USBInjectAll

Beitrag von „Brumbaer“ vom 20. April 2016, 15:27

Mit ElCapitan ist USBInjectAll ins Rampenlicht geraten. Dieser Text, soll über USBInjectAlls Arbeitsweise und Anwendung informieren.

Denglish
Ich verwende englische Ausdrücke, wo es meinem Empfinden nach kein gebräuchliches deutsches Äquivalent gibt (CD - Compact Disk, und nicht KS - Kompakte Scheibe), oder der englische Name allgemein bekannt ist (Computer) oder um Zweideutigkeiten zu vermeiden (Device und Gerät um zwischen einem Gerät aus Sicht des USB Busses und einem Gerät aus Sicht des Anwenders zu unterscheiden).
Es ist nicht notwendig die deutsche Übersetzung dieser Begriffe zu wissen. Es genügt zu wissen was das Ding tut, das so heisst.

USBInjectAll

Wieso, weshalb, warum ?
Seit El Capitan werden die USB Ports eines Hackintosh nicht mehr automatisch dem System bekanntgegeben. Das führt dazu, dass OS X keine USB Geräte erkennt und man z.B. beim Installieren, das gefürchtete Stop Zeichen zu sehen bekommt.
USBInjectAll springt in die Bresche und fügt die Ports wieder ein.Um die Ports einfügen zu können muss es aber wissen, welche USB Ports vorhanden sind.

Kategorien
Dazu teilt USBInjectAll USB Controller in Kategorien ein.
Jeder Controller in der selben Kategorie unterstützt die selbe Anzahl und Art von USB Anschlüssen.
Die momentan bekannten Kategorien sind EH01, EH02, HUB1, HUB2 und bestimmte xHCI Controller die über ihre VendorID und DeviceID identifiziert werden. Im Moment sind dies ausschließlich Controller der Firma Intel (VendorID 8086 und die DeviceIDs 1e31, 8xxx, 9xxx und a12f).
Es ist für einen Rechner möglich Controller von mehr als einer Konfiguration zu besitzen. So treten EH01, EH02 und HUB1, HUB2 meist in Paaren auf.

Wer bin denn du ?

Die Konfiguration wird wie folgt bestimmt.

Mac-Typ, Device Name (EH01, EH02) und EHCI Treiber
Ist der Macintosh ein unterstützer Mac Typ und hat er ein EHCI Device mit dem Namen EH01 oder EH02 so gehört der Controller der Kategorie mit dem selben Namen an.
EHCI ist ein Standard für USB Controller und unterstützt nur USB bis Version 2.0. Der Standard beschreibt einen USB Controller bis auf die Registerebene herab, so dass ein Treiber Controller verschiedener Hersteller steuern kann, solange sie diesem Standard folgen.

Mac-Typ, Device Name (XHC) und XHCI Treiber
Ist der Macintosh ein unterstützer Mac Typ und hat er eine XHCI Device mit dem Namen XHC so gehört der Controller der Kategorie, die der VendorID und ProductID des Controllers entspricht, an.
XHCI ist ein Standard für USB Controller und unterstützt USB 2.0 und USB 3.0. Der Standard beschreibt einen USB Controller bis auf die Registerebene herab, so dass ein Treiber Controller verschiedener Hersteller steuern kann, solange sie diesem Standard folgen. Das X steht for eXtensible - erweiterbar - weshalb verschiedene Portkonfigurationen möglich sind. Aus diesem Grund gibt es nicht eine XHC Kategorie für alle, sondern es gibt für jeden unterstützen Controller eine eigene Kategorie.

USBInternalHub Treiber und Location ID
Bestimme Controller erkennt OS X als interne Hubs. Diese sind an zwei verschiedenen Adressen zu finden: 0x1D100000 und 0x1A100000. Der Kategoriename ist HUB1 oder HUB2 je nach Adresse. Es handelt sich dabei um USB2.0 Controller.

Was nicht passt, wird passend gemacht.
Wenn davon nichts passt, werden auch keine Ports an OS X gemeldet und USBInjectAll wird seinem Namen nicht gerecht. Allerdings kann man u.U. wie unten im zweiten Beispiel gezeigt Board und USBInjectAll einander näher bringen.

Oder auch nicht
Siehe Überschrift.

Suche Anschluss
Die USB Anschlüsse an einem Computer sind für gewöhnlich Buchsen vom Typ A (USB 2.0) oder Typ A SuperSpeed (USB 3.0). Auf dem Mainboard befinden sich zusätzlich USB Anschlüsse in Form von Stiftreihen sogenannten Headern. Sie werden für den Anschluss interner Geräte oder für die Verbindungskabel zu Typ A Buchsen in der Front Blende verwendet.

Die Ports

Jede Kategorie hat eine Liste von unterstützen Ports deren Anzahl von Controller zu Controller variiert. Jedem USB 2.0 Anschluss am Mainboard ist ein HighSpeed Port zugeordnet. Jedem USB 3.0 Anschluss am Mainboard sind zwei Ports zugeordnet. Das liegt daran, dass USB 3.0 Anschlüsse zwei getrennte Busse beinhalten. Einen USB 2.0 Bus und einen SuperSpeed Bus. Entsprechend belegen sie im Controller einen HighSpeed und einen SuperSpeed Port. Umgangssprachlich spricht man auch von USB 2.0 und USB 3.0 Ports, obwohl ein USB 2.0 Port auch zu einem USB 3.0 Anschluss gehören kann.

HUB1 und HUB2 unterstützen jeweils 8 HighSpeed Ports HP11-HP18 und HP21-HP28.
EH01 und EH02 unterstützen 8 bzw. 6 HighSpeed Ports PR11-PR18 und PR21-PR26.
8086_1e31 unterstützt 4 High Speed und 4 SuperSpeed Ports HS01-HS04 und SSP5-SSP8.
8086_8xxx unterstützt 14 High Speed und 6 SuperSpeed Ports HS01-HS14 und SSP1-SSP6.
8086_9xxx unterstützt 9 High Speed und 4 SuperSpeed Ports HS01-HS09 und SSP1-SSP4.
8086_a12f unterstützt 14 High Speed, 10 SuperSpeed und 2 USR Ports HS01-HS14, SS01-SS10 und USR1-USR2.

Was USR1 und USR2 machen und wo sie herkommen ist mir unbekannt. Das Datenblatt des XHCI Controllers spricht nur von 14 HighSpeed und 10 SuperSpeed Ports und erwähnt keine weiteren Ports.
Die Ports in der jeweiligen Liste werden von Controllern der entsprechenden Kategorie unterstützt, es heisst aber nicht, dass alle diese Ports als Anschlüsse am Motherboard zur Verfügung stehen.

Port-Infos
USBInjectAll definiert für jeden Port drei Werte.

Für EH01/EH02 und XHC Ports sind dies:
Der Name ist na ja der Name des Ports. Vier Stellen, 2 oder 3 Buchstaben gefolgt von 2 oder 1 Ziffer. HS und PR stehen für High Speed Ports bzw Ports (USB 2.0), SS und SSP für (Super Speed Port (USB 3.0).
Der USB Connector steht für die Art des Anschlusses. 0 steht für USB-2.0 Typ A, 1 für Mini-AB, 3 für USB 3.0 Typ A, 255 für etwas für das es keinen Code gibt, wie z.B. die Header auf einem Mainboard. Das heisst allerdings nicht dass der Anschluss wirklich dem so definierten Port entspricht, denn USBInjectAll kann den Anschluss ja nicht sehen.
Der port ist die Adresse unter der der Controller den Port anspricht. Die USB 2.0 Ports beginnen bei der Adresse 1. Die USB 3.0 Ports haben höhere Adressen.

Für HUB1 und HUB2 Ports werden definiert:
Der Name. Vier Stellen, 2 Buchstaben gefolgt von 2 Ziffern. HP steht für High Speed Port (USB 2.0).
Der portType ist immer 2.
Der port gibt die Port-Nummer an. Die Ports sind durchnummeriert beginnend bei der 1.

Hau wech den Drech
Die Ports werden in der Reihenfolge ihrer Port Nummer (Inhalt des port-Wertes in den Port-Infos) an OS X gemeldet. Die von der zu erst gefundenen Konfiguration zuerst.

Wer zu spät kommt, den bestraft das Leben
Dummerweise lässt OS X nur 15 Ports zu. Alle Ports nach dem fünfzehnten werden von OS X ignoriert. Dies ist nur bei einigen xHCI Controllern ein Problem, denn nur die haben deutlich mehr als 15 Ports. Da die USB 3.0 Ports wegen ihrer hohen Port Nummern zuletzt gemeldet und somit ignoriert werden, fehlt dann der USB 3.0 Support teilweise oder ganz.

Klassengesellschaft
Es besteht allerdings die Möglichkeit USBInject mitzuteilen, welche USB Ports man gerne an OS X, weiterleiten möchte und welche nicht. Man kann sich also seine 15 Ports aussuchen. Das langt in den meisten Fällen.

Rinn mit de Kinn, anner Leut Kinn sann aach drinn.
Es gibt auch die Möglichkeit OS X so zu patchen, dass mehr als 15 Ports unterstützt werden. Das ist für einen Test hilfreich, für Alltagsbetrieb aber nicht zu empfehlen, denn man weiss nicht ob der Patch nach einem OS X Update noch funktionieren wird und ob nicht doch irgendwo irgendein Programm die 15 Port Beschränkung voraussetzt.

Zwei Beispiele

Clover
Ich beschreibe die Vorgehensweise bei Clover, denn damit bin ich am besten vertraut. Die Vorgehensweise lässt sich allerdings auf andere Methoden der Hackintoshifizierung übertragen.
Wenn von der Clover Installation und ihren Ordner oder der Config.plist die Rede ist, so handelt es sich immer um die Dateien bzw. Ordner auf der EFI Partition von der gestartet wird. Diese

ist im Normalfall im Finder nicht sichtbar und muss gemountet werden, bevor man die Dateien und Ordner auf ihr bearbeiten kann.
Ich verwende hier den Patch, zur Erhöhung der Anzahl der unterstützen Ports, bewusst nicht, damit dieser Text auch gültig bleibt, falls der Patch nicht mehr funktioniert.

Asus Z170i-Pro Gaming

Dieses Board steht stellvertretend für alle Z170 Boards. Der xHCI Controller ist Teil des Z170 Chipsatzes und wird dementsprechend bei allen Z170 Boards verwendet. Die Boards unterscheiden sich allerdings darin wieviele und welchePorts an welche Anschlüsse geführt werden und in der DSDT.

Schuh-Stock

Man bereitet seinen Bootstick wie üblich vor. Ob man Unibeast verwendet oder createinstallmedia aufruft und Clover von Hand installiert, spielt keine Rolle.

Config.plist

Ob man Patches benötigt und/oder bestimmte Flags in der Config.plist setzen muss/oder will, die nicht in Beziehung zu USB stehen, soll an dieser Stelle nicht interessieren.
Es hat sich aber als notwendig erwiesen das FixOwnership Flag im USB Abschnitt der Devices Kategorie in der Config.plist zu setzen.

Kexte

USBInjectAll.kext wird in dem selben Ordner installiert in dem auch FakeSMC.kext installiert wird.
Ich persönlich entferne alle Ordner außer dem Other Ordner aus dem kexts Verzeichnis (EFI/CLOVER/kexts) der EFI Partition des Sticks. FakeSMC.kext und USBInjectAll.kext werden dann in den Other Ordner kopiert.

Habe Port, suche Anschluss

Der Controller wird, wenn er erkannt wird, der 8086_a12f Kategorie angehören, also 14 HighSpeed und 10 Super Speed Ports unterstützen. D.h genau ein SuperSpeed Port wird ohne zusätzliche Konfiguration verfügbar sein und auch nur dann, wenn es tatsächlich mit einem Anschluss verbunden ist. D.h. man muss vermeiden, dass der USB Stick ein SuperSpeed Port verwendet. Die einfachste Methode ist es einen USB 2.0 Stick zu verwenden. Denn selbst,

wenn er in einem USB 3.0 Anschluss steckt, wird er ein HighSpeed Port verwenden. Hat man hingegen einen USB 3.0 Stick, so muss man einen Anschluss verwenden der nur USB 2.0 unterstützt, so dass er gezwungen ist sich mit einem HighSpeed Port zu verbinden.

Blau, blau, blau ist der Zahn

Bluetooth Controller werden meist über USB angeschlossen, selbst wenn sie nicht als Stick daher kommen. In diesem Falle wird Bluetooth nicht funktionieren, solange USB nicht funktioniert.

Versuch macht kluch

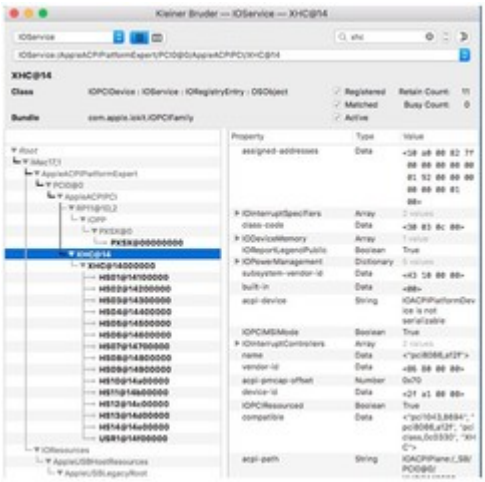
Nun ist es Zeit für einen Boot Versuch. Den Stick in ein entsprechend der obigen Beschreibung ausgesuchten Port und den Rechner eingeschaltet.
Die Wahrscheinlichkeit, dass man in den Installer kommt ist bei Z170 Boards sehr hoch, denn sie sind bis auf die Onchip Grafik einfach zu konfigurieren.
Natürlich kann es immer zu Problemen kommen und die versucht man wie üblich zu lösen.

Stop, Halt, halte, alt

Das das USB Interface die Ursache für ein eventuelles Problem ist, erkennt man am Stop Schild und/oder der Meldung "Can't find boot device".
Sollte dies der Fall sein, so ist vermutlich der Device Name in der DSDT nicht XHC. Wie man den Device-Namen ändert wird im zweiten Beispiel beschrieben.

15 Ports müsst ihr sein

Hat man es geschafft zu booten und OS X zu installieren, geht es nun daran die 15 Ports zu wählen und in diesem Zuge die USB 3.0 Unterstützung herzustellen.
Welche 15 Ports gerade aktuell sind, kann man mit IORegistry Explorer leicht überprüfen.
Gibt man im Suchfeld den Namen des USB-Devices ein (XHC) wird einem das USB-Device, das RootHub-Device und die Ports angezeigt. In unserem Fall sind das die Ports mit den Nummern 1 bis 15, namentlich HS01-HS14 und USR1.



Wo bist du ?

Die Frage welches Port an welchem Anschluss liegt lässt sich auf zweierlei Wegen beantworten. Ich beschreibe erst den einen Weg bist zur fertigen Konfiguration, denn es werden dabei Dinge erklärt, die man für den zweiten Weg braucht.

Hast du einen Plan ?

Wenn die Dokumentation gut ist, ist der einfachste Weg ind Handbuch zu schauen. So gibt es im Handbuch zu Asus Mainboards einen Plan auf dem alle USB Ports eingezeichnet sind. Nicht nur das, sie sind mit Bezeichnungen wie USB910 oder USB3_5-8 versehen. USB3 steht für USB3.0 Anschlüsse nur USB für USB 2.0 Anschlüsse. Die nachfolgende Zahle(en) für das Port. USB 910 sind also 2 Anschlüsse, einer belegt mit HS09 und einer mit HS10. Bei USB 3.0 Anschlüssen entspricht die Nummer ebenfalls den Ports. Da USB 3.0 Anschlüssen allerdings 2 Ports beinhalten ist der Anschluss USB3_5 mit den Ports HS05 und SS05 verbunden. USB3_5-8 sind also vier Anschlüsse die mit den Ports HS05-HS08 und SS05-SS08 verbunden sind. Das spart viel Ausprobieren und sagt einem auch gleich an welchen Anschlüssen welche SuperSpeed Ports liegen.

In Echtigkeit

Der Plan für das Z170i-Pro Gaming weist folgende Anschlüsse aus:
USB3_12(Header), USB3_5-8(4 Buchsen übereinander in der Rückenblende), USB910(2 Buchsen übereinander in der Rückenblende), USB_1112 (Header).
Ein Aufruf von IORegistry Explorer zeigt bei eingeschaltetem Onboard Bluetooth, dass dieses mit dem Port HS13 verbunden ist.
Das sind insgesamt 6 USB 3.0 Anschlüsse und 5 USB 2.0 Anschlüsse macht zusammen 17 Ports. Man will natürlich alle Ports an der Front und Rückenblende verwenden können.

Frontblende habe ich keine also bleiben 10 Ports an der Rückenblende (4 USB3.0 und 2 USB 2.0 Anschlüsse). Ich habe die Wasserkühlung an USB11 und die Lichteffekte an USB12, macht zusammen 12 Ports, dazu kommt der Port für BT macht 13 Ports. Ich könnte nun noch einen USB 3.0 Anschluss von USB3_12 verwenden, entscheide mich aber dagegen, weil ich ihn nicht brauche und halb belegte Header potentielle Fehlerquellen sind.

Positiv- oder Negativ-Liste, das ist hier die Frage.
Nun kann man entweder die zu verwendenden Ports in eine SSDT eintragen oder die nicht zu verwendenden Ports in der config.plist angeben.

Negativ
!!!!! Die Negativ Liste benötigt USBInjectAll Version 0.5.8 oder neuer !!!!!!!!!
Man kann Ports "deaktivieren" indem man sie in der config.plist in einem Kernel Parameter definiert. Dieser hat die Form

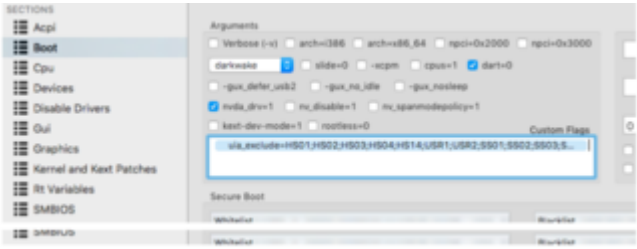
Code

1. -uia_exclude=PORT PORT ist dabei der Name des Ports. Möchte man mehr als ein Port deaktivieren trenn man ihre Namen durch ";".

Ich verwende SS05-SS08, HS05-13 das erfordert also

Code

1. uia_exclude=HS01;HS02;HS03;HS04;HS14;USR1;USR2;SS01;SS02;SS03;SS04;SS09;SS10



Sei doch nicht so negativ

Eine Positv Liste wird in einer SSDT gespeichert, die wiederum in den patched Ordner der Clover Installation kopiert werden muss. Das ist der Ordner /EFI/CLOVER/ACPI/patched Ordner auf der EFI Partition von der gestartet wird.

Zum Editieren bietet sich MaciASL an.

Die zu diesem Beispiel gehörende Datei sieht so aus:

Code

```
1. /*
2. * Intel ACPI Component Architecture
3. * AML Disassembler version 20140926-64 [Oct 24 2014]
4. * Copyright (c) 2000 - 2014 Intel Corporation
5. *
6. * Disassembly of iASLJ9Lkz6.aml, Wed Apr 20 12:56:10 2016
7. *
8. * Original Table Header:
9. * Signature "SSDT"
10. * Length 0x0000026D (621)
11. * Revision 0x01
12. * Checksum 0xDB
13. * OEM ID "Brumba"
14. * OEM Table ID "USBFix"
15. * OEM Revision 0x00003000 (12288)
16. * Compiler ID "INTL"
17. * Compiler Version 0x20140926 (538183974)
18. */
19. DefinitionBlock ("iASLJ9Lkz6.aml", "SSDT", 1, "Brumba", "USBFix", 0x00003000)
20. {
21. Device (UIAC)
22. {
23. Name (_HID, "UIA00000") // _HID: Hardware ID
24. Name (RMCF, Package (0x02)
25. {
26. "8086_a12f",
27. Package (0x04)
28. {
```

```
29. "port-count",
30. Buffer (0x04)
31. {
32. 0x18, 0x00, 0x00, 0x00 /* .... */
33. },
34.
35.
36. "ports",
37. Package (0x1A)
38. {
39. "HS13",
40. Package (0x04)
41. {
42. "UsbConnector",
43. 0xFF,
44. "port",
45. Buffer (0x04)
46. {
47. 0x0D, 0x00, 0x00, 0x00 /* .... */
48. }
49. },
50.
51.
52. "HS05",
53. Package (0x04)
54. {
55. "UsbConnector",
56. 0xFF,
57. "port",
58. Buffer (0x04)
59. {
60. 0x05, 0x00, 0x00, 0x00 /* .... */
61. }
62. },
63.
64.
65. "HS06",
66. Package (0x04)
67. {
68. "UsbConnector",
69. 0xFF,
70. "port",
```

```
Buffer (0x04)
{
0x06, 0x00, 0x00, 0x00 /* .... */
}
},


"HS07",
Package (0x04)
{
"UsbConnector",
0xFF,
"port",
Buffer (0x04)
{
0x07, 0x00, 0x00, 0x00 /* .... */
}
},


"HS08",
Package (0x04)
{
"UsbConnector",
0xFF,
"port",
Buffer (0x04)
{
0x08, 0x00, 0x00, 0x00 /* .... */
}
},


"HS09",
Package (0x04)
{
"UsbConnector",
0xFF,
"port",
Buffer (0x04)
{
0x09, 0x00, 0x00, 0x00 /* .... */
```

```
113. }
114. },
115.
116.
117. "HS10",
118. Package (0x04)
119. {
120. "UsbConnector",
121. 0xFF,
122. "port",
123. Buffer (0x04)
124. {
125. 0x0A, 0x00, 0x00, 0x00 /* .... */
126. }
127. },
128.
129.
130. "HS11",
131. Package (0x04)
132. {
133. "UsbConnector",
134. 0xFF,
135. "port",
136. Buffer (0x04)
137. {
138. 0x0B, 0x00, 0x00, 0x00 /* .... */
139. }
140. },
141.
142.
143. "HS12",
144. Package (0x04)
145. {
146. "UsbConnector",
147. 0xFF,
148. "port",
149. Buffer (0x04)
150. {
151. 0x0C, 0x00, 0x00, 0x00 /* .... */
152. }
153. },
154.
```

```
155.
156.  "SS05",
157.  Package (0x04)
158.  {
159.  "UsbConnector",
160.  0x03,
161.  "port",
162.  Buffer (0x04)
163.  {
164.  0x15, 0x00, 0x00, 0x00 /* .... */
165.  }
166.  },
167.
168.
169.  "SS06",
170.  Package (0x04)
171.  {
172.  "UsbConnector",
173.  0x03,
174.  "port",
175.  Buffer (0x04)
176.  {
177.  0x16, 0x00, 0x00, 0x00 /* .... */
178.  }
179.  },
180.
181.
182.  "SS07",
183.  Package (0x04)
184.  {
185.  "UsbConnector",
186.  0x03,
187.  "port",
188.  Buffer (0x04)
189.  {
190.  0x17, 0x00, 0x00, 0x00 /* .... */
191.  }
192.  },
193.
194.
195.  "SS08",
196.  Package (0x04)
```

```
{
"UsbConnector",
0x03,
"port",
Buffer (0x04)
{
0x18, 0x00, 0x00, 0x00 /* .... */
}
}
}
}
})
}
}
```

Alles anzeigen

Die Datei muss als *ACPI Machine Language Binary* gespeichert werden. Diese Dateien haben die Endung *aml*. Beim Sichern wird die Datei kompiliert und lässt sich erst speichern, wenn keine Fehler mehr vorhanden sind. SSDT-USB-BoardBezeichnung.aml bietet sich als Dateiname an.

Hex, Hex

Im Rahmen einer SSDT oder DSDT werden Zahlen oft als Hexadezimalzahl dargestellt. Dies ist daran zu erkennen, dass am Anfang der Zahl ein 0x steht und zusätzlich zu Ziffern, die Buchstaben a-f vorkommen können. Ich überlasse es Google zu erklären, wie man Hexadezimalzahl in Dezimalzahlen umwandelt und umgekehrt. Es sei aber angemerkt, dass der "OS X" Taschenrechner dies kann.
Man kann an Stelle der Hexadezimalzahlen auch Dezimalzahlen eingeben, aber wenn man die Datei erneut öffnet, werden diese durch Hexadezimalzahlen ersetzt.

Aller Anfang ist leicht

Die Datei beginnt mit Kommentaren, die beim Kompilieren automatisch erstellt werden und uns jetzt nicht interessieren.

Richtig los geht es mit:

Code

```
1. DefinitionBlock ("iASLDlx6pm.aml", "SSDT", 1, "Brumbaer", "USBFix", 0x00003000)
2. {
3. Device (UIAC)
4. {
5. Name (_HID, "UIA00000") // _HID: Hardware ID
```

{ und } definieren Blöcke und müssen ausgeglichen sein. Alles zwischen den Klammern ist der Inhalt des Blocks. Ein Block kann wieder Blöcke enthalten.

Am Ende der Datei wir man also

Code

```
1. }
2. }
```

finden, womit die zwei Blöcke wieder geschlossen werden.
Der DefinitionBlock, sagt worum es sich bei der Datei handelt. Den "Brumbaer" kann man durch ein eigenes Kürzel ersetzen, den Rest lässt man so.
USBInjectAll sucht nach einem Device mit Namen *UIAC* und wenn es dieses findet, verwendet es dessen Port Liste, statt die einer Kategorie.
Der erste "Inhalt" des Device-Blocks ist die Hardware ID. Nicht ändern.
Darauf folgen die eigentliche Konfiguration mit Namen *RMFC*, sie ist ein Package, eine Daten-Sammlung, mit 2 Einträgen *Package(0x02)*.Der Inhalt eines Packages ist wieder ein Block und somit von geschweiften Klammern umgeben. Ein Package hat so viele Einträge wie in der Klammer angegeben wurde. Die Einträge werden durch Kommata getrennt.
Wir werden es häufig sehen, dass zwei aufeinanderfolgende Einträge zusammengehören. Der erste Eintrag ist der Name oder eine Beschreibung der Daten, die im zweiten Eintrag folgen.
So ist der erste Eintrag des RMFC-Packages der Name der Konfiguration. In diesem Fall *8086_a12f* und der zweite Eintrag wieder ein Package, diesmal mit 4 Einträgen, den eigentlichen Daten.

Die Einträge sind

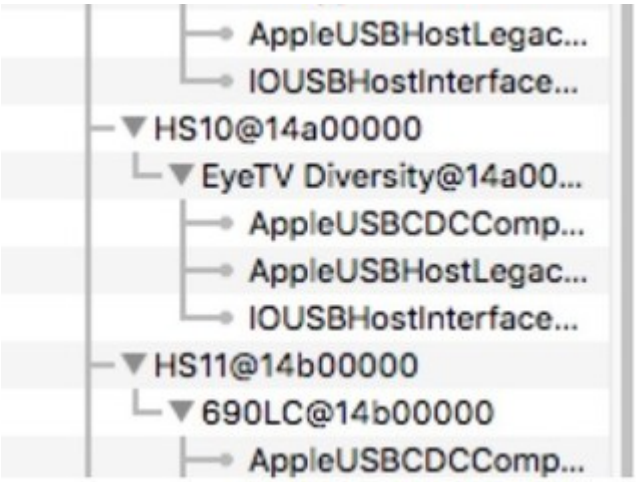
der Text *port-count* als Bezeichnung des nachfolgenden Buffers. Ein Buffer ist wie ein Package, aber jeder Eintrag besteht aus genau einem Byte.
Der Buffer hat in diesem Fall 4 Byte Länge. Das erste Byte enthält die höchste vorkommende Port-Adresse, nicht die Anzahl der Ports, wie der vorhergehende Text vermuten lässt. Die Port-Adressen werden weiter unten eingetragen und man kopiert die höchste hierher. Die anderen Bytes werden auf 0x00 gesetzt.
Der Text *ports* als Beschreibung des nachfolgenden Packages.
Das Package hat doppelt so vielen Einträge, wie Ports. In unserem Falle 13 Ports, also 26 Einträge in Hex 0x1A.
Wie schon zuvor gehören immer zwei Einträge zusammen und beschreiben einen Port. In diesem Fall jeweils ein Text und ein Package mit 4 Einträgen.
Der Text entspricht dem Namen des Ports.
Das Package enthält die Beschreibung des Ports, wie sie auch in USBInjectAll gespeichert ist.
Jedes Package hat 4 Felder, ebenfalls wieder Paare aus Bezeichnung und zugehörigen Daten.
Der erste *UsbConnector* wird von einem Byte gefolgt, dass die Art des Anschlusses angibt. Für uns sind nur 3 Werte interessant 0 (Typ-A USB 2.0 Buchse), 3 (Typ A SuperSpeed Buchse sprich USB 3.0) und 0xFF (alles andere, z.B. Header).
Der zweite *port*, ein Buffer mit 4 Bytes. Das erste Byte gibt die Port-Adresse an, die anderen Bytes werden auf 0x00 gesetzt.
Die Port Adresse kann man leicht mit IORegistry Explorer bestimmen. Man wählt links oben IOACPIPlane an und kann dann weiter unten die Port Namen gefolgt von einem @ und der Port Adresse sehen. Diese wird als Hexadezimalzahl angegeben allerdings ohne das 0x.
Nicht vergessen die höchste vorkommende Adresse weiter oben bei *port-count* einzutragen.

Speichert man die Datei oder drückt man den Compile Button, wird die Datei kompiliert.
Die häufigsten Fehler sind vergessene Kommata oder Klammern.

Fast hätte ich sie vergessen.
Die zweite Methode, die 15 Ports zu wählen. Man entscheidet sich für die Positiv oder Negativ Listen Variante. Bei der Negativ Listen Variante braucht man erst mal nichts zu tun. Bei der Positiv Listen Variante erstellt man eine SSDT wie oben beschreiben, aber mit allen vom Controller unterstützen Ports.
Nun kehrt man zum IORegistry Explorer zurück.
Entfernt man den Suchbegriff im IORegistry Explorer und scrollt bis zu den Port Einträgen, so kann man sehen welche Ports mit einem Gerät verbunden sind. Das sind die Ports, die ein Dreieck vor dem Namen und somit untergeordnete Einträge haben.

In diesem Beispiel sieht man, dass an dem Anschluss an dem EyeTV angeschlossen ist das

HS10 Port anliegt.



Rinn in die Kartoffeln, raus aus die Kartoffeln

Will man wissen welches Port an einem unbelegten Anschluss anliegt, steckt man ein USB 2.0 Gerät ein und das USB Gerät wird in IORegistry Explorer "unter" dem jeweiligen Port sichtbar.
Ein USB 3.0 Gerät wird nur erscheinen, wenn der zugehörige SuperSpeed Port schon in der Liste ist - aber wir haben ja noch kein SuperSpeed Port in der Liste, also wird auch kein USB 3.0 Gerät erscheinen, deshalb ein USB 2.0 gerät nehmen.
Haben wir in jeden Anschluss einmal ein USB 2.0 Gerät gesteckt und geschaut welches Port zu dem Anschluss gehört wissen wir welche Ports in der Liste mit einem Anschluss verbunden sind und wir müssen uns entscheiden welche wir weiterverwenden wollen. Nicht vergessen wir können maximal 15 Ports wählen und davon gehen noch die Super Speed Ports ab. Nun tragen wir die Ports, die wir nicht weiterverwenden wollen in die Negativ Liste ein bzw. löschen sie aus der SSDT. Nicht vergessen *port-count* und die Packagegröße bei *ports* anzupassen.
Dann ein Neustart. Nun sind nur noch die gewählten Ports in der Liste und die anderen wurden durch Ports mit höherer Nummer ersetzt. Dies sind nun wahrscheinlich auch SuperSpeed Ports. Nun suchen wir die Anschlüsse für die neuen Ports. Zum Auffinden der SuperSpeed Ports, muss man auch ein SuperSpeed fähiges Gerät - sprich ein USB 3.0 Gerät - haben. Dann wiederholt man die Schritte von vorher. Man entfernt die Ports, die man nicht braucht bzw. für die es keinen Anschluss gibt per Negativ oder Positiv Liste und rebootet. Man macht das solange bis keine neuen Ports mehr nachrücken, alle Anschlüsse gefunden sind, oder man 15 Ports "voll" hat.

Stop - Noch'n Beispiel

Ein TYAN S7050 Mainboard - kannte ich bis dahin auch nicht.

Wie gehabt Stick vorbereiten und starten. Und es wird das Stopzeichen erscheinen. Das bedeutet USBInjectAll findet keine unterstützte Konfiguration.
Da OS X nicht läuft, kann man auch kein IORegistry Explorer verwenden um Informationen zu sammeln.

Originelles

Die Informationen über den oder die USB Controller stehen in der DSDT und Clover ist in der Lage eine Kopie der DSDT des Boards als aml Datei zu erzeugen.
Dazu startet man mit Clover und im Clover Boot Menü drückt man die F4 Taste. Weder donnert und blitzt es, noch bleibt die Welt stehen, man merkt nicht das sich was geändert hat, aber Clover hat eine Kopie aller ACPI Tabellen im origin Ordner (EFI/CLOVER/ACPI/origin) der Clover Installation gespeichert.
Die Tabellen, werden als aml Dateien gespeichert und können mit MaciASL oder einem vergleichbaren PC Programm geöffnet werden. Uns interessiert wie gesagt die DSDT.
Ein USB Controller ist eine Device-Block, der einen Device-Block für den Root Hub enthält, der wiederum für jeden Port einen Device-Block enthält irgendwo in der DSDT. Die DSDT ist wie die oben gezeigte SSDT organisiert, nur viel, viel größer. Mehr unterschiedliche Befehle, mehr Blöcke, einfach von allem mehr. In diesem Wust müssen wir das oder die Devices finden, die den USB Controller mit seinen Ports beschreibt.

Ist denn schon Ostern ?

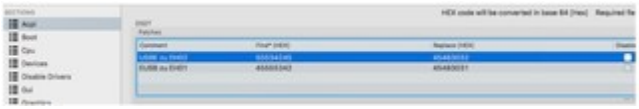
Fröhliches Suchen. Es gibt ein paar Dinge, die die Suche vereinfachen. Der Root Hub hat meist einen Namen der HUB beinhaltet. Also nach einem Device mit HUB im Namen suchen und schauen ob der umschließende Block ein USB Device sein könnte und ob der Root Hub Block Blöcke enthält, die wie USB Port Devices aussehen.
Oder man sucht nach den Ports, deren Namen oft mit SS, HS oder PR beginnen.
Suchen nach HUB findet tatsächlich zwei Devices. Beide mit Namen HUBN und umgeben von Blöcken mit dem Namen EUSB und USBE. Wenn das kein Hinweis ist, weiß ich auch nicht. Ein kurzer Blick auf die untergeordneten Devices zeigt PR30-PR37 in dem HUBN unter EUSB und PR30-PR35 in dem HUBN unter USBE.
Das Board hat also zwei USB Controller, wir wissen aus der Boardbeschreibung, dass dies alles USB 2.0 Ports sind.

Been there, seen that

Weiter oben sind die Konfigurationen gelistet, die USBInjectAll unterstützt und siehe da es gibt ein Pärchen, das 8 und 6 USB 2.0 Ports enthält.
Voraussetzung ist allerdings, dass die Devices EH01 und EH02 heißen. Also einfach EUSB in EH01 und USBE in EH02 ändern und aufs Beste hoffen.

Patch as Patch can

Man könnte nun die DSDT ändern, aber Clover bietet einen Patch Mechanismus an, der es einem in solch einfachen Fällen, erspart die DSDT anzupassen.
Dazu erstellt man zwei DSDT Patches in der Config.plist.
Man gibt jedem Patch einen Namen. Der Find Wert ist der alt Name als Hex Folge. Man findet im Netz ASCII Tabellen, die angeben welcher Hexwert, welchem Buchstaben entspricht. Man schreibt diese Werte ohne 0x hintereinander in das Find Feld. E = 0x45 U = 0x55 S = 0x53 B = 0x42 wird zu 45555342. Der Wert für das Replace Feld bestimmt man auf die selbe Art und bekommt dann die folgenden beiden Patches.



Kurz und schmerzlos

Config.plist geändert, reboot und siehe da es geht. Da die beiden Controller zusammen nur 14 Ports zur Verfügung stellen, muss man keine Negativ oder Positiv Liste erstellen und ist somit schon fertig.

Ich hoffe, dass dieser Text etwas Klarheit schafft und nicht mehr Fragen aufwirft, als er beantwortet.

PS.

Ich habe den Text mehrmals Korrektur gelesen, aber wie immer wenn man etwas liest, was man selbst geschrieben hat, entgeht einem das eine oder andere und man realisiert auch etwaige Gedankensprünge nicht.
Solltet ihr also über Fehler, Unstimmigkeiten oder scheinbar ohne Zusammenhang entstandene Dinge stolpern, PM oder Post.
Einige Dinge, wie z.B. der Aufbau einer DSDT würden den Rahmen dieses Artikels sprengen und müssen ggf. an anderer Stelle nachgereicht werden.

Beitrag von „Capitan-Paule“ vom 20. April 2016, 15:49

Hi.

Top arbeit. Danke schön.

einen kleinen Fehler hab ich gefunden, oben schreibst du Text und meinst bestimmt Kext

Beitrag von „Doctor Plagiat“ vom 20. April 2016, 16:11

@Brumbaer Ein grandioser Artikel. Vielen Dank daür. 

Beitrag von „Brumbaer“ vom 20. April 2016, 16:27

> Zitat von Capitan-Paule
>
> einen kleinen Fehler hab ich gefunden, oben schreibst du Text und meinst bestimmt Kext

Danke für das aufmerksame Lesen, aber an der Stelle soll es tatsächlich Text heißen.

Beitrag von „al6042“ vom 20. April 2016, 17:07

Hallo @Brumbaer
Vielen Dank für den tollen Beitrag....

Da habe ich ja heute Abend etwas, dass ich während dem DFB-Pokal schauen tun kann...

Beitrag von „McRudolfo“ vom 20. April 2016, 17:57

@Brumbaer

Vielen Dank für den großartigen Artikel und die Arbeit und Zeit, die du da 'reingesteckt hast!

Beitrag von „spoli“ vom 20. April 2016, 22:41

top gemacht - thx

Beitrag von „YogiBear“ vom 21. April 2016, 00:59

Schöner Artikel, danke @Brumbaer !

Wenn HS01 einen HighSpeed USB-Port benennt und SS01 einen SuperSpeed USB-Port, dann würde ich sagen, dass US01 einen "UltraSpeed"-USB-Port (also offiziell SuperSpeed+) darstellt. Da ist allerdings nur eine Vermutung, denn die DeviceID 8086:a12f bringt mich jeweils zu Z170-Mainboards, die USB3.1-Ports besitzen...

Beitrag von „Brumbaer“ vom 21. April 2016, 08:31

Die 3.1 Funktionalität auf den Z170 Boards wird durch ASMedia Chips hergestellt. Der Z170 kann kein 3.1.
Das Datenblatt weißt wie gesagt auch nur 14 + 10 Ports aus.
Inzwischen glaube ich, dass USR für USB Reserved steht.
Adresse 0x00 ist für gewöhnlich für den Root Hub reserviert.
Die HS Ports haben die nachfolgenden Adressen 0x01 bis 0x0E.
Die SS Ports genauso durchzunummerieren und dabei das erste nicht benutzte Bit (zur Unterscheidung HS, SS) zu setzen also 0x11 bis 0x1A, ist eine übliche Vorgehensweise.
Das lässt die Adressen 0x0F und 0x10 frei.
Der BIOS Schreiber hat die Ports definiert, bemerkte die Lücke beim Editieren und und beschloss, statt die Ports zu löschen, sie als Reserved zu bezeichnen. Auch das ist eine übliche Vorgehensweise um Lücken zu füllen.

Oder es hat einen ganz anderen Grund

Beitrag von „ak1848“ vom 21. April 2016, 17:44

Bekomme meine USB Ports mit Clover und El Capitan einfach nicht zum laufen, weder mit USBInjectAll noch mit diesem USB Fix: USB-Fix für El Capitan V1

Weiß ehrlich gesagt nicht mehr, wie ich weiter machen soll. Könnte sich ggf. jemand die Zeit nehmen um mal gemeinsam rüber zu schauen mit Teamviewer o.ä. ?

Beitrag von „griven“ vom 22. April 2016, 00:00

Wie schon im anderen Thread geschrieben bitte einmal eine unberührte DSDT hochladen und verraten welche SystemDefinition Du benutzt...

Beitrag von „Brumbaer“ vom 29. April 2016, 23:26

Zwei Nachträge:

Die Negativ Liste benötigt USBInjectAll Version 0.5.8 oder neuer. Habe ich oben im Artikel nachgetragen.

In diesem Thread
[Skylake] [2 Fälle gelöst] USB 3.0 Geräte gehen nicht an USB 3.0 Ports
wurden an Hand des Artikels Konfigurationen nicht als Demo, sondern für existierende Boards erstellt.
Er zeigt die Abläufe nochmal am "lebenden" Objekt.

Beitrag von „kuckkuck“ vom 10. August 2016, 01:12

Hey Brumbear!
Klasse Anleitung, Respekt! Nachdem seit kurzem mein USB nicht mehr ging, dachte ich mir benutz ich doch einfach mal kurz diese schöne Methode hier! Deswegen wollte ich dir jetzt nur kurz das HeadsUp für deine Anleitung, auch für Ozmosis geben!
Das uia_exclude Bootarg wird dabei am besten einfach in die defaults.plist gesetz und natürlich per Leerzeichen von anderen Bootargs getrennt. Das ganze kann dann zB so aussehen:

Property-List	Typ	Wert
▾ Wurzel	Dictionary	5 Schlüssel/Wert-Paare
▸ Defaults:1F8E0C...22-2B63745FA101	Dictionary	11 Schlüssel/Wert-Paare
▸ Defaults:4D1FDA...-4BCCA8B30102	Dictionary	21 Schlüssel/Wert-Paare
▾ Defaults:7C43611...0-FE41999C9FB2	Dictionary	2 Schlüssel/Wert-Paare
boot-args	String	uia_exclude=HS02;HS07;HS08;HS11;HS13;HS14;USR1;USR2;SSP1;SSP2 darkwake=4 -v
csr-active-config	Daten	4 Bytes: 67000000
Timestamp	Zahl	0
Version	String	1.0.1

Der USBInjectAll.kext, kommt wie gehabt nach S/L/E und dann funktioniert auch schon alles!
Wichtig: Nachdem die defaults.plist geändert wurde Nvram Reset machen, damit die Bootargs ausgelesen und die Änderungen angenommen werden!
Schönen Feierabend

Beitrag von „IngoG“ vom 13. September 2016, 20:22

Hallo zusammen!

Also erst mal ein großes Dankeschön an Brumbaer für die tolle Anleitung!

Ich hatte auch USB Probleme mit meinem Asus Z170-K

Ich konnte mit Hilfe der Anleitung die SSDT so anpassen, dass nun alle relevanten USB-Ports (14 Stück also auch für El Capitan geeignet) inkl. USB 3.0 funktionieren.

Das sind die Ports:
HS01, HS02, HS03, HS04, (hören zu den USB 3.0)
HS09, HS10, HS11, HS12, HS13, HS14 (2x USB Rückseite und die beiden 9-Pin Stecker auf dem Board)
SS01, SS02, SS03, SS04 (Beide USB 3, 19-Pin Stecker auf dem Board)

Wer die gebrauchen kann - einfach laden und im Pfad CLOVER->ACPI->patched auf der EFI Partition ablegen.

SSDT-USB-Z170-K.aml.zip

Gruß

Ingo

Beitrag von „griven“ vom 14. September 2016, 23:06

@IngoG danke für das Bereitstellen der SSDT wird sicher einigen helfen

---

Beitrag von „noEE“ vom 16. September 2016, 22:46

Zum Glück gibt es hier auf dem Board ja sehr kompetente Leute und vielleicht kann mir jemand bei meinen USB 3.0 Problem helfen?

Ich bekomme die 2 hinteren USB 3.0 Anschlüsse auf meinen Asus Z170-P nicht zum laufen, obwohl ich der Meinung bin, dass sie noch der Public Beta gingen. Bin jetz auf der Sierra GM und es sieht wie folgt aus:

USBInjectAll.kext befindet sich in Clover und der change 15 port limit to 30 in AppleUSBXHCIPCI Patch ist eingetragen (was bisher funktionierte). Stecke ich meine externe HDD an, leuchtet sie entweder grün für 2.0 oder blau für 3.0 aber es passiert nichts. An den anderen Anschlüssen leuchtet sie allerdings grün. Starte ich mit angeschlossener HDD leuchtet sie dauerhaft blau, wird aber nicht erkannt und beim umstecken leuchtet sie wieder gar nicht.

Die SSDT von IngoG funktioniert bei mir nicht.

Meine SSDT hänge ich mal mit an. Ich muss zugeben, dass ich beim Thema DSDT und SSDT nicht ganz so firm bin.

SSDT.aml

Beitrag von „IngoG“ vom 19. September 2016, 23:19

Hallo noEE!

Sorry, habe deinen Beitrag erst eben gesehen!
Meines Erachtens hat das nichts mit deiner angehängten Datei zu tun.
Hast du mal im IORegistry Explorer im Suchfeld die Namen der USB-Devices

anzeigen lassen? (XHC eintippen).

Daran kannst du prüfen ob alles korrekt angesprochen wird. Die USB auf unseren Boards sollten identisch sein.
Zusätzlich zu meiner Datei im patched Ordner muss in dem kexts/Other Ordner die GenericUSBXHCI.kext (bei mir Version 1.2.11) und die USBInjectAll.kext (0.5.12) hinterlegt sein.

Vielleicht hat sich zu Sierra aber auch wieder etwas verändert...

Gruß

Ingo

EDIT:

Mir ist aufgefallen, das das Z-170P nicht die beiden USB 3.1 Anschlüsse hat - deine "Nummerierung" der Ports müsste daher von meinen abweichen - wie schon geschrieben mit dem IORegistryExplorer überprüfen!

Beitrag von „jemue“ vom 3. März 2017, 01:58

Danke für den schönen Artikel!

Ich habe es nun teilweise zum Laufen gebracht, aber das mit der SSDT bekomm ich nicht hin. 🙁

Ich habe die Ports identifiziert, die ich brauche: HS01 - HS06, HS09, HS10, HS13, SSP1 - SSP6
D.h. ich brauche nicht: HS07, HS08, HS11, HS12, HS14

Mit USBInjectAll und uia_exclude zeigt mit der IORegistryExplorer alles korrekt an.

Nun habe ich mir eine SSDT erstellt und habe nur noch die 15 Ports drin, die ich brauche. Soweit so gut. Gespeichert unter SSDT.aml (siehe Anhang) und nach /Efi/CLOVER/ACPI/patched kopiert.
Ich habe Clover v4012 und quasi eine Vanilla Config. Keine DSDT, nichts umgestellt.

Nach dem Neustart sehe ich dann wieder HS01 - HS14 + SSP1 ☹

Ich gehe also stark davon aus, dass Clover die SSDT nicht lädt. Aber warum? Muss ich ihm irgendwo noch sagen, dass er das tun soll?

Ansonsten hab ich hier und da mal gelesen, dass man die Datei "SSDT.asl" (statt .aml) nennen soll, aber auch das hat nichts geändert.

Hat jemand eine Idee?

Danke schon mal!

EDIT:
Im Boot Log steht sogar folgendes:

```
5:312  0:000  ===  [  ACPIPatchedAML  ]  ====================================5:312
0:000 Unsorted5:312 0:000 Inserting SSDT-1.AML from EFI\CLOVER\ACPI\patched ...
Success
```

Aber irgendwie macht er es wohl trotzdem nicht :?

Beitrag von „Doctor Plagiat“ vom 6. März 2017, 18:05

Vielleicht stimmt ja mit der SSDT irgendwas nicht, aber die benötigst du doch nicht. Du musst nur die exclude-Liste im Clover-Configurator unter "Boot" eintragen.

Aber bitte nicht meine Liste auf dem Bild abchreiben. Du hast ja selber schon festgestellt, das du HS07, HS08, HS11, HS12, HS14 nicht brauchst.

Beitrag von „All the pugs!“ vom 6. März 2017, 18:44

Portnummern stimmen? Ich füge die SSDTs immer unten in Clover bei SortOrder hinzu. Aber wenn im Log schon steht, daß er sie findet, weiß ich nicht, ob man das überhaupt braucht.

Beitrag von „jemue“ vom 6. März 2017, 22:40

Doctor: Ich hatte ja schon gesagt, dass es mit "uia_exclude" wunderbar funktioniert.

@All the pugs! : soweit ich weiß, braucht man SortOrder nur, um die Reihenfolge explizit festzulegen. Wenn man das nicht macht, sollte Clover sie einfach alphabetisch laden.

Was ich schon herausgefunden habe:
Oben im Beispiel steht "8086_a12f" . Das trifft wohl auf Skylake Chipsätze zu. Bei mir muss es wohl "8086_8c31" sein (Z87 - Haswell). Das war definitiv ein Fehler. Aber auch damit erhalte ich dasselbe Ergebnis ☹️

Beitrag von „maschinenwart“ vom 7. März 2017, 12:50

> Zitat von jemue
>
> Ansonsten hab ich hier und da mal gelesen, dass man die Datei "SSDT.asl" (statt .aml) nennen soll, aber auch das hat nichts geändert.

...einfach nur umbenennen und speichern funktioniert nicht. Du musst die angepasste Vorlage mit maciASL "kompilieren"...

- Die SSDT-UIAC-ALL.dsl mit maciASL öffnen
- Die Vorlage anpassen
- Die angepasste SSDT-Vorlage speichern:
  - Save As > „SDT-UIAC-ALL“
  - Format: ACPI Machine Language Binary (.aml)
- Vorzugsweise auf dem Desktop speichern, damit du sie leicht findest
- Dann die fertige SSDT inden patched-Ordner verschieben

Beitrag von „al6042“ vom 15. März 2017, 19:11

@jemue
Ich nutze auf meine Z87X-Board keinen USBInjectAll, sondern die FakePCIID/FakePCIIDXhciMux-Kombi und fahre sehr gut ohne ein und ausklammern von USB-Ports.
Achte aber auf en KextsToPatch-Eintrag zur Deaktivierung des 15-Port-Limits.

Beitrag von „joshi1999“ vom 3. Mai 2017, 19:37

Ich wusste, ich habe etwas vergessen, naja das Wochenende kommt bestimmt! 

Wird einem erst bewusst wenn man vorne die USB Anschlüsse nutzen möchte und dies

irgendwie nicht funktioniert 😄

Danke für den Post 🙂

Beitrag von „AirArt“ vom 27. April 2020, 15:00

Brumbaer ich grabe das Thema hier mal hoch. Kann ich deine SSDT aus dem Beispiel nehmen (fahre immer noch auf Z170I), als .aml kompilieren, dann bei opencore in den ACPI Ordner rein, config.plist entsprechend anpassen und bei Kernel - Quirks - XhciPortLimit auf false setzen?

Edit: Hat genau so funktioniert 😆