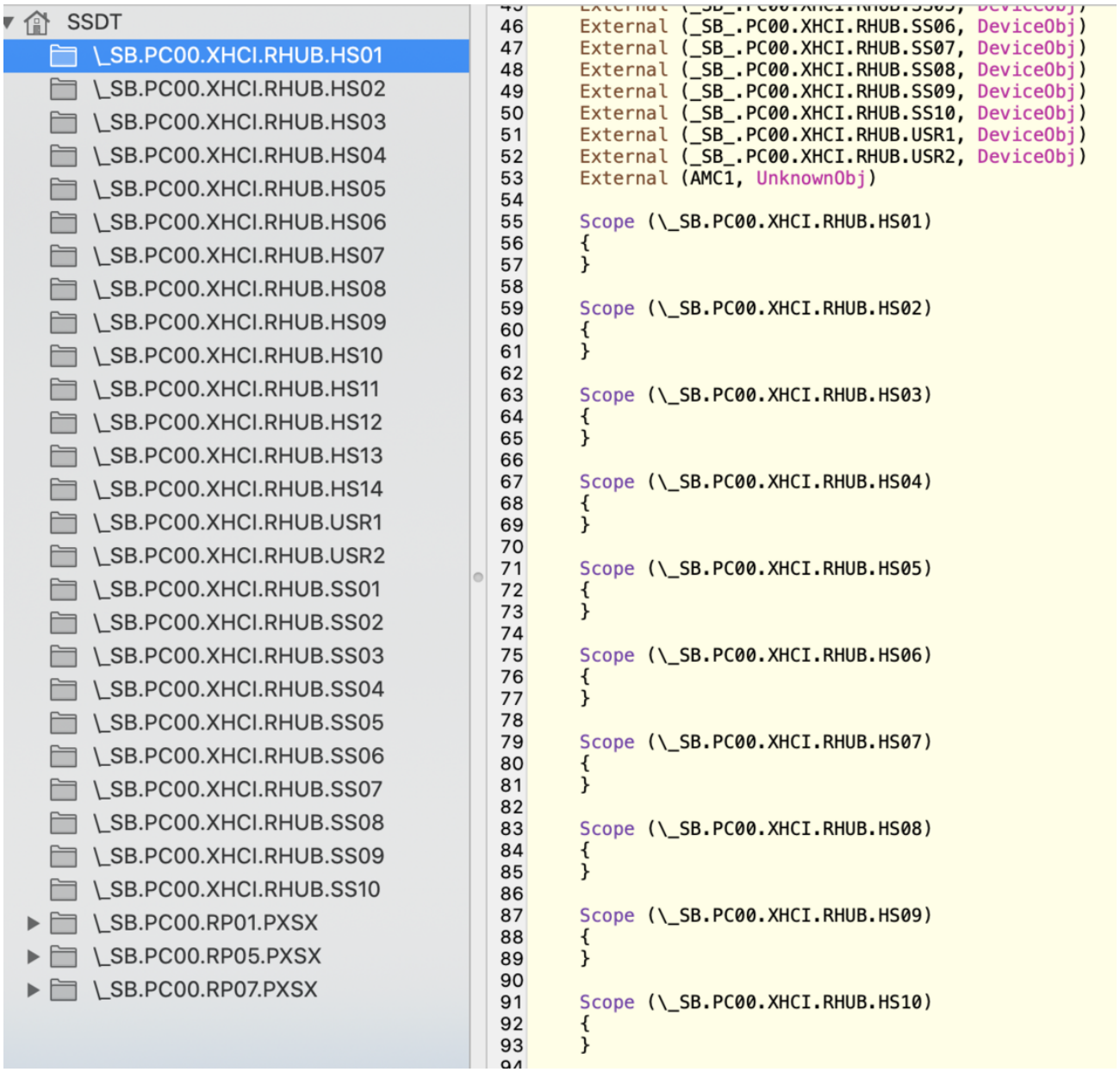
In meinem Fall hatte ich mich dazu entschlossen, je einen USB3 und USB2 Pfostenstecker nicht zu nutzen, da mein Gehäuse eh nur zwei USB3 anbietet. Somit fielen HS03/SS03 und HS04/SS04 sowie HS07/HS08 (vergleiche Bild) macOS zum Opfer und ich bin mit 14 Ports am XHCI im Limit von maximal 15 Ports.

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

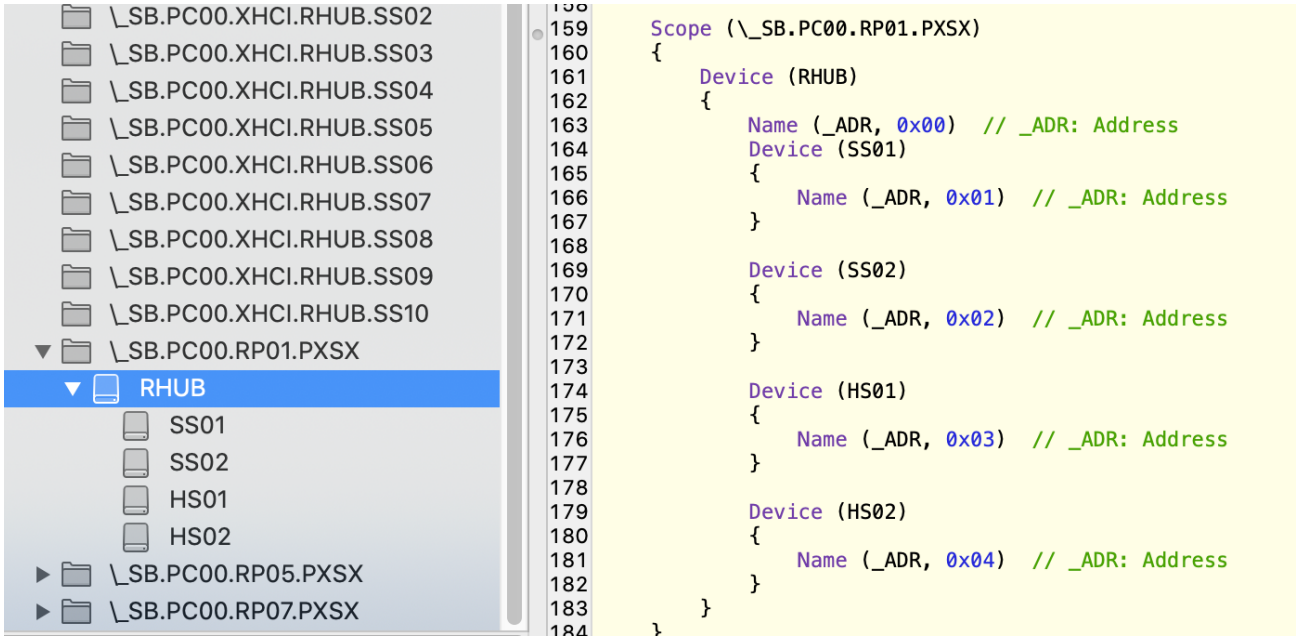
```
DefinitionBlock ("", "SSDT", 2, "ALASKA", "A M I ", 0x00000000)
{
    External (_SB_.PC00.RP01.PXSX, DeviceObj)
    External (_SB_.PC00.RP05.PXSX, DeviceObj)
    External (_SB_.PC00.RP07.PXSX, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB, DeviceObj)
    External (_SB_.PC00.XHCI.RHUB.HS01, DeviceObj)
    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.

Beitrag von „EdD1024“ vom 11. Oktober 2021, 14:24

Großartig - danke!

Man lernt nicht aus. Hatte letztens mit USB2 Probleme, jetzt weiß ich: falsch deklariert.

Beitrag von „N0b0dy“ vom 11. Oktober 2021, 17:57

Lieber Nico oder halt apfelnico

du hast hier alles gut beschrieben aber GPLD und GUPC müssen nicht entfernt werden besonders für die Leute, die gerne alle Ausgänge unter Windows oder Linux nutzen möchten, stattdessen kann man hier eine neue Methode schreiben und mittel _OSI jeden Ausgang definieren, was unter macOS machen soll.

Im Grunde genomen GUPC Methode macht nicht anders als was Name (_UPC, ...) unter macOS tut und genauso mit GPLD ist nicht anders als Name (_PLD, ...)

Hier kann man neue Methode definieren mit 2 veränderten Variablen, die so aussieht

Code

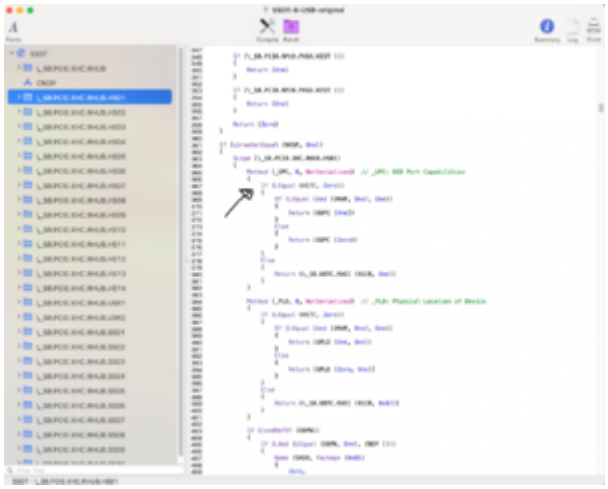
```
1. Method (MUPC, 2, Serialized)
2. {
3. Name (PCKG, Package (0x04)
4. {
5. Zero,
6. Zero,
7. Zero,
8. Zero
9. })
10. Store (Arg0, Index (PCKG, Zero))
11. Store (Arg1, Index (PCKG, One))
12. Return (PCKG)
13. }
```

Alles anzeigen

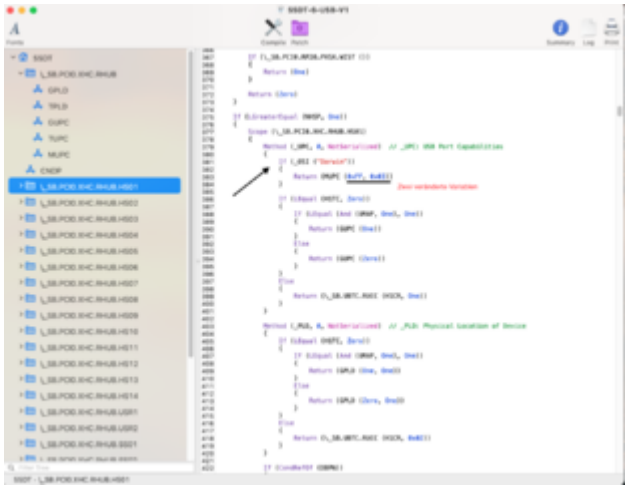
So könne wir hier der erste und zweite index von der Name (PCKG, ...) ändern

Danach definieren jeden Ausgang in der SSDT ohne was daran zu löschen, indem wir die Methode sagen, was sie zurückgeben soll

So sieht nicht geänderte SSDT



Und so sieht nachher



Wenn man den Ausgang unter macOS deaktivieren möchte dann einfach statt

Return (MUPC (0xff, 0x03))

nimmt

Return (MUPC (0x00, 0xff)) und so weiter....

So bleiben alle Ausgänge auch unter Windows oder Linux und nicht nur 15 Ausgänge wie unter macOS

Beitrag von „apfelnico“ vom 11. Oktober 2021, 20:24

N0b0dy

Ist aber exakt das, was ich sonst auch direkt gemacht habe. Auch beim ersten Beispiel sind ALLE Ports für Windows und andere Systeme vorhanden, nur eben sauber beschrieben. Per "_OSI" nur für macOS bestimmte Ports entfernt.

Habe mir das gerade angeschaut was du gepostet hast, so geht es auch. Ein Problem dabei bleibt: Die Ports sind nicht korrekt für "nicht-Darwin"-Systeme deklariert. Denn mit "GUPC" wird ein "universelles" Konstrukt für "_UPC" geliefert, wobei nur die erste Zeile ausgetauscht wird (aktiv oder nicht). Auch für Windows oder Linux ist es sicher nicht verkehrt, wenn man es richtig macht. Und zwar ACPI-konform, das hat nichts mit Apple zu tun.

Habe mal dein Vorschlag aufgegriffen und direkt "GUPC" verändert, so das zwei Parameter übergeben werden können:

Code

```
1. Method (GUPC, 2, Serialized)
2. {
3. Name (PCKG, Package (0x04)
4. {
5. 0xFF,
6. 0x03,
7. Zero,
8. Zero
9. })
10. PCKG [Zero] = Arg0
11. PCKG [One] = Arg1
12. Return (PCKG) /* \GUPC.PCKG */
13. }
```

Alles anzeigen

dann sieht das bei einem aktiven Port beispielsweise so aus:

Code

```
1. Scope (HS01)
2. {
3. Method (_UPC, 0, NotSerialized) // _UPC: USB Port Capabilities
4. {
5. Return (GUPC (0xFF, 0x03))
6. }
7.
8. Method (_PLD, 0, NotSerialized) // _PLD: Physical Location of Device
9. {
10. Return (GPLD (DerefOf (UHSD [Zero]), One))
11. }
12. }
```

Alles anzeigen

bei einem nicht vorhandenen:

Code

```
1. Scope (USR1)
2. {
3. Method (_UPC, 0, NotSerialized) // _UPC: USB Port Capabilities
4. {
5. Return (GUPC (Zero, Zero))
6. }
7.
8. Method (_PLD, 0, NotSerialized) // _PLD: Physical Location of Device
9. {
10. Return (GPLD (Zero, Zero))
11. }
12. }
```

Alles anzeigen

und bei einem, der nur in macOS deaktiviert sein soll:

Code

```
Scope (HS03)
{
Method (_UPC, 0, NotSerialized) // _UPC: USB Port Capabilities
{
If (_OSI ("Darwin"))
{
Return (GUPC (Zero, Zero))
}
Else
{
Return (GUPC (0xFF, 0x03))
}
}

Method (_PLD, 0, NotSerialized) // _PLD: Physical Location of Device
{
If (_OSI ("Darwin"))
{
Return (GPLD (Zero, Zero))
}
Else
{
Return (GPLD (DerefOf (UHSD [0x02]), 0x03))
}
}
}
```

Alles anzeigen

Anbei eine überarbeitete SSDT mit diesen Änderungen.

Beitrag von „5T33Z0“ vom 12. Oktober 2021, 11:57

Habe mich gerade mal daran versucht, aber in meiner orginalen SSDT sind unter _SB.PCI0.XHC.RHUB noch andere Methoden: GPLD, TPLD, GUPC und TUPC und insgesamt ist da noch viel mehr anderer Kram drin neben den 26 Ports. Da weiß ich jetzt nicht, wie ich weitermachen soll.

SSDT-7-xh_cmsd4.aml (Ist von nem Gigabyte Z490 Vision G, Bios F21a)

Beitrag von „apfelnico“ vom 12. Oktober 2021, 12:44

ST33Z0

TUPC liegt da zwar, wird aber nicht benutzt. Aber eventuell von einer anderen SSDT darauf zugegriffen. Innerhalb dieser SSDT spielt diese Methode keine Rolle. Die ersten beiden Methoden brauchst du nicht anzufassen, GUPC könnte, wie ich in Post #4 schrieb, geändert werden. Sobald du diese Methode änderst, _MÜSSEN_ bei jedem Port zwei Angaben (Arg0, Arg1) gemacht werden, schaue dir dazu weiter den Post #4 an. Du kannst bei den Ports innerhalb von "_UPC" (also innerhalb der geschweiften Klammer) alles rausnehmen und dich dafür an dem halten, wie es in Post #4 in der beigefügten SSDT steht. Die Werte sollten klar sein, hatte ich ganz oben geschrieben, Beispiel für _OSI-Abfrage siehst du ebenfalls in der SSDT.

Bei dir steht etwas mehr in der SSDT drin, bei den Ports gehts es um eine Abfrage, ob der überhaupt per BIOS aktiv geschaltet ist (Variable dazu auslesen) und wenn ja, wie dann zu verfahren ist. Das kann man ja entfernen. Ansonsten ist da noch etliches zu den RP geschrieben, ob aktiv und wenn ja, was dann ... den Rest kannst du völlig unbeeindruckt drin lassen, hat ja seine Aufgaben.

Beitrag von „N0b0dy“ vom 12. Oktober 2021, 13:00

Ja klar kannst du auch GUPC ändern statt eine neue hinzufügen aber dann musst du hier vieles dabei ändern und für die meisten werden nicht mitbekommen, was wurde gemacht oder geändert.

Für die meisten wissen nicht mal wie USBMap.kext erstellen müssen, wie denn das damit, kannst du vergessen

Es gibt noch dritte Möglichkeit, wo du den Device RHUB unter macOS unterdrückst/deaktivierst und einen neuen Device definierest (XHUB oder ZHUB....) und wie du oben am Anfang "Post Nr. 1" alles geklärt hast, die Ausgänge für MacOS deklarierst , da bleibt RHUB unversehrt für andere Systeme und die original SSDT muss nicht in OpenCore deaktiviert

ST33Z0 wenn du uns mitteilst, welche USBs unter macOS behalten möchtest dann könnte deine SSDT bearbeiten und so damit wäre ein Beispiel für die anderen

Screenshot von Hackintool würde reichen

Beitrag von „apfelnico“ vom 12. Oktober 2021, 13:16

N0b0dy

Das ist nichts für Einsteiger, völlig klar. Und ich möchte eben die Ports korrekt beschrieben haben. Etwas, was die Hersteller versäumt haben. Das hat zunächst NICHTS mit macOS zu tun. Lediglich für macOS kommen noch bestimmte Einträge hinzu.

ST33Z0

habe mal deine SSDT überarbeitet und liegt anbei.

Du siehst, die Methode "GUPC" wurde verändert und jeder Port hat einen dazu passenden Eintrag. Die Werte sind bei allen Ports zunächst gleich (aktiv, usb3). Das musst du noch ändern, das weiß ich natürlich nicht.

Wie ein fehlender Port (also wirklich physisch vom Hersteller auf diesem Board nicht bestückt, gegenüber den 26 möglichen Ports des Controllers) aussieht, siehst du unter "USR1/2". Das könntest du für die entsprechenden Ports auch so festlegen.

Wie ein Port aussieht, den du nur unter macOS deaktivieren möchtest wegen des PortLimits,

siehst du beispielhaft unter "HS01".

Viel Erfolg!

Beitrag von „G.com“ vom 12. Oktober 2021, 13:32

Wo genau liegt denn der Vorteil zur Kext Methode?

Beitrag von „5T33Z0“ vom 12. Oktober 2021, 13:33

apfelnico Geil, vielen Dank. Dann kann ich die Datei schon mal als Template benutzen und entsprechend anpassen.

Beitrag von „apfelnico“ vom 12. Oktober 2021, 13:58

> Zitat von G.com
>
> Wo genau liegt denn der Vorteil zur Kext Methode?

Wenn sonst alles läuft, möchte man ja etwas zu tun haben.

Kext ist prima. Aber, diese selbsterstellte USB-Kext ist ja keine ausführbare Kext, sondern übermittelt lediglich fehlerbereinigte und weitere Beschreibungen über USB an die eigentliche Kext. Etwas, was nicht notwendig wäre, würde die ACPI schon korrekt sein. Also mehrere Stufen darunter. Und wer das reparieren möchte und den nötigen Ehrgeiz mitbringt, der ist herzlich eingeladen hier mitzumachen. Ein tatsächlicher Vorteil ist dann auch gegeben, da nun alles ab Basis stimmt. Diese selbstgebauten "HelperKexte" funktionieren natürlich auch nur solange, wie eben diese Mechanismen verfügbar sind. Und auch da hat sich in den letzten macOS-Versionen einiges geändert, plötzlich funktionierten Kexte nicht mehr, die von

"Hackintool" erstellt wurden, weil es nun andere Schnittstellen gab. Mit der SSDT-Methode hingegen sind die Ports sauber beschrieben schon auf ACPI-Ebene. Das muss hier keiner machen, ein Kext funktioniert auch. Es war nur mal 'ne Frage danach und ich habs mal versucht aufzudröseln.

Beitrag von „5T33Z0“ vom 12. Oktober 2021, 14:00

> Zitat von G.com
>
> Wo genau liegt denn der Vorteil zur Kext Methode?

USB Ports, die via ACPI Tabelle definiert werden, sind betriebssystem-agnostisch, wie es so schön heißt. Das heißt, sie funktionieren dann mit jedem OS. Wenn man aber eine USBPorts.kext für Catalina gebastelt hat für iMac20,2, dann funzt die Kext nicht, wenn man auf iMac19,1 wechselt weil man zum Beispiel macOS Mojave nutzen will. Dann muss man da wieder dran rumfummeln.

Zudem ist es seit macOS seit Big Sur 11.4(?) nicht mehr so einfach, die Ports zu mappen, da der XHCI port limit quirk nicht mehr funktioniert. Den ganzen Stress kann man halt damit umgehen, dass die Ports via ACPI gemapt werden. Das macht man einmal und hat dann Ruhe bis sich etwas am BIOS ändert. Dann muss man eventuell die Angaben der tabelle, die gedroppt werden soll in der Config anpassen (falls sich das Original geändert hat)

EDIT: Habe jetzt ein bisschen Knoten im Kopf vom Gefrickel. Bis auf die beiden Front Panel USB 3 Ports (HS09/SS09) scheint alles soweit zu funktionieren. Die SSDT wird auch geladen (kann sie unter maciASL über "File > New from ACPI" öffnen). Könnte sich das vielleicht jemand ansehen? Weiß nicht, wo da der Fehler ist. Anbei die SSDT und ein PDF mit Auslistung der Ports und Mainboard-Skizze. Thx

---

Beitrag von „G.com“ vom 12. Oktober 2021, 18:03

apfelnico ST33Z0 Danke Euch beiden. Da ich aus verschiedenen Gründen bis auf Weiteres bei Catalina und meiner Hardware bleibe erst einmal nicht relevant, aber nach Hardwarewechsel

ganz sicher ein Teil des Projektes. Vote für Aufnahme der Thematik ins Dortania.

Beitrag von „N0b0dy“ vom 12. Oktober 2021, 18:58

hier ist meine Variante, du kannst sie testen und berichten...

Viel Erfolg

ST33Z0

Beitrag von „5T33Z0“ vom 12. Oktober 2021, 19:36

> Zitat von N0b0dy
>
> hier ist meine Variante, du kannst sie testen und berichten...
>
> Viel Erfolg
>
> ST33Z0

Vielen Dank für die Mühe. Die Front Panel USB 3 (SS09/SS10) funzen leider immer nocht nicht. Werde morgen mal rein gucken, wo das Kabel zum Front Panel dran hängt. Vielleicht liegt's ja daran. Mit meinem USBPorts Kext funktionierts komischerweise.

Beitrag von „N0b0dy“ vom 12. Oktober 2021, 19:57

lade bitte hier oder als PN eine gespeicherte IOReg File mit USBPorts Kext und eine mit meiner SSDT hoch dann sehe es genau an

ST33Z0

Beitrag von „apfelnico“ vom 12. Oktober 2021, 20:06

ST33Z0

Alle durchdekliniert laut deiner Zeichnung, 15 Ports für macOS:

Beitrag von „5T33Z0“ vom 12. Oktober 2021, 20:49

apfelnico Vielen Dank. Leider funktioniere SS09 und SS10 damit auch nicht. Ein USB 2 Stick wird allerdings erkannt.

Ich habe festgestellt, dass es noch eine "SSDT-5-A M I.aml" gibt, in der RHUB und HS10 und HS14 vorkommen. Weiß nicht, ob das von Bedeutung ist.

N0b0dy IOReg für beide Fälle anbei.

Beitrag von „N0b0dy“ vom 12. Oktober 2021, 20:59

IOReg mit ACPI deutet darauf, dass deine geänderte SSDT nicht richtig geladen, du solltest nachprüfen, ob wirklich original SSDT in OpenCore deaktiviert ist

ST33Z0

Beitrag von „cobanramo“ vom 12. Oktober 2021, 21:04

Kann man mir bei meinem Z590 Pro auch weiterhelfen?🤔

Ich würd das ganze auch mal mit Acpi austesten.
Ich schnalls anscheinend doch noch nicht so recht...

anscheinend muss ich ja den "SSDT-5-xh_rksu4.aml" patchen,
Da gibt es noch ne USB-C Tabelle die verstehe ich jetzt überhaupt nicht...

Anbei hab ich den Kext Info.plist den ich zurzeit einsetze..

Gruss Coban

Beitrag von „N0b0dy“ vom 12. Oktober 2021, 21:18

cobanramo

dein Info.plist lässt sich nicht herunterladen

Beitrag von „cobanramo“ vom 12. Oktober 2021, 21:19

hmm interessant, ich kann laden..

Hab mal auch als Zip hinterlegt.

Gruss Coban

Beitrag von „5T33Z0“ vom 12. Oktober 2021, 21:24

apfelnico N0b0dy

YAY! Nachdem ich dann die OEMTableID als HEX eingetragen habe, ging es. Meine ACPI Datei funzt, die von N0b0dy funzt auch. Muss dann die Tage mal alle gemappten Ports einzeln testen. Nehme jetzt erstmal die von N0b0dy, weil ich meinen eigenen ASL "skills" nicht so recht über den Weg traue Aber nach knapp nem halben Jahr beschäftigen ASL Gefrickel am Laptop, erkenne ich langsam schemenhaft Strukturen.

Vielen Dank für eure Hilfe!!!

Beitrag von „N0b0dy“ vom 12. Oktober 2021, 21:45

erstmal sehe ich die Ports so richtig deklariert bei diesem Mainboard und so einfach ist bei dir 👍

Im Anhang ist deine SSDT musst du nur in OC einbinden und die original deaktivieren

Viel Erfolg

Merkung: was steckt bei dir in HS13 und HS14, ist die WLAN BCM... oder normal USB2.0?

Weil in original SSDT sind als USB2.0 deklariert aber du hast sie in Info.plist als Intern !!!

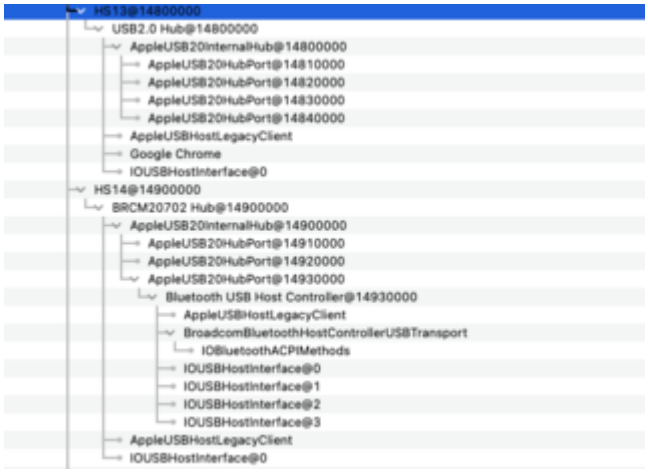
cobanramo

Beitrag von „cobanramo“ vom 12. Oktober 2021, 21:54

Danke dir, ich habs gern unkompliziert und klar 😉

An der HS13 taucht bei mir mit den tools immer eine interne Hub auf, daher hatte ich den mal als intern gesetzt.

An der HS14 ist die BRCM20702 Bluetooth Modul, also auch Intern.



Die original SSdt verursacht auf meinem Z590 beim starten eine acpi Fehler, daher hatte ich den sowieso deaktivert/unterdrückt gehabt.

Ich teste mal dein SSDT und berichte, danke nochmals im voraus.

Gruss Coban

Beitrag von „N0b0dy“ vom 12. Oktober 2021, 21:58

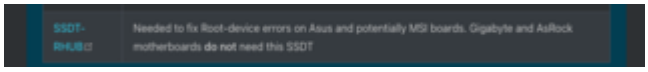
moment mal wenn du RHUB deaktiviert dann wird bei dir nicht klappen aber testen kannst du schon mal

Dann bei dir ist am besten die dritte Variante, die ich in Post Nr. #7 geschrieben habe

---

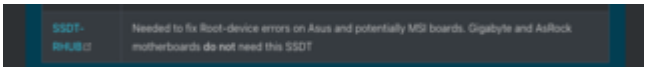
Beitrag von „cobanramo“ vom 12. Oktober 2021, 22:03

ich setze kein SSDT-RHUB wenn du dies meinst.



Bei mir macht es kein unterschied mit oder ohne diesen Ssdt, der Kext funktioniert perfekt.

Beitrag von „N0b0dy“ vom 12. Oktober 2021, 22:20

> Zitat von cobanramo
>
> ich setze kein SSDT-RHUB wenn du dies meinst.
>
> 
>
>
> Bei mir macht es kein unterschied mit oder ohne diesen Ssdt, der Kext funktioniert perfekt.

Es ist das selbe, ob du SSDT-RHUB nutzt oder die SSDT unterdrückst , beide geben das selbe Ziel

EDIT: Schick mir deine EFI und gespeicherte IOReg File ohne USB_Kext dann bastele für dich SSDT für USB

Beitrag von „cobanramo“ vom 12. Oktober 2021, 22:49

Jep, mit dem SSDT hab kein funktionierenden USB mehr gehabt, ich muss kurz über Umweg um rückgängig zu machen.

Einen kleinen moment bitte, Ioreg mit Kext und ohne kext plus das Efi kommt gleich.

Beitrag von „cobanramo“ vom 12. Oktober 2021, 23:11

> Zitat von N0b0dy
>
> ob du SSDT-RHUB nutzt oder die SSDT unterdrückst

Das hatte ich jetzt nicht so verstanden, gut zu wissen das es den gleichen efekt hat.
Die files hab ich dir gesendet.

Das musste ich grad mal über remote erledigen, ohne kext hatte ich jetzt kein funktionierende Tastatur und Maus

---

Beitrag von „NickRandom“ vom 12. Oktober 2021, 23:23

Hi,

könnt ihr vielleicht mal in diesen Fred Brauche Hilfe bei USB2 Initialisierung reingucken und mich mit der Nase draufstoßen, wie ich eure, wie ich finde, Supermethode zur Lösung meines USB2-Problems adaptieren könnte? Vielen Dank schon mal im voraus.

Beitrag von „5T33Z0“ vom 13. Oktober 2021, 10:30

Bei den "Drop" Table Regeln, denen ich bisher begegnet bin war zusätzlich auch die Funktion "ALL" aktiviert. Was ist der Unterschied zwischen an und aus?

Beitrag von „cobanramo“ vom 13. Oktober 2021, 11:56

So wie ich das verstehe wird bei "All=Enable" Alle erkannten Tabellen gelöscht, sonst halt nur der erste der erkannt wird.

1. **All**
   Type: `plist boolean`
   Failsafe: `false` (Only delete the first matched table)
   Description: Set to `true` to delete all ACPI tables matching the condition.

Gruss Coban

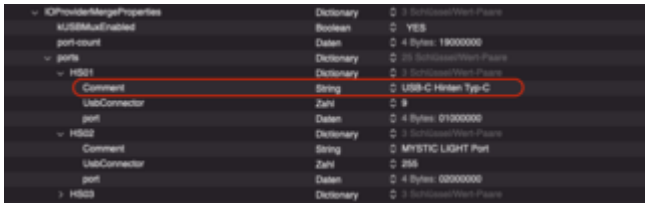
Beitrag von „cobanramo“ vom 13. Oktober 2021, 12:18

N0b0dy
Ich danke für deine Unterstützung, jetzt hab ich mal ausführlich getestet und kann bestätigen das es dem Kext genau entspricht.

Alle Ports funktionieren so wie Sie es sollten.
Ich häng mal den Ergebnis hier rein, es hilft evtl. ja dem einem oder anderen bei ähnlicher Board.

Allerdings hätte ich noch ne frage, kann man so wie bei der Kext lösung die Ports individuell benamsen?

Wenn ja hättet Ihr ne beispiel wie man dies erreicht?



Beitrag von „N0b0dy“ vom 13. Oktober 2021, 13:29

Ich glaube, das kannst mittel _DSM Methode machen dann siehst du es unter IOReg oder meinst du wo anders.

Ich habe das bis jetzt noch nicht gemacht aber werde es bei mir testen und berichten.

cobanramo

Beitrag von „5T33Z0“ vom 13. Oktober 2021, 14:30

Ergänzung zu "USRx" Ports:

"These ports are known as "USBR" ports, or more specifically USB Redirection Ports. Use of these is for remote management but real Macs don't ship with USBR devices and so has no support for them OS-wise. You can actually ignore these entries in your USB map."

Ich habe einen USB C port an meinem rechner. Ich nutze den zwar nicht, aber mich würde interessieren, was die Bezeichnung bedeutet: "USB 3.2 Type-C+Sw." Sw = switch? Was wird denn da geswitcht?

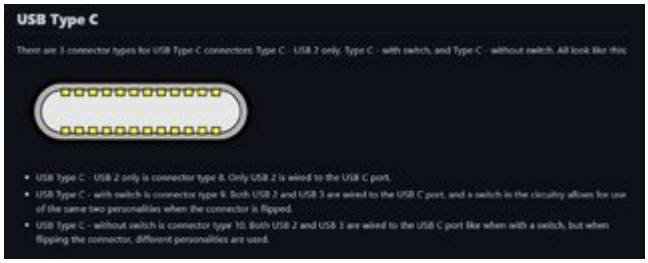
Beitrag von „Harper Lewis“ vom 13. Oktober 2021, 16:32

Wenn ich es richtig verstanden habe, ändert sich bei Ports ohne Switch auch der USB2-Port, wenn man die Orientierung des Steckes ändert. Müsste man mal ausprobieren…

Beitrag von „LetsGo“ vom 13. Oktober 2021, 16:45

ST33Z0

Wenn man den USB-C Stick 2-mal in den Port steckt (also den Stick beim zweiten mal umdreht) und immer der selbe Port aufleuchtet hat der Port einen Switch connector-type = 9. Leuchtet nach dem Umdrehen des Sticks ein anderer Port auf, wäre dieser connector-type =10.



Und ja, Sw steht für switch.

Beitrag von „maschinenwart“ vom 13. Oktober 2021, 22:26

> Zitat von apfelnico
>
> ST33Z0
>
> ...TUPC liegt da zwar, wird aber nicht benutzt. Aber eventuell von einer anderen SSDT darauf zugegriffen. Innerhalb dieser SSDT spielt diese Methode keine Rolle...

Hallo apfelnico

danke für wieder mal eine hervorragende Beschreibung!

Ich habe da mal eine Bitte, könntest du dir eventuell die SSDT / DSDT von meinem GA-Z390 D Board anschauen?

(SSDT-6-AMI) Die Ports HS08 und HS13 verweisen auf die Methoden TUPC / TPLD. Diese beiden Ports sind die USB2 Ports des Onboard Thunderbolt Controllers. Wie würde man bei diesenPorts vorgehen, um sie richtig einzubinden oder zu deaktivieren?

Beste Grüße

mw

Beitrag von „5T33Z0“ vom 13. Oktober 2021, 22:50

Könnte mir jemand diese Konstruktion erklären?

Ich verstehe das so: die Return-Werte für _UPC und _PLD landen da in Arg0 und Arg1 und werden dann als Package weitergereicht an Methode GUPC? Und dann? Würds ganz gerne verstehen.

Code

```
1. Method (GUPC, 2, Serialized)
2. {
3. Name (PCKG, Package (0x04)
4. {
5. 0xFF,
6. 0x03,
7. Zero,
8. Zero
9. })
10. PCKG [Zero] = Arg0
11. PCKG [One] = Arg1
12. Return (PCKG) /* \GUPC.PCKG */
13. }
```

Alles anzeigen

Beitrag von „apfelnico“ vom 14. Oktober 2021, 08:54

> Zitat von ST33Z0
>
> Ich habe einen USB C port an meinem rechner. Ich nutze den zwar nicht, aber mich würde interessieren, was die Bezeichnung bedeutet: "USB 3.2 Type-C+Sw." Sw = switch? Was wird denn da geswitcht? offline

hatte ich im Eingangsthread geschrieben, vielleicht nicht genügend erklärt.

> Zitat von apfelnico
>
> 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 Portgenutzt wird)
>
> 0xFF – USB2 intern (zum Beispiel für Bluetooth)

Dein hinterer USB-C Port ist soweit ich mich erinnere, HS08/SS08. Die bekommen "0x09" (SW = Switch). Denn du kannst den Stecker verdrehsicher einstecken, völlig egal wie rum, es sind alle Kontakte doppelt zum eigentlichen Port gelegt.

Der interne USB-E (so heißt die Buchse), den du für ein Gehäuse-USB-C verwenden kannst, liegt auf SS01 und SS02 für USB3.2, beide Ports werden abwechselnd je nach Steckerrichtung genutzt, bekommen also beide "0x0A". Der passende usb2-Anteil dazu ist SS01. Auch der wird mit "0x0A" deklariert. Der wechselt nicht mit einem anderen Port. Denn SS02 soll nach deinem PDF ja ein interner USB2-Hub sein.

> Zitat von ST33Z0
>
> Könnte mir jemand diese Konstruktion erklären?

Das "Package" (die vier Werte) werden übergeben in jeden Port in die Methode _UPC. Die ersten beiden Werte ("Zero" und "One") des Package sind variabel und werden mittels Argument "Arg0" und "Arg1" angetriggert.

In jeden Port hingegen hast du dann die eigentliche Methode "_UPC".

Beispielsweise:

Code

```
1. Method (_UPC, 0, NotSerialized) // _UPC: USB Port Capabilities
2. {
3. Return (GUPC (0xFF, 0x03))
4. }
```

Hier wird letztlich über "return" gesagt, packe mir den Inhalt des Package von GUPC mit Arg0=0xFF, Arg1=0x03 rein. Mann kann das dann wieder differenzierter machen mit:

Code

```
1. Method (_UPC, 0, NotSerialized) // _UPC: USB Port Capabilities
2. {
3. If (_OSI ("Darwin"))
4. {
5. Return (GUPC (Zero, Zero))
6. }
7. Else
8. {
9. Return (GUPC (0xFF, 0x03))
10. }
11. }
```

Alles anzeigen

In diesem Falle würde bei "Darwin" (macOS) dieser Port deaktiviert, unter allen anderen Systemen allerdings als "USB3.x" (mit "normaler" USB-A Schnittstelle) deklariert werden.

Beitrag von „5T33Z0“ vom 14. Oktober 2021, 09:21

apfelnico Vielen Dank für die Erläuterungen. Du hast dann in meinem Fall "Arg1" zu "GUPC" hinzugefügt, damit man Port an/aus und Porttyp unabhängig von einander deklarieren kann, richtig?

Beitrag von „apfelnico“ vom 14. Oktober 2021, 09:22

ST33Z0

mal etwas näher an deinem speziellen Code dran:

Code

```
Method (_UPC, 0, NotSerialized) // _UPC: USB Port Capabilities
{
If (_OSI ("Darwin"))
{
Return (GUPC (Zero, Zero))
}
Else
{
Return (GUPC (0xFF, 0x0A))
}
}

Method (_PLD, 0, NotSerialized) // _PLD: Physical Location of Device
{
If (_OSI ("Darwin"))
{
Return (GPLD (Zero, Zero))
}
ElseIf ((H1TC == Zero))
{
If (((UMAP & One) == One))
{
Return (GPLD (One, One))
}
Else
{
Return (GPLD (Zero, One))
}
}
Else
{
Return (\_SB.UBTC.RUCC (H1CR, 0x02))
}
}
```

Alles anzeigen

_UPC ist einfach, falls du aber auch _PLD ändern möchtest – hier wird nur beschrieben, wo sich der Port befindet, ob er zu einer Gruppe gehört, welcher Farbe er hat etc, sieht letztendlich so

ausgeschrieben aus:

Code

```
1. Name (_PLD, Package (0x01) // _PLD: Physical Location of Device
2. {
3. ToPLD (
4. PLD_Revision = 0x1,
5. PLD_IgnoreColor = 0x1,
6. PLD_Red = 0x0,
7. PLD_Green = 0x0,
8. PLD_Blue = 0x0,
9. PLD_Width = 0x0,
10. PLD_Height = 0x0,
11. PLD_UserVisible = 0x1,
12. PLD_Dock = 0x0,
13. PLD_Lid = 0x0,
14. PLD_Panel = "UNKNOWN",
15. PLD_VerticalPosition = "UPPER",
16. PLD_HorizontalPosition = "LEFT",
17. PLD_Shape = "UNKNOWN",
18. PLD_GroupOrientation = 0x0,
19. PLD_GroupToken = 0x0,
20. PLD_GroupPosition = 0x0,
21. PLD_Bay = 0x0,
22. PLD_Ejectable = 0x0,
23. PLD_EjectRequired = 0x0,
24. PLD_CabinetNumber = 0x0,
25. PLD_CardCageNumber = 0x0,
26. PLD_Reference = 0x0,
27. PLD_Rotation = 0x0,
28. PLD_Order = 0x0,
29. PLD_VerticalOffset = 0x0,
30. PLD_HorizontalOffset = 0x0)
31.
32. })
```

Alles anzeigen

Bei dir im vorhanden Code werden da auch zwei Variablen übergeben, und es ist schon eine If/Else-Schleife vorhanden. Die habe ich dann mal um If/ElseIf/Else verlängert, um im Falle "Darwin" (macOS) nur "Zero, Zero" zu übergeben (nix), während unter allen anderen Systemen das gemacht wird, was dort eh schon stand. Die Werte "Zero, Zero" sieht man bei dir bei den beiden "Platzhalterports" USR1 und USR2. Daran hatte ich mich orientiert. Die sind eh IMMER tot.

Beitrag von „apfelnico“ vom 14. Oktober 2021, 09:33

Hier kann etwas dazu gestöbert werden, welche Position wofür steht und noch mehr:

https://uefi.org/specs/ACPI/6.…upc-usb-port-capabilities

Das ist jetzt nur der Direktlink zu dieser Thematik. Generell ist das DIE Adresse, um sich Klarheit zu verschaffen.

Beitrag von „5T33Z0“ vom 14. Oktober 2021, 09:37

Ja, das mit den If/else und OSI-weichen hatte ich verstanden. Hatte nur noch nicht kapiert wie diese "Args" funktionieren. Aber jetzt hab ich's glaub ich verstanden. Danke. In dem Dokument habe ich gestern auch schon gediggt.

Wenn ich die Werte 3 und 4 des Packages auch noch übergeben wollen würde, würde ich noch

PCKG [Two] = Arg2 und PCKG [Three] = Arg3 einbauen, ja?

Beitrag von „apfelnico“ vom 14. Oktober 2021, 10:03

> Zitat von maschinenwart
>
> Ich habe da mal eine Bitte, könntest du dir eventuell die SSDT / DSDT von meinem GA-Z390 D Board anschauen?
>
> (SSDT-6-AMI) Die Ports HS08 und HS13 verweisen auf die Methoden TUPC / TPLD. Diese beiden Ports sind die USB2 Ports des Onboard Thunderbolt Controllers. Wie würde man bei diesen Ports vorgehen, um sie richtig einzubinden oder zu deaktivieren?

Hier siehst du, dass der zweite Wert (One) getauscht wird. Und zwar auf den Wert "0x0A", welcher nebenbei bemerkt falsch wäre, wenn die beiden HS08/13 auf je einen Thunderboltport laufen. Korrekt wäre dann "0x09" – USB-C (SW). Gehen die beide an einen Thunderboltport, dann bleibt es bei "0x0A". Möchtest du diese auch ausschalten könnten, füge in die Methode "TUPC" die Zeile "PCKG [Zero] = Arg0" ein und ändere die Zeile "PCKG [One] = Arg0" in "PCKG [One] = Arg1".

Dann wäre in zum Beispiel HS08 der Code so zu gestalten

Code

1. Method (_UPC, 0, NotSerialized) // _UPC: USB Port Capabilities
2. {
3. Return (TUPC (0xFF, 0x09))
4. }

mit "0xFF" ist der aktiv, mit "Zero" (0x00) aus. Analog dazu HS13 gestalten.

Edit: "GA-Z390 D" hat keinen onboard Thunderbolt.Controller?

Beitrag von „apfelnico“ vom 14. Oktober 2021, 10:06

> Zitat von ST33Z0
>
> Wenn ich die Werte 3 und 4 des Packages auch noch übergeben wollen würde, würde

> ich noch
>
> PCKG [Two] = Arg2 und PCKG [Three] = Arg3 einbauen, ja?

 fast.

Nur "Zero" und "One" sind zulässig als alternativer Schreibweise zu "0x00" und "0x01". In dem Fall wäre also "0x02" und "0x03" angebracht. Allerdings findet da nix weiter statt, sind Platzhalter für eventuelle zukünftige Werte, müssen aber wie gehabt als "Viererpackage" übergeben werden.

Beitrag von „5T33Z0“ vom 14. Oktober 2021, 10:21

Okay. Danke. Ja, mir ging es nur darum, die Logik zu verstehen. Nutzen möchte ich das nicht. Packages müssen also immer mindestens 4 werte enthalten, ja?

Beitrag von „maschinenwart“ vom 14. Oktober 2021, 10:36

> Zitat von apfelnico
>
> Edit: "GA-Z390 D" hat keinen onboard Thunderbolt.Controller?

...ich meinte das Z390 Designare  sorry...und vielen Dank für die Erläuterung. Werde ich nachher gleich mal testen...

Edit:

Funktioniert ausgezeichnet! 

Beitrag von „apfelnico“ vom 14. Oktober 2021, 11:31

> Zitat von ST33Z0
>
> Packages müssen also immer mindestens 4 werte enthalten, ja?

Nein, beliebig. Nur diese für _UPC jedoch. Denn der korrekte (direkte) Ausdruck in einem Port wäre zum Beispiel:

Code

```
1. Name (\_UPC, Package (0x04) // _UPC: USB Port Capabilities
2. {
3. 0xFF,
4. 0x03,
5. Zero,
6. Zero
7. })
```

Das entspricht letztendlich dem Konstrukt Methode _UPC und GUPC. So wie ich es im ersten Thread beschrieb.

Beitrag von „apfelnico“ vom 14. Oktober 2021, 16:10

Eine weitere, eigene Variante für Methode _UPC innerhalb eines Ports.

inklusive "Weiche":

Code

```
1. Method (_UPC, 0, NotSerialized) // _UPC: USB Port Capabilities
2. {
3. Name (HACK, Package (0x04)
4. {
5. Zero,
6. Zero,
```

```
7. Zero,
8. Zero
9. })
10. Name (ORIG, Package (0x04)
11. {
12. 0xFF,
13. 0x03,
14. Zero,
15. Zero
16. })
17. If (_OSI ("Darwin"))
18. {
19. Return (HACK)
20. }
21. Else
22. {
23. Return (ORIG)
24. }
25. }
```

Alles anzeigen

und ohne "Weiche":

Code

```
1. Method (_UPC, 0, NotSerialized) // _UPC: USB Port Capabilities
2. {
3. Name (ORIG, Package (0x04)
4. {
5. 0xFF,
6. 0x03,
7. Zero,
8. Zero
9. })
10. Return (ORIG)
11. }
12. }
```

Alles anzeigen

Beitrag von „5T33Z0“ vom 14. Oktober 2021, 16:24

Was ich bis jetzt noch gar nicht verstanden habe, ist, wie man erkennt, an welchem Port der USB stick gerade steckt. Mit IORegExploere, nehme ich an?

Ich kannte meine Ports ja schon vorher, von daher ware es in meinem Fall egal, aber das gilt ja für die meisten nicht, denke ich.

Beitrag von „apfelnico“ vom 14. Oktober 2021, 16:31

Jupp, IORegistryExplorer zeigt das wunderbar dynamisch an. Bei einem neuen Gerät wird ein Baum hinzugefügt, dabei für eine kurze Zeit grün eingefärbt. Beim Entfernen ist's rot.

Beitrag von „maschinenwart“ vom 15. Oktober 2021, 13:03

> Zitat von apfelnico
>
> ...
>
> 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.
> 

Hierzu hätte ich noch ein paar Fragen. Ich benutze dieses Board ja auch schon einige Jahre als Hauptrechner aber sleep/wake hatte bisher bei mir noch nie funktioniert.

Funktioniert das bei dir und wenn ja, welche Einstellungen (BIOS) sind hierfür notwendig?

Benutzt du den internen Bluetooth und wenn ja, benötigt man hierfür besondere Kexts?

Beste Grüße

MW

Beitrag von „apfelnico“ vom 15. Oktober 2021, 13:39

maschinenwart

"normale", öfter gepostete BIOS-Einstellungen. Sleep funktioniert so wie es soll. Benutze internes WLan und Bluetooth, keine extra Kexts. Denn ich habe vom Mainboard die weiße Plastik-Abdeckung abgenommen, den "Blechkasten" abgeschraubt und das interne Wifi/BT-Modul gegen ein originales von Apple (inklusive Adapter) eingesetzt. Die Antennenstecker passten perfekt.

Aber auch, wenn du eine extra Karte nutzt und dessen BT-Modul an einen internen USB-Slot gesteckt ist, kannst du diese Einstellungen des Ports auf deinen Port übertragen.

Du kannst mir gern dein EFI zukommen lassen, dann schaue ich mir das an.

Beitrag von „maschinenwart“ vom 15. Oktober 2021, 14:02

...das Angebot nehme ich gerne wahr, vielen Dank!

Ich habe eine Fenvi am USB-Port HS08 angeschlossen. Die funktioniert ja auch OOB.

Im Anhang mein EFI...

Beitrag von „apfelnico“ vom 15. Oktober 2021, 15:44

maschinenwart

bitte schicke mir noch ein aktuelles IORegistryExplorer-File. Muss noch etwas nachschauen. Gern auch per PM

Beitrag von „maschinenwart“ vom 15. Oktober 2021, 16:15

...hab ich dir gerade geschickt... 

Beitrag von „5T33Z0“ vom 15. Oktober 2021, 16:52

_PLD zuweisen ist eher "Kür" als "Pflicht", oder? Zumindest wenn man es so detailliert macht, wie im Codesnippet.

Beitrag von „apfelnico“ vom 15. Oktober 2021, 17:04

ST33Z0

Jein. 

Position, Form und Farbe muss nicht. Die Punkte "Ejectable" und "Visible" sind dann doch wieder interessant.

Beitrag von „5T33Z0“ vom 15. Oktober 2021, 18:01

Ist das wohl der Ursprung der Xhci Port Limit Patches und des XhciPortlimit Quirks?

https://pikeralpha.wordpress.c...sb-fix-no-kexts-required/

Beitrag von „N0b0dy“ vom 15. Oktober 2021, 22:03

cobanramo

Ich habe versucht zu jeden Ausgang eine neue _DSM Methode zu schreiben, um damit den comment in IOReg. zu zeigen, hat leider nicht geklappt.

Vlt. kann hier apfelnico weiterhelfen 👍

Beitrag von „apfelnico“ vom 16. Oktober 2021, 08:44

N0b0dy cobanramo

Wobei könnte ich helfen?

Beitrag von „N0b0dy“ vom 16. Oktober 2021, 08:51

Er möchte jeden Ausgang seine Position in IOReg mittel Comment zeigen lassen.

Ich habe zu den anderen Methoden UPC und PLD noch die bekannte DSM Methode hinzugefügt aber leider wird nicht gezeigt.

Beitrag von „apfelnico“ vom 16. Oktober 2021, 08:59

Dafür ist PLD da. Dort kann auf den Millimeter genau mit welcher Form und welche Farbe, zu welchem Subset etc. zugehörig exakt definiert werden. Wozu auch immer sich das ausgetüftelt wurde und was auch immer diese Infos darstellen kann. Mir erscheinen dort zwei Infos wichtig: Visable und Ejectable.

Beitrag von „5T33Z0“ vom 16. Oktober 2021, 09:58

apfelnico N0b0dy

Hab mir die Beschreibung für _PLD durchgelesen:

https://uefi.org/specs/ACPI/6....tml#buffer-0-return-value

Das Codesnippet scheint mir ein wenig veraltet. "Visible" scheint sich auf die Sichtbarkeit des Physischen Ports zu beziehen, nicht auf die Sichtbarkeit im Betriebssystem selbst. Der Wert heißt eigentlich "User Visible" – also das es einen physischen Port gibt, den man erreichen kann: "set if the device connection point can be seen by the user without disassembly."

Mir scheinen nur relevant:

- **Revision**: 0x02 (im Snippet noch 0x01)

- **Ejectable** (Set if the device is ejectable. Indicates ejectability in the absence of _EJx objects.)

Für die Position könnte man es versuchen mit:

- Group Token

- Group Position und evtl

- Group Orientation

Aber mir scheint das eher ne Spielerei zu sein im Privatbereich. Ich glaube, das Macht nur Sinn, für irgendwelche großen Anlagen mit etlichen Ports, um den Port überhaupt erstmal zu finden, weil es so viele gibt.

Beitrag von „cobanramo“ vom 16. Oktober 2021, 13:02

Ich spiel mal bissl rum mit deinem beispiel nachher Nico.
Es wäre schön gewesen wenn man das wie beim Kext "Comment" definieren könnte dachte ich mir.

So wichtig ist das ja jetzt auch nicht.

Gruss Coban

Beitrag von „apfelnico“ vom 16. Oktober 2021, 14:54

> Zitat von cobanramo
>
> Es wäre schön gewesen wenn man das wie beim Kext "Comment" definieren könnte dachte ich mir.

Das könnte ja so aussehen:

Code

```
1. Scope (HS01)
2. {
3. Name (INFO, "hier meine Lieblingsinfos")
4. Method (_UPC, 0, NotSerialized) // _UPC: USB Port Capabilities
5. {
6. Return (GUPC (0xFF, 0x03))
7. }
```

8.
9. Method (_PLD, 0, NotSerialized) // _PLD: Physical Location of Device
10. {
11. Return (GPLD (DerefOf (UHSD [Zero]), One))
12. }
13. }

Alles anzeigen

Beitrag von „cobanramo“ vom 16. Oktober 2021, 17:19

Code

1. Device (HS02)
2. {
3. Name (_ADR, 0x02) // _ADR: Address
4. Name (INFO, "MYSTIC LIGHT Port Intern")
5. Method (_UPC, 0, NotSerialized) // _UPC: USB Port Capabilities
6. {
7. Return (GUPC (One, Zero))
8. }
9.
10. Method (_PLD, 0, NotSerialized) // _PLD: Physical Location of Device
11. {
12. Return (GPLD (Zero, 0x02))
13. }
14. }

Alles anzeigen

Verstehe ich richtig Nico das das Info eben nur für den SSDT ist und nicht im Ioreg auftaucht wie mit der Kext & Comment lösung?

Beitrag von „apfelnico“ vom 16. Oktober 2021, 18:45

Das war nur mal so ein Gedanke, auf jeden Fall APCI-konform. Bin immer noch unterwegs und habe keinen Zugang zu meinem Rechner, kann nichts testen.

Beitrag von „N0b0dy“ vom 16. Oktober 2021, 19:30

Ich habe es getestet, wird nicht gezeigt

Beitrag von „cobanramo“ vom 16. Oktober 2021, 19:37

Kein Problem Nico, das eilt nicht, wie gesagt das ist nur ne test und soll informativ beitragen. Hab mal deine "Name Info" mit meinem ssdt bestückt, wäre halt schon bissl luxus gewesen wenn die infos`s auch im Ioreg auftauchen würden

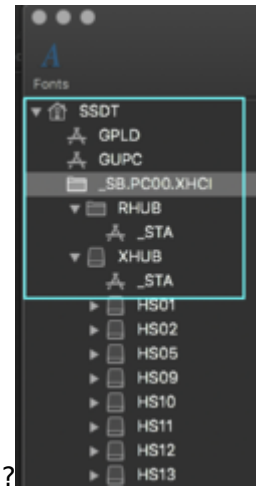
Dank N0b0dy und dir hab ich jetzt auch ein lupenrein funktionierendes ssdt für diesen Board.

Hier hab ich noch ein paar Kleinigkeiten korrigiert auf dem SSDT.

Beitrag von „5T33Z0“ vom 16. Oktober 2021, 20:17

cobanramo Hast Du das alles so strukturiert, oder war das schon so?

Also erst die Routinen, dann der Device Pfad und dann die XHUB und RHUB und die Ports.

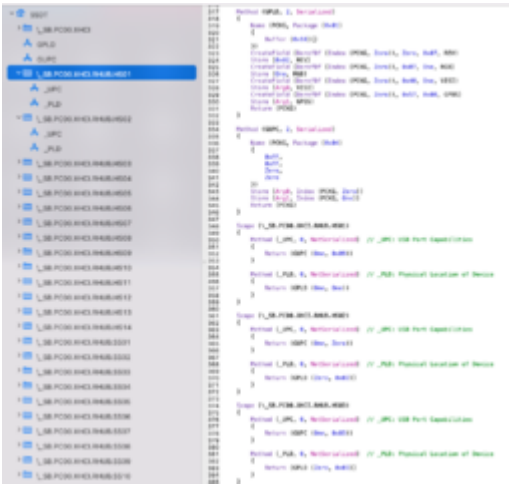
? 

Beitrag von „cobanramo“ vom 16. Oktober 2021, 20:25

Das ist das kunst vom N0b0dy gewesen, er hat also die ganze Arbeit geleistet, hab nur die adressen und die infos nachgebessert

Gruss Coban

EDIT: Original sieht das bei mir so aus;



Beitrag von „N0b0dy“ vom 16. Oktober 2021, 20:43

apfelnico ST33Z0

Aus eure verlinkte Webseiten könnte ich die Method GPLD entziffern. zum Beispiel nehme ich sie aus Coban's SSDT

Code

```
1. Method (GPLD, 2, Serialized)
2. {
3. Name (PCKG, Package (0x01)
4. {
5. Buffer (0x10){}
6. })
7. CreateField (DerefOf (Index (PCKG, Zero)), Zero, 0x07, REV)
8. Store (0x02, REV)
9. CreateField (DerefOf (Index (PCKG, Zero)), 0x07, One, RGB)
10. Store (One, RGB)
11. CreateField (DerefOf (Index (PCKG, Zero)), 0x40, One, VISI)
12. Store (Arg0, VISI)
13. CreateField (DerefOf (Index (PCKG, Zero)), 0x57, 0x08, GPOS)
14. Store (Arg1, GPOS)
15. Return (PCKG)
16. }
```

Alles anzeigen

1-CreateField (DerefOf (Index (PCKG, Zero)), Zero, 0x07, REV) beschreibt "Revision", daher stehet REV und hat 8 bits von null bis 7

Store (0x02, REV) dann gibt immer Revision 2

2-CreateField (DerefOf (Index (PCKG, Zero)), 0x07, One, RGB) beschreibt "Ignore Color", daher steht auch RGB und ha nur ein Bit.

Store (One, RGB) dann gibt immer eins, das heißt inaktive

CreateField (DerefOf (Index (PCKG, Zero)), 0x40, One, VISI) beschreibt "User Visible", daher steht VISI und ebenfalls hat ein Bit also aktive oder inaktive.

Bei "Store (Arg0, VISI)" steht erstes Argument für die Methode entweder 0 invisible oder 1 visible

CreateField (DerefOf (Index (PCKG, Zero)), 0x57, 0x08, GPOS)

Store (Arg1, GPOS) hier beschreibt "Group Position" an den Bit 87, daher steht auch GPOS

Bei "Store (Arg1, GPOS) ist das zweite Argument für die Methode, die eben falls hier ändern können

In diesem Beispiel

Code

```
1. Method (_PLD, 0, NotSerialized) // _PLD: Physical Location of Device
2. {
3. Return (GPLD (One, One))
4. }
```

gibt uns GPLD zurück (Revesion=2, RGB=0, Visible=Yes, Postion=1)

das in Buffer nicht anders als so sehen würde

Code

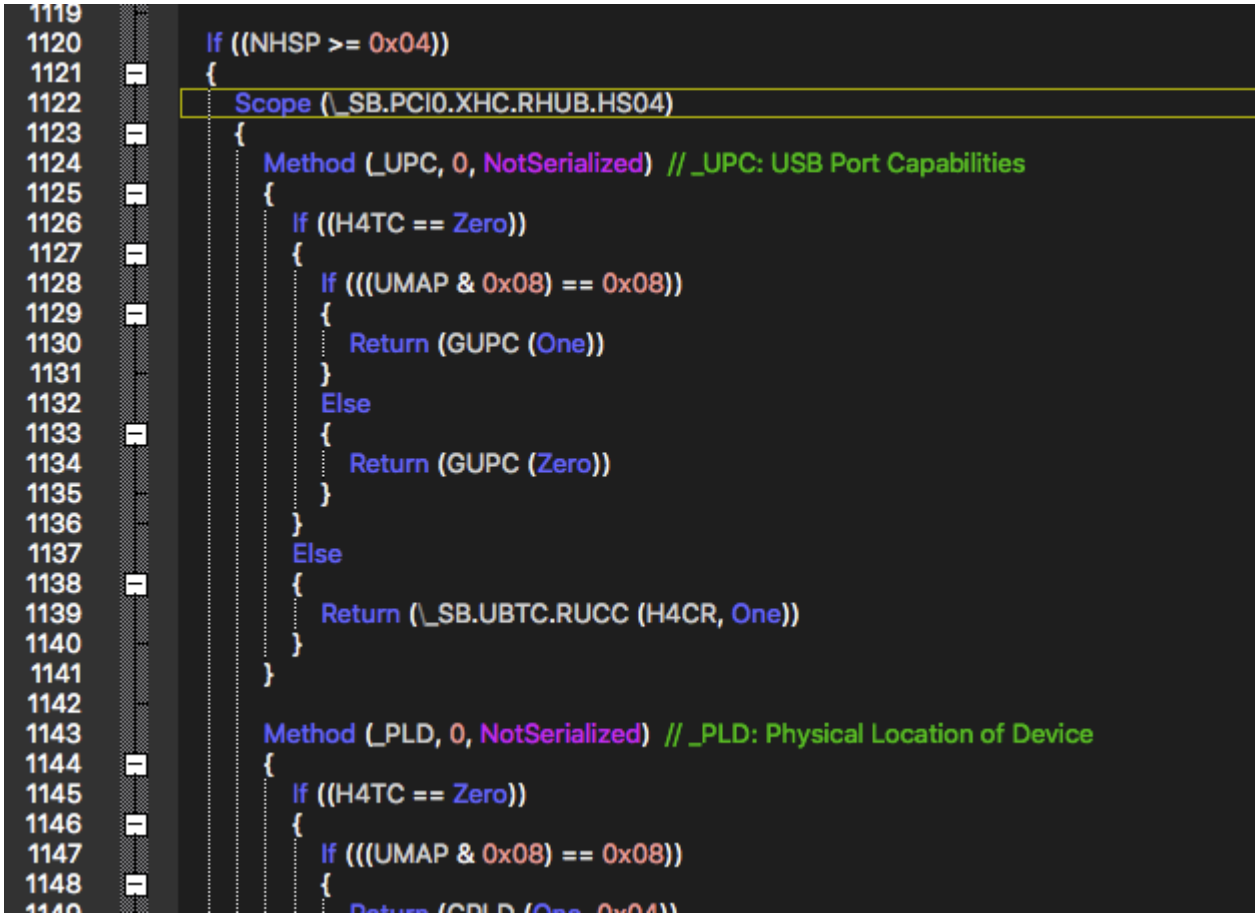
```
1. /* 0000 */ 0x82, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
2. /* 0008 */ 0x31, 0x1C, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

Beitrag von „5T33Z0“ vom 16. Oktober 2021, 21:15

Hab gerade mal QtiASL ausprobiert:

https://github.com/ic005k/QtiASL/releases

Sieht nicht so schick aus, aber es ist cross-platform und man hat gestrichelte Linien zwischen den Klammern, wodurch man die Hierarchie leichter erkennt.



```
1119
1120    If ((NHSP >= 0x04))
1121    {
1122        Scope (\_SB.PCI0.XHC.RHUB.HS04)
1123        {
1124            Method (_UPC, 0, NotSerialized)  // _UPC: USB Port Capabilities
1125            {
1126                If ((H4TC == Zero))
1127                {
1128                    If (((UMAP & 0x08) == 0x08))
1129                    {
1130                        Return (GUPC (One))
1131                    }
1132                    Else
1133                    {
1134                        Return (GUPC (Zero))
1135                    }
1136                }
1137                Else
1138                {
1139                    Return (\_SB.UBTC.RUCC (H4CR, One))
1140                }
1141            }
1142
1143            Method (_PLD, 0, NotSerialized)  // _PLD: Physical Location of Device
1144            {
1145                If ((H4TC == Zero))
1146                {
1147                    If (((UMAP & 0x08) == 0x08))
1148                    {
```

cobanramo Wenn man in "GPLD" 4 von diesen "createfield" Funktionen drin hat, warum brauch man dann nicht unter "Return" nicht auch 4 werte, sondern trotzdem nur 2?

Beitrag von „cobanramo“ vom 16. Oktober 2021, 22:39

ST33Z0
Ich vermute mal das du den N0b0dy fragen wolltest warum das so ist,

Ich bin wirklich absolut nicht fit im Bereich Acpi, denke aber das es bei den returns eben nur das angegeben wird was nötig ist und bei denen wo es nötig wird die fehlenden auch mitgegeben wird.

Gruss Coban

EDIT: vielleicht lieg ich auch völlig falsch aber ich könnte mir noch vorstellen das man eben über diesen "Visible=Yes, Postion=1" genau die Ports die am Board (interne header oder fest verdrahtet und herausgeführte Usb Ports) darstellt oder steuert.

Beitrag von „5T33Z0“ vom 16. Oktober 2021, 23:09

achso, ja, sorry, war für @N0b0dy

Beitrag von „N0b0dy“ vom 16. Oktober 2021, 23:10

GPLD gibt immer diese 4 Werte zurück aber der erste und zweite sind immer gleich für alle Ports daher müssen sie nicht geändert werden.

Dritte und vierte müssen nun ja angepasst werden daher sind sie variabel

Wie oben schon geschrieben bei dem dritte handelt es sich um den Ausgang, ob er für uns sichtbar ist oder nicht, wenn _UPC deaktiviert dann _PLD ist irrelevant also für uns nicht sichtbar daher null aber wenn aktiviert ist dann muss sichtbar sein, dass heißt eins

beim vierten Wert entspricht USB Position und ist gleich wie USB Nummer 1, 2, 3, ...

Beitrag von „cobanramo“ vom 16. Oktober 2021, 23:16

Genau, das ist mir mit der 4. wert auch aufgefallen, die entspricht immer den _ADR wert.

Die 3. wert mit visible ist in dem fall die Interne header Port oder Real Port nehm an.

Beitrag von „apfelnico“ vom 17. Oktober 2021, 12:49

> Zitat von cobanramo
>
> Die 3. wert mit visible ist in dem fall die Interne header Port ...

Nich unbedingt, oder eher selten. Die "internen" Header-Ports sind normalerweise für Gehäusefront-USB. Also genau so gekennzeichnet wie die rückwärtigen USB. "Intern" bedeutet hier eher internes Bluetooth, welches auch nicht "ejectable" ist. Oder internes LED-Steuerungsgedöhns.

Oder aber auch interne HUB, erst Ports dahinter sind wieder "user visible". Bitte schaut euch in diesem Zusammenhang auch "ejectable" an und ob das dann jeweils richtig gesetzt wurde.

Beitrag von „cobanramo“ vom 17. Oktober 2021, 13:02

Danke für die ausführlichen Erklärung, glaub langsam verstehe ich dat ding auch besser, wird gleich gecheckt.

Konntest du das mit dem Info`s dem Ioreg übergeben was in Erfahrung bringen?

Gruss Coban

Beitrag von „atl“ vom 21. Oktober 2021, 18:34

> Zitat von apfelnico
>
> 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.

Könntest du das noch erklären? Was muss ich da übernehmen, um das auch für mein System anpassen zu können. Ich habe meine SSDT mal angepasst und seit dem habe ich das Problem, wenn der Rechner längere Zeit (> 30min) im Sleep ist.

Beitrag von „5T33Z0“ vom 21. Oktober 2021, 18:47

atl Das bezieht sich auf beiden Werte für das Package "Name (_UPC, Package (0x04)" in Port HS13:

Code

```
1. Scope (\_SB.PC00.XHCI.RHUB.HS13)
2. {
3. Method (_STA, 0, NotSerialized) // _STA: Status
4. {
5. Return (0x0F)
6. }
7.
8. Name (_UID, One) // _UID: Unique ID
9. Name (_UPC, Package (0x04) // _UPC: USB Port Capabilities
10. {
11. 0xFF,
12. 0xFF,
13. Zero,
14. Zero
15. })
```

Alles anzeigen

Der erste Wert – 0xFF (für Arg0) – bedeutet, dass der Port eingeschaltet ist. Der zweite Wert –

0xFF(für Arg1) – bezieht sich auf die Art des Ports. 0xFF bedeutet interner 2.0 USB port. Da hängt seine BT Card dran.

Die Werte für übliche Ports an PC mainboards sind im ersten Post aufgelistet:

> Zitat
>
> 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)

Beitrag von „atl“ vom 21. Oktober 2021, 19:02

5T33Z0, was du meinst, habe ich schon berücksichtigt und meinen Port als internen definiert. Ich denke eher, mit "Special" sind folgende Methoden innerhalb von HS13 gemeint, denn apfelnico erwähnte etwas mit "wecken" und "DeepSleep":

Code

```
1. Method (_PRW, 0, NotSerialized) // _PRW: Power Resources for Wake
2. {
3. Return (Package (0x02)
4. {
5. 0x6D,
6. 0x04
7. })
8. }
9.
10. Method (_CRS, 0, Serialized) // _CRS: Current Resource Settings
11. {
12. Name (ABUF, Buffer (0x02)
```

```
{
0x79, 0x00 // y.
})
Return (ABUF) /* \_SB_.PC00.XHCI.RHUB.HS13._CRS.ABUF */
}

Method (DTGP, 5, NotSerialized)
{
If ((Arg0 == ToUUID ("a0b5b7c6-1318-441c-b0c9-fe695eaf949b") /* Unknown UUID */))
{
If ((Arg1 == One))
{
If ((Arg2 == Zero))
{
Arg4 = Buffer (One)
{
0x03 // .
}
Return (One)
}

If ((Arg2 == One))
{
Return (One)
}
}
}

Arg4 = Buffer (One)
{
0x00 // .
}
Return (Zero)
}

Method (_DSM, 4, NotSerialized) // _DSM: Device-Specific Method
{
If ((Arg0 == ToUUID ("a0b5b7c6-1318-441c-b0c9-fe695eaf949b") /* Unknown UUID */))
{
Local0 = Package (0x08)
{
"baud",
```

```
55. Buffer (0x08)
56. {
57. 0xC0, 0xC6, 0x2D, 0x00, 0x00, 0x00, 0x00, 0x00 // ..-.....
58. },
59.
60. "parity",
61. Buffer (0x08)
62. {
63. 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 // ........
64. },
65.
66. "dataBits",
67. Buffer (0x08)
68. {
69. 0x08, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 // ........
70. },
71.
72. "stopBits",
73. Buffer (0x08)
74. {
75. 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 // ........
76. }
77. }
78. DTGP (Arg0, Arg1, Arg2, Arg3, RefOf (Local0))
79. Return (Local0)
80. }
81.
82. Return (Zero)
83. }
```

Alles anzeigen

Beitrag von „pstr“ vom 24. Oktober 2021, 11:20

atl kurze Frage weil Du ja ein ähnliches Mainboard hast. Die SSDT Anpassungen zu meinem z390pro (SSDT-6) haben leider keinen weitergehenden Erfolg hinsichtlich Bluetooth am HS11 welches unter Monterey nicht mehr funktioniert wie bisher.

Die obigen Anpassungen der Methoden DTGP und DSM können auch nicht greifen da die UUID

nicht passt oder sehe ich falsch apfelnico ?

Dennoch gute Sache USBPorts.kext obsolet gemacht zu haben

Grüße

Beitrag von „atl“ vom 24. Oktober 2021, 12:21

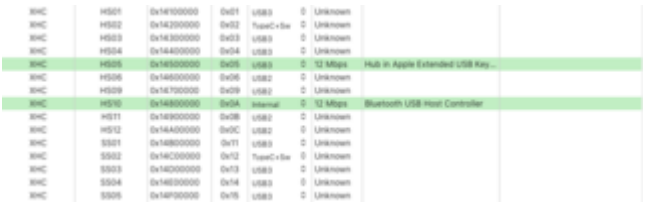
pstr, da kann ich leider keine wirklich greifbare Aussage machen, da ich Monterey im Moment nur sporadisch im Einsatz habe noch Bluetooth wirklich einsetze. Aber bei meinen Tests funktioniert Bluetooth.

Im Moment habe ich aber ein anderes Problem. Seit der Umstellung auf die SSDT schläft der Mac nicht mehr. Er wacht immer mit folgender Meldung sofort wieder auf:

Code

1. 2021-10-24 00:26:06.776911+0200 0x1649 Default 0x0 191 0 powerd: [powerd:sleepWake] Wake reason: "<private>" identity: "<private>"
2. 2021-10-24 00:26:52.874388+0200 0x74 Default 0x0 0 0 kernel: (AppleACPIPlatform) AppleACPIPlatformPower Wake reason: HS10 XDCI CNVW USBW (User)
3. 2021-10-24 00:26:52.874389+0200 0x74 Default 0x0 0 0 kernel: (AppleACPIPlatform) AppleACPIPlatformPower Wake reason: HS10 XDCI CNVW USBW (User)

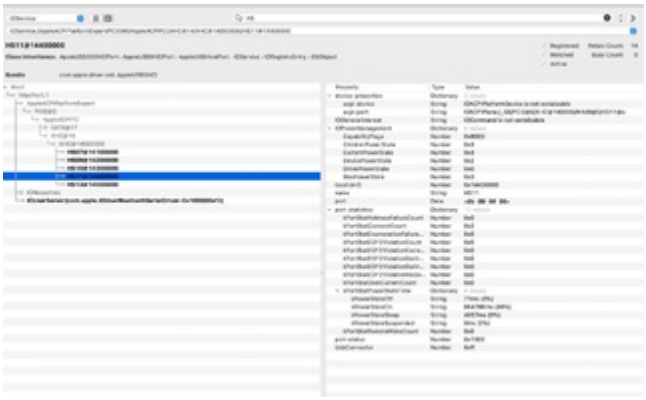
Am HS10 ist meine Bluetooth-Karte angeschlossen und der Port ist auch korrekt als interner USB2-Port deklariert.



Selbst wenn ich alles anderen USB-Geräte abziehe, wacht der Rechner sofort wieder auf.

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

Bir mir ist das HS11, der IOReg sieht so aus, kannst ja mal schauen wie es bei Dir ist



Beitrag von „5T33Z0“ vom 24. Oktober 2021, 14:02

pstr Es gibt generell Probleme mit Bluetoth unter Monterey. Soll heißen: fall Bluetooth unter Big Surr oder sonst wo funktioniert, dann liegt's höchstwahrscheinlich nicht an der SSDT.

Beitrag von „atl“ vom 24. Oktober 2021, 18:19

pstr, das sieht bei mir genauso aus:

Beitrag von „N0b0dy“ vom 24. Oktober 2021, 18:43

> Zitat von atl
>
> Selbst wenn ich alles anderen USB-Geräte abziehe, wacht der Rechner sofort wieder auf.

Ich habe das selbe MB und ich sehe deine USBs falsch deklariert HS02, HS06 und HS09 müssen USB3.0 sein und nicht wie auf dem Foto zu sehen...!

Beitrag von „atl“ vom 24. Oktober 2021, 18:53

> Zitat von N0b0dy
>
> ich sehe deine USBs falsch deklariert HS02, HS06 und HS09 müssen USB3.0 sein

Ich habe die Ports getestet und dabei folgende Port-Nummerierung erhalten:

Damit ist bei mir HS02 der USB2-Part von Port 03, der ein TypeC-SW Port ist, deshalb hat der auch diesen Typ. HS06 hat bei mir den Typ "USB2", da ich den USB 3.0-Teil (SS06) deaktiviert habe. Und HS09 ist Teil des internen USB-Connectors, der nur USB2-Ports hat. Meiner Meinung nach sollte das Mapping damit richtig sein.

Beitrag von „N0b0dy“ vom 24. Oktober 2021, 19:09

ich entschuldige mich, habe darauf nicht geachtet, weil HS02 bei mir aus ist aber mit HS06 muss trotzdem USB3.0, egal ob SS06 aus ist oder nicht und bei HS09 muss eigentlich aus sein, wenn BT in HS10 steckt oder hast noch was mit BT header eingesteck ???

atl

Beitrag von „a1k0n“ vom 24. Oktober 2021, 19:31

@atl

Für das Gigabyte Z390m hab ich eine OpenCore 0.7.5 EFI die komplett Big Sur und Monterey kompatibel ist. Nativ install (Beta 10) sowie Upgrade von Big Sur getestet.

Die config.plist ist komplett minimalisiert und ausschließlich nur mit den nötigsten Sachen

gefüllt.

Ausserdem sind alle USB Slots sinnvoll definiert. iPad Pro M1 wird am USB-C Anschluß erkannt und USB Power funktioniert auch. Das MB arbeitet in Verbindung mit einer Fenvi919 und RX580.

Wlan ist ESTi/DE gemappt und alle Apple-Dienste laufen 1A bis auf das iMacPro Sidecar-Schwarze-Bild Problem aufgrund vom fehlenden T2 Chip.

Sleep/Wake rennt auch 1A.

Falls interesse besteht gern hier nochmal melden und ich würde das ganze dann anbieten. Auch ein schneller Switch auf iMac19.1 wäre möglich mit der config.plist Headless oder IGPU only.

Biosversion F9m und Settings könnte ich bei bedarf mit anhängen.

Auch ein anschließender Switch auf VAULT, AppleSecureBoot und BiosSecureBoot ist möglich.

Beitrag von „atl“ vom 24. Oktober 2021, 20:30

N0b0dy, alles gut. Ob HS06 als USB3 oder USB2 deklariert ist, macht aber keinen Unterschied - auf das Sleep-Problem bezogen. Am HS09 hängt der USB2-Frontanschluß vom Gehäuse. Ich habe getrennte Kabel für beides an dem Header HS09+10.

a1k0n, ich hatte auch mal eine EFI, die 1A rannte. Bis ich versucht habe, Big Sur / Monterey mit meinen Thunderbolt-Displays zum Laufen zu bekommen. Bis auf das Sleep/Wake Problem läuft jetzt auch alles. Ich nutze die BIOS-Version F9l, da die F9m in Kombination mit der TitanRidge-Karte und einer Samsung 970 NMVe SSD bei Big Sur zu Abstürzen führt. Du kannst mir gerne deine EFI mal zukommen lassen oder besser noch, veröffentliche sie im Hardware-Center: Gigabyte Z390M Gaming . Ich schaue sie mir dann mal an und vergleiche mit meiner.

Schaden kann es ja nicht.

Beitrag von „pstr“ vom 24. Oktober 2021, 20:38

> Zitat von 5T33Z0
>
> pstr Es gibt generell Probleme mit Bluetoth unter Monterey. Soll heißen: fall Bluetooth unter Big Surr oder sonst wo funktioniert, dann liegt's höchstwahrscheinlich nicht an der SSDT.

Jein, irgendwo ist da noch was nicht korrekt.

ein Wake per Bluetooth geht auch noch innerhalb von geschätzt 30 min nach Sleep-Begin. Später reagiert der Kasten nur noch auf Power-Taste oder über USB angeschlossene Geräte.

Vermutlich geht die Stromversorgung von PCIe 1x Steckplatz runter sodass das BT Modul nicht mehr reagiert (aber nur mal ein Vermutung)

Das wiederum mag von MB zu MB anders im Verhalten sein und dann hat mal jemand keine Probleme oder eben wie unsereins.

{Glaskugel off}

Beitrag von „atl“ vom 24. Oktober 2021, 21:17

N0b0dy, a1k0n, ich habe jetzt noch einmal etwas gestestet. Sleep funktioniert sobald ich die USBPorts.kext anstelle der SSDT aktiviere. Das Portmapping ist bei beiden gleich.

Beitrag von „a1k0n“ vom 24. Oktober 2021, 21:47

Ich nutze auch nur die USBPort.kext + die 2 SSDTs die Hackintool erstellt.

Beitrag von „N0b0dy“ vom 24. Oktober 2021, 22:06

hier kann ich leider nichts sagen, seitdem ich das MB habe, nutze nur ssdt, die ich nach meiner Anleitung in post #3 erstellt habe, alles funktioniert prima

Du kannst mir deine EFI kommen lassen dann sehe ich die ssdt und die Kext an oder du kannst bei der Kext bleiben

atl

Beitrag von „atl“ vom 28. Oktober 2021, 01:23

N0b0dy, noch kurzes Feedback zum meinem Sleep-Problem. Ich habe jetzt mal eine SSDT nach deiner Methode erstellt und in der Tat funktioniert sie genau wie die USBPorts.kext. Aber auch damit habe ich Probleme in der Art, dass System (zumindest unter Catalina) nach einer Weile (gefühlt 15 in.) aufwacht und dann ohne aktiven Bildschirm läuft.

Aber ich glaube, dass da einfach diverse macOS Prozesse die Ursache für die Waches sind. Dem gehe ich noch bei Gelegenheit noch einmal auf den Grund. Zumindest wacht das System jetzt nicht mehr direkt nach dem Aktvieren des Sleep-Mode wieder auf. Das ist schon einmal ein großer Fortschritt.

> Zitat von apfelnico
>
> Oder aber auch interne HUB, erst Ports dahinter sind wieder "user visible". Bitte schaut euch in diesem Zusammenhang auch "ejectable" an und ob das dann jeweils richtig gesetzt wurde.

Würde das nicht bedeuten, dass ich die 2 USB-Ports, die mit der TitanRidge-Karte verbunden sind, um den USB2-Teil zur Verfügung zu stellen, auch als "internal" definiert werden sollten?