Еще уточнение. Оказывается, БИОС, прошитый в РОМ карточки не совпадает с тем, что формируется на адресе 0xc0000 — тень рома. Так вот, нам нужен именно он, теневой, а не тот БИОС, который прожигаем программатором.
Короче. Для мобильных радеонов ставим Yes, хотя никакого файла нет, для остальных карточек No. Других вариантов история не зафиксировала.
А вот и наступили новые времена. Для компьютеров с UEFI-only БИОСом, на легаси-адресе нет никакого ВидеоБиоса. Подкладываем в файле и ждем новых решений. Или... и так работает?

```
<key>DualLink</key>
<integer>0</integer>
```

По умолчанию инжектируется значение 1, но для некоторых старых конфигураций этот параметр=1 приводит к учетверению экрана. Помогает установка в 0, как в приведенном примере.

```
<key>BootDisplay</key>
<integer>1</integer>
```

Указывает, какой из дисплеев является основным. Именно он будет зажигаться при старте и пробуждаться после сна. Обычно это номер 0, но иногда выходы нумеруются не в том порядке, смотрите в иореге, как у вас. Ревизия 3399.

```
<key>PatchVBios</key>
<true/>
```

Кловер вносит исправление в тень ВидеоБиоса по адресу 0xC0000, чтобы он поддерживал тот видеорежим, который является максимальным для подключенного монитора. Например, в EDID монитора есть мода 1920x1080, а в ВидеоБиосе такой нету. Кловер пропишет ее в качестве первой моды и запустит в использование. Если монитор сам не формирует EDID, его можно инжектировать, как показано ниже.
Были случаи, когда включение этого патча приводило к панике, черный экран при попытке загрузиться. Для первой попытки выключите этот параметр.
Либо для патча используется значение из файла config.plist

```
<key>GUI</key>
<dict>
   <key>ScreenResolution</key>
   <string>1440x900</string>
```

Если автоматика ошиблась, можно прописать патч ВидеоБиоса вручную, по стандартному алгоритму Найти/Заменить.

```
<key>PatchVBiosBytes</key>
<array>
   <dict>
      <key>Find</key>
      <data>gAeoAqAF</data>
      <key>Replace</key>
      <data>gAeoAjgE</data>
   </dict>
</array>
```

Можно в одном БИОСе сделать несколько патчей 0,1,2... К примеру, успех с Нвидия пришел при четырех патчах.
Этот пример из ВидеоБиоса ATIRadeon HD6670, заменяющий моду 1920x1440 на более приемлимую 1920x1080. При таком способе следует выбирать моду с такой же горизонталью.
Для успешного выставления полного разрешения экрана в интерфейсе Кловера, а потом и в системе, если используется безопасный режим (без видеодрайверов), необходимо иметь EDID. Для этого введена группа параметров

```
<key>EDID</key>
<dict>
```

В этом словаре такие пункты

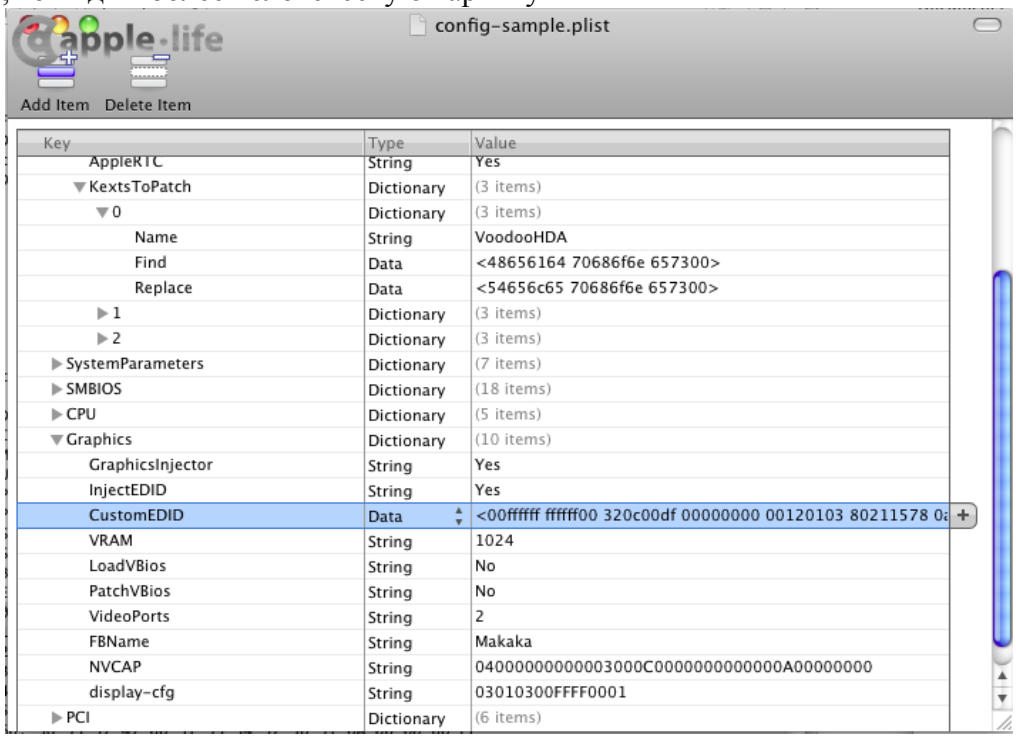
```
<key>Inject</key>
<true/>
```

1. Существуют мониторы без DDC, например панели ноутбуков.
2. Существует варианты, когда DDC есть, но Эппловские драйвера его не видят.

На второй вариант мы ставим просто Inject=yes, и Кловер сам извлечет EDID , и подсунет его драйверам. Необходимость такого действия подмечена в теме про мобильные радеоны. Для компьютеров UEFI-only считаем первый вариант, поскольку Кловер не извлечет ВидеоБиос.
На первый вариант мы еще должны прописать новый EDID ручками. Пишем так:

```
<key>Custom</key>
  <data>AP///////wAyDADfAAAAAAASAQOAIRV4CunVmVlTjigm
UFQAAAABAQEBAQEBAQEBAQEB3iGgcFCEHzAgIFYAS88QAAAY3iGgcFCEHzAgIFYAS88QAAAAAAA/
gBXNjU3RwAxNTRXUDEKAAAA/gAjMz1IZYSq/wIBCiAgAJo=</data>
```

Кстати, искусственно подсунутый EDID будет также использован для автоматического патча видеобиоса. См.выше.
Откуда взять EDID? Ну раз из этого компа он никак не извлекается, то берем чужой. Главное условие - чтобы было правильное максимальное разрешение.
В свой образец конфиг-плиста я ставлю EDID от Делла Инспирон. Матрица 1440x900.
Буковки в этом примере — это стандартная шифровка XML, если же смотреть через PlistEditor, то видим более человеческую картину



Еще вариант изготовления ЕДИДа — воспользоваться программой ViewSonic EDID Editor (версия 3.1.5), которая, при желании, легко портируется в OSX. Есть и OSX редакторы, например AW_EDID_Editor. Но это уже не касается собственно Кловера. Изучайте теорию.
Кловер дает вам возможность инжектировать свой EDID, хороший, качественный.
Есть также сведения, что эппловские драйвера проверяют изготовителя, поэтому был придуман патч, чтобы автоматически извлеченный ЕДИД поправлялся на Эппл.

```
<key>ProductID</key>
<string>0x9221</string>
<key>VendorID</key>
<string>0x1006</string>
```

Не могу сказать, чтобы это кому-то в чем-то помогло. Скорее наоборот, подставил значения, и яркость перестала регулироваться.
Новые ключи

```
<key>HorizontalSyncPulseWidth</key>
<string>0x11</string>
```

Есть такой параметр в спецификации EDID, прописан в байтах 63 и 65 в секции Detailed Timing. Некие хакеры обнаружили, что параметр влияет на известную проблему восьми яблок. Ищите объяснения на форумах, кому чего удалось добиться с изменением этого параметра. Да, влияет, и даже очень!

```
<key>VideoInputSignal</key>
<string>0x80</string>
```

Второй параметр от тех же хакеров. Это байт 0x14 в EDID, определяет свойства коннектора.
бит 7 - аналоговый=0 или цифровой=1.
биты 6 - 1 определены только для аналоговых сигналов.
бит 0 для цифрового означает, что сигнал совместим со стандартом VESA DFP 1.x

```
<key>VideoPorts</key>
<integer>2</integer>
```

Количество видеовыходов на карте, включая TVO и/или HDMI. Выбранный фрейм из Эппловского списка может не соответствовать нашей реальной карте. Влияет на количество инжектированных коннекторов. Может помочь борьбе с мнимыми мониторами.

```
<key>FBName</key>
<string>Makaka</string>
```

Этот параметр специфичен для ATI Radeon, к которым имеется три десятка разных фреймбуферов без какой-либо закономерности кому что. Кловер автоматически выбирает из таблицы на большинство известных карточек наиболее подходящее имя. Однако, другие пользователи точно такой же карточки оспаривают, им нужно другое имя. Вот и напишите в этот параметр то, что вам кажется наиболее правильным. Общее правило: не знаете, что писать, вообще сотрите параметр.
Но не пишите уж эту макаку! Специально прописал для абсурда – нет, все равно копируют в свой конфиг!
Есть такая мысль, что вся разница в этих фреймах и заключается в наборе коннекторов, а поскольку вы и так их собираетесь патчить, то никакой разницы, который вы возьмете за основу. Разве что, фрейм должен соответствовать семейству видеокарт. Например Wormy не будет работать с Радеоном 6670.

```
<key>RadeonDeInit</key>
<true/>
```

Этот ключ работает с картами ATI/AMD Radeon 6xxx и выше. А может и 5xxx, не видел отзывов. Он исправляет содержимое регистров GPU таким образом, что карта становится правильно инициализированной, и драйвера MacOSX работают с ней как нужно. Карточка включается при старте, и при пробуждении после сна. Спасибо vit9696 и Mieze.

```
<key>NVCAP</key>
<string>0400000000000003000C0000000000000A00000000</string>
```

Это параметр для видеокарт NVidia, конфигурирует типы и назначения видеопортов. В этой строке 40 шестнадцатеричных цифр заглавными буквами. Теория здесь отсутствует, есть эмпирика, да еще и с противоречивыми результатами. Есть вот такая табличка, но ее правильность оспаривается.

Yellow = Desktop 1 (primary)	Green = Desktop 2 (secondary)

DVI+VGA

	TV	DVI 2	VGA 2	VGA 1	Bin string	Conversion to hex
Desktop 1	0	0	0	1	1	1
Desktop 2	1	1	1	0	1110	0e
NVCAP	TV + DVI + VGA2			VGA1	04000000 00000100 0e000000 00000007 00000000	

DUAL DVI

	DVI 2	VGA 2	DVI 1	VGA 1	Bin string	Conversion to hex
Desktop 1	0	0	1	1	11	3
Desktop 2	1	1	0	0	1100	0c
NVCAP	DVI2+VGA2		DVI1+VGA1		04000000 00000300 0c000000 00000007 00000000	

DUAL DVI + TV on hardware channel 2 (maybe not supported)

	TV	DVI 2	VGA 2	DVI 1	VGA 1	Bin string	hex
Desktop 1	0	0	0	1	1	11	3
Desktop 2	1	1	1	0	0	11100	1c
NVCAP	DVI2+VGA2+TV			DVI1+VGA1		04000000 00000300 1c000000 00000007 00000000	

DUAL DVI + TV on hardware channel 1 (maybe not supported)

	DVI 2	VGA 2	TV	DVI 1	VGA 1	Bin string	hex
Desktop 1	1	1	0	0	0	11000	18
Desktop 2	0	0	1	1	1	111	7
NVCAP	DVI2+VGA2		DVI1+VGA1+TV			04000000 00001800 07000000 00000007 00000000	

The hardware channel and desktops are intentionnaly swapped so that TV out stays in use for secondary desktop.

DUAL DVI + TV on hardware channel 1 & 2

	TV2	DVI2	VGA2	TV1	DVI1	VGA1	Bin string	to hex
Desktop 1	0	0	0	1	1	1	111	7
Desktop 2	1	1	1	0	0	0	111000	38
NVCAP	TV2+DVI2+VGA2			TV1+DVI1+VGA1			04000000 00000700 38000000 00000007 00000000	

Laptop with VGA + TV out

	TV	DVI 2	VGA 2	LVDS	Bin string	Conversion to hex
Desktop 1	0	0	0	1	1	1
Desktop 2	1	0	1	0	1010	5
NVCAP	External VGA+TV			LCD	04000000 00000100 05000000 00000007 00000000	

Laptop with DVI + TV out

	TV	DVI 2	VGA 2	LVDS	Bin string	Conversion to hex
Desktop 1	0	0	0	1	1	1
Desktop 2	1	1	1	0	1110	0e
NVCAP	External DVI+VGA+TV			LCD	04000000 00000100 0e000000 00000007 00000000	

Первый байт всегда 04 (в МакБуке 05!). Второй байт LID=01 для ноутбуков.
На форумах можно найти другие способы вычисления правильного значения этой строки. А Кловер и сам пытается вычислить из БИОСа.

```
<key>display-cfg</key>
<string>03010300FFFF0001</string>
```
Это тоже параметр только для карт NVidia. Подробности смотрите в обсуждениях http://www.insanelymac.com/forum/topic/215236-nvidia-injection/

Однако, сведения, приведенные там являются спорными. Реальные конфиги можно посмотреть в теме ~~http://www.projectosx.com/forum/index.php?showtopic=370~~ — А вообще-то, конфиг по-умолчанию, который создает Кловер, похоже и является лучшим вариантом. Просто не указывайте вообще этот параметр, дайте возможность Кловеру его вычислить.

```
<key>NvidiaGeneric</key>
<true/>
```
Если тру, то вместо имени Gigabyte Geforce 7300LE
будет использовано имя NVIDIA Geforce 7300LE
Зачем - не знаю. Может кому-то нужны реальные данные, а кому-то похожее на нативника.

```
<key>NvidiaSingle</key>
<false/>
```
Из той же серии непонятных патчей. Если стоит тру, то инжектировать только первую карточку, вторую нет.

```
<key>NvidiaNoEFI</key>
<false/>
```
Добавляет к инжекту Нвидии свойство `NVDA,noEFI`
Объяснения от FredWst http://www.insanelymac.com/forum/topic/306156-clover-bugissue-report-and-patch/page-107?p=2443062#entry2443062
Утверждает, что без этого на экране артефакты на его GT640.

```
<key>ig-platform-id</key>
<string>0x01620005</string>
```
Этот параметр необходим для запуска видеокарточки Intel HDxxxx, спор о конкретных значениях не привел к единому правилу, поэтому параметр просто вынесен в конфиг — подбирайте. Кстати, Кловер и сам предложит некое значение.
Теперь и у меня есть результат. Мой Skylake запустился только с параметром 0x193b0000, он соответствует конфигурации, в которой есть HDMI выход, как у меня.

KernelAndKextPatches

Эта группа параметров для осуществления бинарных патчей на лету. Надо заметить, что это осуществимо, только если загрузка происходит через kernelcache либо через параметр `ForceKextsToLoad`. Если kext не загрузился, и не присутствует в кеше, то эти фиксы не работают. Начиная с версии 5119 патчи происходят по внутренним алгоритмам, которые не зависят от версии системы. А для своих патчей сделана возможность поиск по символам.

```
<key>Debug</key>
<true/>
```
Если вы захотите понаблюдать на ходом, как происходит патч кекстов. Вообще-то, этот ключ для разработчиков.

```
<key>KernelCpu</key>
<true/>
```

Предотвращает панику ядра на неподдерживаемом ЦПУ, в частности Yonah, Atom, Haswell для старых систем. Или какой-нибудь Broadwell-E. Нынче этот патч следует считать устаревшим, и применять вместо него FakeCPUID и другие патчи ядра.
Нужно понимать, что в ядре есть и другие алгоритмы, которые будут неправильно работать с неподдерживаемым ЦПУ, поэтому не ждите, что этот патч решит все ваши проблемы. Очень сомнительно, что это будет работать с Pentium M, Pentium 4 или AMD, для таких случаев лучше все же найти специально сделанное ядро.

```
<key>FakeCPUID</key>
<string>0x010676</string>
```

Этот патч, введенный с ревизии 2748, служит для замены KernelCpu. Он не просто блокирует панику ядра, он подменяет ИД процессора, чтобы во всех вызовах откликался как поддерживаемый. В частности, он влияет и на кекст AppleIntelCPUPowerManagement.kext. В данном примере он подставляет ИД процессора Пенрин, который поддерживается всеми версиями OSX начиная с 10.5, вплоть до 10.14, в котором ему пришел конец.

```
<key>AppleIntelCPUPM</key>
<true/>
```

Оказывается, БИОС на материнских платах АСУС (который раз нам АСУС портит настроение?) пишет в MSR регистр 0xE2 бит 14, и регистр становится ReadOnly, но он используется в кексте **AppleIntelCPUPowerManagement**, причем используется по записи. Авторы этого фикса не придумали ничего лучшего, как исправить сам кекст, ибо вернуть регистру E2 его былую функциональность можно только перезагрузкой.
Ставьте Yes, если при старте системы вы имеете панику на этот кекст. (да, регистр E2 имеет свойство WriteOnce, т.е. записать в него можно только один раз до перезагрузки). Актуально для процессоров Sandy и выше. Либо перепрошивайте БИОС. А как другие операционки в этом случае? Говорят, что и для Виндоус это хорошо.

```
<key>AppleRTC</key>
<true/>
```

~~Операционная система OSX как-то не так работает с CMOS, как это предусмотрено BIOSом, в результате при пробуждении из сна или при перезагрузке происходит сброс CMOS. Не у всех, больше в этом грехе замечены платы от Gigabyte. Более того, часто эта проблема решается просто патчем DSDT: Device(RTC) что делает и Кловер.~~
~~Однако, в некоторых случаях и этот патч не помогает. Тогда можно поправить сам кекст AppleRTC, что здесь и делается.~~ Устарело! vit9696 исследовал проблему, и поправил в Кловере операции с RTC, теперь рекомендуемое значение ключа <false/>, поскольку это влияет на гибернацию. Хотя, спорный вопрос, у меня ключ гибернации все равно сохраняется в NVRAM.

```
<key>KernelLapic</key>
<false/>
```

На ноутбуках HP есть проблема с lapic, которая решается запуском с cpus=1, или теперь с этим патчем <true/>. Проблема и решение существует очень давно, со времен Хамелеона, но пока новые разработчики не добрались до ноутбуков HP, чтобы разобраться серьезнее, в чем тут фокус.

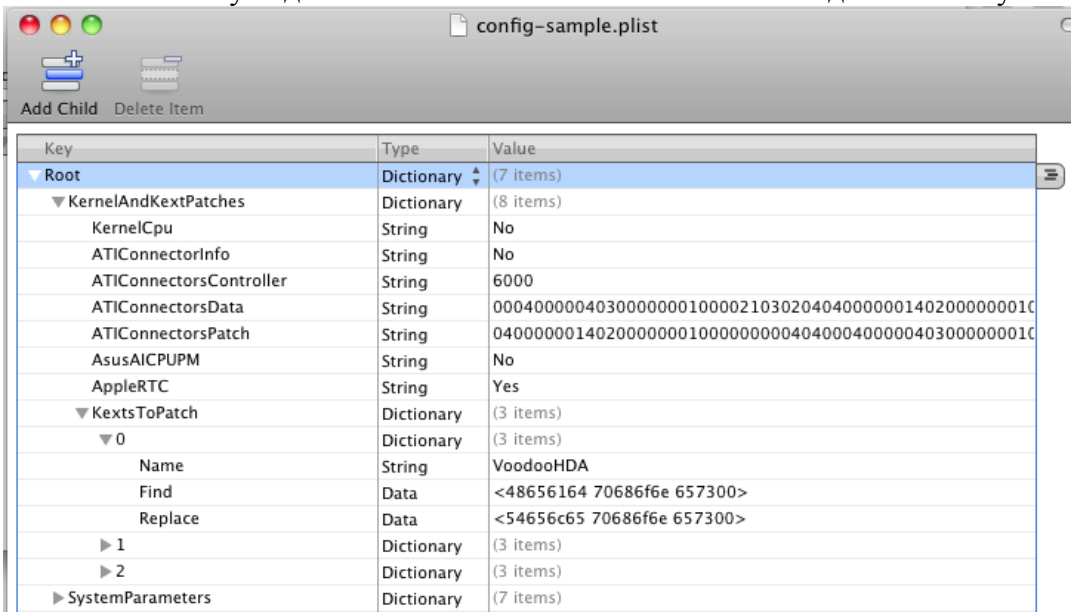
```
<key>KernelPM</key>
<false/>
```

Оказывается, начиная с системы 10.9 есть некоторое управление CPUPM заложенное прямо в ядро. Этот патч предотвращает панику ядра, для тех случаев, когда 0xE2 залочен в БИОСе.

```
<key>KernelXCPM</key>
<false/>
```
В системе 10.12+ прекращена поддержка XCPM для процессоров IvyBridge. Ничего страшного, можно пойти эппловским путем, ну а кто привык к XCPM, может поставить этот ключ в <true/>.

```
<key>DellSMBIOSPatch</key>
<true/>
```
Замечено, что на ноутбуках Dell с процессором Skylake и выше, сам UEFI BIOS портит наш готовый SMBIOS. Непонятно зачем, но с этим нужно бороться. Патч хитрый, вручную так не сделаешь. Нужен только на ноутбуках Делл с процессором Skylake (и выше?).

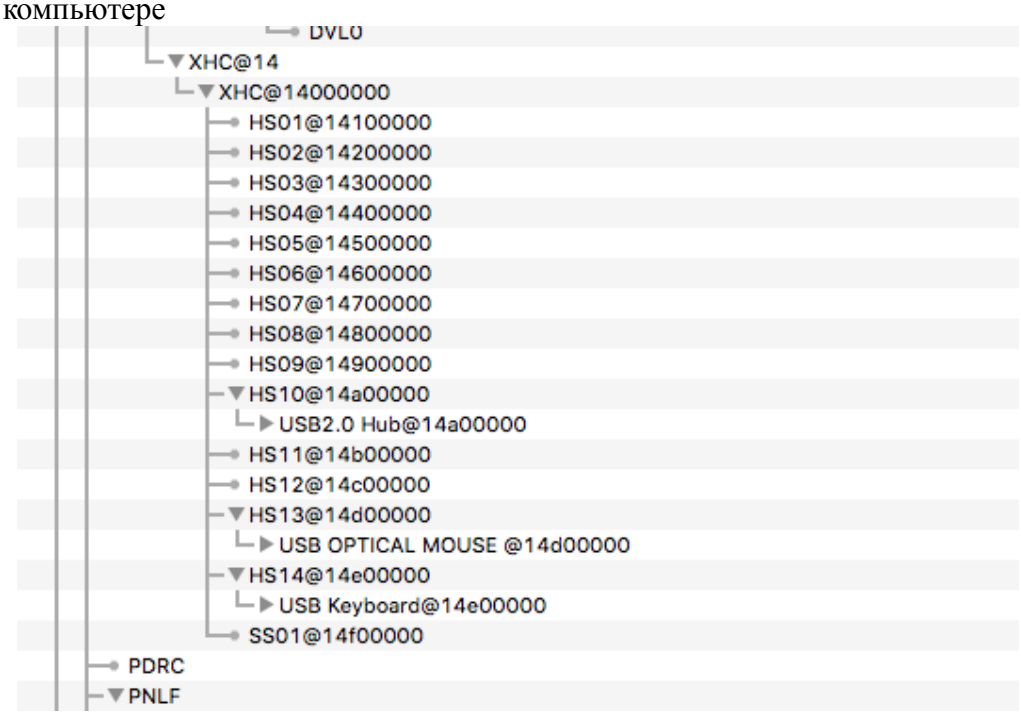
```
<key>KextsToPatch</key>
<array>
```
Помимо специфических патчей, можно сделать патч любого другого кекста, принцип простой: 16 ричная строка, что искать, и строка, на что заменить. Начиная с ревизии 5095 можно делать патчи по маске, смотрите параграф Patching with Mask.
Образец: патчим VoodooHDA на предмет замены названия Headphones на Telephones. Условие — количество букв должно быть таким же. Либо меньше и дополнить нулями.



Этот метод успешно применяется для включения поддержки Trim для SSD
http://www.applelife.ru/threads/clover.32052/page-539#post-310105
Вот еще один очень полезный патч: борьба с желтыми иконками и нерабочим DVD проигрывателем (который не работает для внешних приводов):
Оригинальная тема http://www.applelife.ru/threads/Меняем-external-на-internal.38111/

```
<dict>
   <key>Name</key>
   <string>AppleAHCIPort</string>
   <key>Find</key>
   <data>RXh0ZXJuYWw=</data>
   <key>Replace</key>
   <data>SW50ZXJuYWw=</data>
</dict>
```
Чтобы выбрать модель MacPro4,1 или 5,1, не имея памяти с ECC. AppleTyMCEDriver patch
Замечен и более простой метод **-nehalem_error_disable disables the AppleTyMCEDriver** Спасибо AkimoA.
Один из полезнейших патчей - снятие ограничения на количество портов в контроллере USB3. Проблема в том, что кекст нумерует все порты, сначала как USB2, затем как USB3, и

общая сумма по понятиям Эппл не должна превосходить 15 портов. Результат иллюстрирую на своем компьютере



То есть, пересчитав 14 портов HS (usb2), он принял только один порт SS (usb3), тогда как по спеку чипсета их 10.

Патч следующий (зависит от версии системы)

```
<dict>
   <key>Find</key>
   <data>g32UDw+DlwQ=</data>
   <key>Comment</key>
   <string>USB 3.0 limit High Sierra 10.13.4</string>
   <key>Disabled</key>
   <false/>
   <key>MatchOS</key>
   <string>10.13</string>
   <key>Name</key>
   <string>com.apple.driver.usb.AppleUSBXHCI</string>
   <key>Replace</key>
   <data>g32UGA+DlwQ=</data>
</dict>
```

Образец, кстати, иллюстрирует дополнительные ключи патча:

```
<key>Disabled</key>
<true/>
```

Можно отключать в конфиге сомнительные ключи, чтобы потом их включить в интерфейсе Кловера.

```
<key>Comment</key>
<string>USB 3.0 limit High Sierra 10.13.4</string>
```

Комментарий тоже виден в интерфейсе Кловера, но не оказывает действия на систему.

```
<key>MatchOS</key>
<string>10.13</string>
```

Часто бывает, что данный патч применим только к конкретной версии системы, для другой версии нужен другой патч. Пишем оба, и проставляем версию, в том числе полную 10.13.5 или укороченную, как в примере. Впрочем, у вас вряд ли будут две системы 10.13.2 и 10.13.4, поэтому в полной версии смысла особого нет, просто не забывайте обновлять патчи вместе с апдейтом системы.

Бывает необходимость править не бинарную часть кекста, а его info.plist. В этом случае секция выглядит следующим образом

```
<dict>
   <key>Name</key>
   <string>AppleHDAController</string>
   <key>Comment</key>
   <string>Patch_to_not_load_this_driver</string>
   <key>InfoPlistPatch</key>
   <true/>
   <key>Find</key>
   <string>0x04020000</string>
   <key>Replace</key>
   <string>0x44220000</string>
</dict>
```

Здесь есть одно осложнение. Патч предполагается делать в кернелкэше, но, если мы делаем патч инфо-плиста, чтобы кекст грузился, там этого кекста еще нет, поскольку он еще не загрузился. Поэтому загружаться нужно дважды. Первый раз с игнорированием кэша (~~ключ NoCache~~), тогда FSInject загрузит этот кекст, и второй раз уже с кешем, где он и будет успешно патчиться. Поставить в конфиге ForceKextsToLoad.
В ревизии 3154, и затем в 3256 патч инфо-плиста поправлен (спасибо юзеру solstice). Теперь в поиск можно включать несколько строк, исключив все невидимые символы, такие как перевод строки и табуляцию. Задавать поиск теперь нужно в виде **<data>**, ибо служебные символы, такие как "<", невозможно задать в текстовом виде. Длины строк поиска и замены могут отличаться, но задавать необходимо одинаковую длину, дополнив пробелами. Пример

```
<dict>
   <key>Comment</key>
   <string>Power state 1 - 0</string>
   <key>Name</key>
   <string>AppleIntelHDGraphicsFB</string>
   <key>InfoPlistPatch</key>
   <true/>
   <key>Find</key>
   <data>PGtleT5Qb3dlclN0YXRlczwva2V5PjxpbnRlZ2VyPjE8L2ludGVnZXI+</data>
   <key>Replace</key>
   <data>PGtleT5Qb3dlclN0YXRlczwva2V5PjxpbnRlZ2VyPjA8L2ludGVnZXI+</data>
</dict>
```

Если у вас есть образец конфига со множеством патчей, но в конкретном случае применяете только некоторые, то лишние можно запретить

```
   <key>Disable</key>
   <true/>
```

Это было очень удобно осуществлять программой Property List Editor, в которой <true/> представлена просто галочкой. В ревизиях Кловера 3990+ эти патчи можно разрешать и запрещать в меню Кловера, ставя эту галочку.

Patching with Mask

Начиная с ревизии 5095 введена возможность осуществлять бинарные патчи с маской.
Это касается KextPatches, KernelPatches и BootPatches
Расскажу в отдельном месте, имея ввиду все три пункта.
Выглядит это так (конкретные данные нереальные, просто как метод.
Итак, кроме шестнадцатеричной строки Find мы можем задать еще и маску MaskFind, битовую. Если какой-то бит = 1, то ищем точное соответствие, если =0, то игнорируем разницу.
А для строки Replace маска MaskReplace означает, что бит=1 — делаем замену, бит=0 — оставляем как было.
Пример:

1. Найти все строки в указанном кексте, {также работает и в ядре, или в boot.efi} clever или Clever. Разница в первой букве, она отличается на бит 0x20. То есть задаем маску поиска DF FF FF FF FF FF. Это значит, что мы добиваемся соответствия всех байтов (букв), кроме первого, в котором игнорируем большая буква или маленькая. Маску поиска можно сократить, ибо по-умолчанию все недостающие байты предполагаются FF. Это означает, что неуказанные байты обязательно должны совпадать
2. Заменить в найденных словах третью букву на "o", то есть получим clover или Clover соответственно.
MaskReplace = 00 00 FF 00 00 00
Строку можно сократить справа, ибо предполагается заполнение нулем. При этом те байты, которые мы не заменяем, можно было бы и в Replace не указывать, но там требование точного соответствия длины.
Для сохранения обратной совместимости нового Кловера со старым конфигом предполагается, что неуказанная маска (отсутствующая) вся состоит из FFFFFF, то есть точное соответствие строке поиска, и полная замена всех байтов на указанные.

Символьный патчинг.

Начиная с ревизии 5119 мы имеем больше возможностей поиска и патча.
Общий синтаксис

```
<dict>
        <key>Comment</key>
        <string>Symbolic patch example got lapic panic</string>
        <key>MatchOS</key>
        <string>All</string>
        <key>Disabled</key>
        <true/>
        <key>Procedure</key>
        <string>_lapic_interrupt</string>
        <key>RangeFind</key>
        <integer>200</integer>
        <key>StartPattern</key>
        <data>ACnHeAAx241H+oM=</data>
        <key>MaskStart</key>
        <data>/////wA=</data>
        <key>Find</key>
        <data>6AAA//+DAAAAAAAA</data>
        <key>MaskFind</key>
        <data>/wAA////AAAAAP//</data>
        <key>Replace</key>
        <data>6AAA//8xwJCQkJCQ</data>
        <key>MaskReplace</key>
        <data>/wAA////////////</data>
</dict>
```

MatchOS ставим All, поскольку считаем этот метод патчинга не зависящем от версии системы.
Disabled пока true, поскольку это нереальный пример.
Procedure здесь пишем название процедуры, которую ищем. Реальное имя может быть длиннее, но сравнение идет по наличию подстроки. Будьте уверены, что такая подстрока встречается только в этой процедуре.
RangeFind длина кодов для поиска. В общем случае просто размер этой процедуры, или меньше. Таким способом мы ускоряем поиск, не перебирая всех миллионов строк.
StartPattern была изобретена до символьного патча. Это стартовая точка, откуда искать наш паттерн. Если знаем название процедуры, то StartPattern уже вряд ли нужна. Тем не менее пусть будет. К нему также применимо **RangeFind**.
MaskStart это маска для стартовой точки, то есть для **StartPattern**.
И далее пары **Find**/**MaskFind** и **Replace**/**MaskReplace**.

Начиная с ревизии 2814 появилась возможность принудительной подгрузки кекстов

```
<key>ForceKextsToLoad</key>
<array>
  <string>\System\Library\Extensions\AppleHDA.kext</string>
</array>
```

Таким образом преодолевается нежелание кекстов грузиться. Необходим для подгрузки IOXXXFamily, который необходим для загрузки основного кекста, но он зависит от этой фамилии. К примеру, IONetworkFamily.
Или даже целую папку \Extra\Extensions (ревизия 2816+). Подразумеваются папки на основной партиции, другие партиции/тома/диски не предусмотрены.
Обратите внимание на наклон слеша! Для выполнения этой функции необходим драйвер FSInject.efi.

```
<key>ATIConnectorsController</key>
<string>6000</string>
```

Для полноценного запуска карточек ATI (AMD) Radeon 5000 и 6000 серий недостаточно инжектировать свойства в реестр, необходимо еще подкорректировать коннекторы в соответствующем контроллере. В данном случае указываем на 6000 контроллер. Следующие два свойства указывают, что найти, и на что изменить.

```
<key>ATIConnectorsData</key>

<string>00040000040300000001000021030204040000001402000000010000000004031000000010000000
0001000000000001</string>
```

```
<key>ATIConnectorsPatch</key>

<string>040000001402000000010000000004040004000004030000000100001102010500000000000000000
0000000000000000</string>
```

Этот метод работает только для систем 10.7 и выше.
В 10.12 коннекторы будут другие, поэтому этот метод следует считать устаревшим, хотя методика вычисления все та же.
Расскажу подробнее, как получить эти цифры.
Оригинальная статья от bcc9
http://www.insanelymac.com/forum/index.php?showtopic=249642
Полный рецепт от Xmedik на русском языке с обсуждениями
http://www.applelife.ru/threads/Завод-ati-hd-6xxx-5xxx-4xxx.28890/
Здесь изложу короче, с учетом специфики Кловера.

1. Прежде всего, надо получить свой видеобиос. Загрузиться в CloverGUI и нажать F6. Ваш Биос будет сохранен в файле /EFI/CLOVER/misc/c0000.bin, если, конечно, Кловер установлен в раздел с файловой системой FAT32.
2. Загрузите по одной из этих ссылок программу radeon_bios_decode. В ту же папку с этой утилитой положите файл биоса c0000.bin. Допустим, это папка `~/RadeonPatch`
   Выполняем в терминале следующие команды
   ```
   cd ~/RadeonPatch
   ./radeon_bios_decode < c0000.bin
   ```
3. На экране вы получите информацию по вашим коннекторам, которую стоит скопировать/сфотографировать для дальнейшего использования.
   Вот что у меня

```
iMac:test slice$ ./radeon_bios_decode <c0000.bin
ATOM BIOS Rom:
```

```
    SubsystemVendorID: 0x1458 SubsystemID: 0x2557
    IOBaseAddress: 0xe000
    Filename: R667D32I.F1
    BIOS Bootup Message:
GV-R667D3-2GI/F1

PCI ID: 1002:6758
Connector at index 0
    Type [@offset 44282]: HDMI-A (11)
    Encoder [@offset 44286]: INTERNAL_UNIPHY2 (0x21)
    i2cid [@offset 44356]: 0x92, OSX senseid: 0x3
Connector at index 1
    Type [@offset 44292]: DVI-D (3)
    Encoder [@offset 44296]: INTERNAL_UNIPHY (0x1e)
    i2cid [@offset 44383]: 0x95, OSX senseid: 0x6
Connector at index 2
    Type [@offset 44302]: VGA (1)
    Encoder [@offset 44306]: INTERNAL_KLDSCP_DAC1 (0x15)
    i2cid [@offset 44410]: 0x90, OSX senseid: 0x1
```

4. Загрузите по одной из ссылок скрипт **ati-personality.pl**
5. Положите в эту же папку, и выполните в терминале
   `perl ati-personality.pl -386 >frames.txt`
   если вы делает это для 32-битной системы, или

   `perl ati-personality.pl >frames.txt`
   для 64-битной.
   Внимание! В Сиерре тексты поменялись, так что патч получается системно-зависимый.

6. Теперь нужно определиться с выбором подходящего фреймбуфера. Эппл предлагает нам широкий выбор: и птички, и рыбки, и даже обезьяны. Но реальные отличия там в основном в коннекторах, которые мы и собираемся изменить. Если не слишком задумываться, то простой вариант подбора:

5000 серия: мобильный — Alouatta, десктоп — Baboon
6000 серия: мобильный — Cattail, десктоп — Ipomoea
7000 серия: мобильный — Pondweed, десктоп — Futomaki.

7. Для выбранного фреймбуфера берем распечатку коннекторов из нашего файла frames.txt, полученного на шаге 5.
   0000000 **00 04** 00 00 04 03 00 00 00 01 00 00 **12 04** 01 **05**
   0000010 00 08 00 00 04 02 00 00 00 01 00 00 **11 02** 04 03
   0000020 10 00 00 00 10 00 00 00 00 01 00 00 **00 00** 00 **02**
   Красным цветом выделены цифры, которые необходимо править. Синие цифры – просто адреса, нужно отбросить. Третья цифра с конца – encoderid, последняя цифра — senseid. Первые 4 цифры в каждой строке –
   тип монитора (точнее сказать тип коннектора).

```
ConnectorType
 02 00 00 00  LVDS
 04 00 00 00  DVI_DL(Dual Link)
 00 02 00 00  DVI_SL(Single Link)
 10 00 00 00  VGA
 80 00 00 00  S-Video
 00 04 00 00  DP
```

```
00 08 00 00  HDMI
```

8. senseid мы получили на шаге 3 для каждого из наших коннекторов. encoder можно просто всюду занулить. На остальные цифры не обращаем внимания.
   Получаем следующую таблицу:
```
0000000 04 00 00 00 04 03 00 00 00 01 00 00 10 00 01 06
0000020 10 00 00 00 10 00 00 00 00 01 00 00 00 00 00 01
0000010 00 08 00 00 04 02 00 00 00 01 00 00 12 00 04 03
```

T.e. Первая строка DVI-D, вторая – VGA, третья – HDMI, и все с моими значениями senseid.

9. И еще рецепт от Sergey_Galan. http://www.applelife.ru/threads/mobility-ati-radeon-hd5650m-hd5470m-hd4570m-hd4650m.29028/page-58#post-379044
   Вторая цифра с конца HotPlugID должна следовать по порядку 00, 01, 02. Это влияет на сон и пробуждение. (выделил красным)
```
0000000 04 00 00 00 04 03 00 00 00 01 00 00 10 00 00 06
0000020 10 00 00 00 10 00 00 00 00 01 00 00 00 00 01 01
0000010 00 08 00 00 04 02 00 00 00 01 00 00 12 00 02 03
```
10. Отбросив синие цифры остальные вписываем в config.plist без пробелов и переносов строк. Исходная таблица в ATIConnectorsData, после наших правок в ATIConnectorsPatch. Смотрите образец выше по тексту.
11. Еще я видел ситуацию, когда VGA разъем был представлен среди коннекторов как DVI-I (DVI-SL). И патч сработал с таким использованием.
12. Рецепт от eierfrucht https://applelife.ru/threads/sony-vaio-vpceb3m1r.522504/page-3#post-537081:
    Самый интересный момент -- это биты FEATURES у LVDS коннектора, советую перебором попробовать 08 01, 08 00, 09 01 и 09 00, на одном из них все должно начать нормально просыпаться. (Выделил оранжевым)
```
0000000 02 00 00 00 40 00 00 00 09 01 00 00 00 00 00 07
0000010 00 04 00 00 04 06 00 00 00 73 00 00 11 02 01 01
```
13. Там же, "***Senseid LVDS панели поставить 0x7***", потому что Сони VAIO.

По-новому нужно сделать так

```
<key>ForceKextsToLoad</key>
<array>
   <string>\System\Library\Extensions\AMD6000Controller.kext</string>
   <string>\System\Library\Extensions\AMDFramebuffer.kext</string>
</array>
<key>KextsToPatch</key>
<array>
   <dict>
      <key>Comment</key>
      <string>ATI Connector patch new way</string>
      <key>Disabled</key>
      <false/>
      <key>Find</key>
      <data>AAQAAAQDAAAAAQAAIQMCBAQAAAAUAgAAAAEAAAAABAMQAAAAEAAAAAABAAAAAAAB</data>
      <key>MatchOS</key>
      <string>10.9,10.10,10.11</string>
      <key>Name</key>
      <string>AMD6000Controller</string>
      <key>Replace</key>
      <data>BAAAABQCAAAAAQAAAAAEBAAEAAAEAwAAAAEAABECAQUAAAAAAAAAAAAAAAAAAAAA</data>
   </dict>
</array>
```

Для 10.12 продублировать весь <dict> с другими строками Find/Replace, они длиннее.

Мы также можем задать бинарные патчи для boot.efi или для ядра kernel. Способ одинаковый, поэтому покажу на одном примере

<key>KernelToPatch</key>

<key>BootPatches</key>

Подробнее про маски смотрите в параграфе **Patching with Mask**.
Хочу заметить, что эта группа патчей практически никем не применяется, кроме собственно разработчиков, для того, чтобы искать ошибки в тяжелых случаях, неочевидных при внешнем осмотре. Так мы искали проблемы гибернации, так мы искали момент, когда E2 лочится в кернеле.

Devices

Группа параметров для остальных PCI устройств и шины вообще.

<key>Inject</key>
`<false/>`

Поставить это значение в true – вся внутренняя инжекция заменяется на ввод единой строки Properties, которая соответствует Эпловскому инжекту по протоколу `APPLE_GETVAR_PROTOCOL c GUID={0x91BD12FE, 0xF6C3, 0x44FB, {0xA5, 0xB7, 0x51, 0x22, 0xAB, 0x30, 0x3A, 0xE0}};` и используется на настоящих Маках. По старому хакеры называли это EFIstrings.

<key>Properties</key>

`<string>0207364862FA54HG345</string>`

Устарело! Начиная с ревизии 4497 делаем не строку 16ричных символов, а словарь по правилам gfxutil.

```
    <dict>
      <key>PciRoot(0x0)/Pci(0x14,0x0)</key>
      <dict>
        <key>AAPL,clock-id</key>
        <data>AA==</data>
        <key>AAPL,current-available</key>
        <data>sAQ=</data>
        <key>AAPL,current-extra</key>
        <data>vAI=</data>
        <key>AAPL,current-in-sleep</key>
        <data>6AM=</data>
        <key>built-in</key>
        <data>AA==</data>
        <key>device_type</key>
        <string>XHCI</string>
      </dict>
      <key>PciRoot(0x0)/Pci(0x19,0x0)</key>
      <dict>
        <key>built-in</key>
        <data>AQ==</data>
      </dict>
```

Можно, к примеру, с помощью DarwinDumper посмотреть, какой плист сгенерировал Кловер по-умолчанию, то есть, сделать автоматические инжекты, или как у вас было сделано по-старому, загрузиться в систему, и в системе вызвать в терминале
`clover-genconfig >config-gen.plist`
Там ищете такой словарь, и копируете в свой конфиг, а старые методы инжекта все выключаете. Еще можно посмотреть такой словарь в отчете DarwinDumper с настоящего Мака похожей конфигурации. Синтаксис такой же. Только Кловер понимает немного больше, он понимает `<string>, <integer>, <true>, <false>, <real>.`
Сегодня функция clover-genconfig и редактор plist включены в приложение Clover.app.
Пользуйтесь!

В принципе, такой же результат достигается и вставкой методов _DSM в DSDT, если он уже есть, и если заниматься его совершенствованиями. Кому как. В принципе Properties лучше, чем _DSM, он нативный для настоящиков, и срабатывает раньше запуска ядра.

```
<key>PCIRootUID</key>
<integer>0</integer>
```

Оказывается, инжекция свойств видеокарты зависит от того, какое число стоит в DevicePath=PciRoot(0x0) или PciRoot(0x1). Ранее считалось, что это аппаратная характеристика. Однако, еще на заре хакинтошестроения выяснилось, что это число – просто идентификатор, прописанный в DSDT. Вот здесь:

```
        Device (PCI0)
        {
            Name (_HID, EisaId ("PNP0A08"))
            Name (_CID, EisaId ("PNP0A03"))
            Name (_ADR, Zero)

            Name (_UID, Zero)
```

_UID=Zero – значит 0, если же равно **One**, значит 1.

Причем, если это число поменять насильно, оно поменяется, и будет успешно работать. Так вот, настоящие Маки имеют всегда 0. И соответственно boot.efi всегда предполагает 0, поэтому лучше, если поправить свой ДСДТ, Clover делает это по-умолчанию, и такого ключа в конфиге больше нет.

```
<key>Audio</key>
<dict>
   <key>Inject</key>
   <string>887</string>
   <key>ResetHDA</key>
   <true/>
   <key>AFGLowPowerState</key>
   <true/>
</dict>
```

Inject

Инжекция свойств звуковой карточки. Правда, это работает, только если устройство в DSDT называется HDEF, если же заниматься его переименованием, то и остальное можно другим способом инжектировать. Также эти усилия не нужны при использовании драйвера VoodooHDA.

Варианты значений следующие:

NO – ничего не инжектируется, например, если будете сами инжектировать свойства через Properties

Detect – автоматическое определение установленной звуковой микросхемы, чтобы ее ID употребить в качестве лейаута. Вообще-то бред, но очень популярный. Во многих случаях не мешает, и оказывает влияние на отображание звуковой карточки в Систем-Профайлере.

883 – в десятичном виде номер лейаута. Имеется ввиду Realtek ALC883.

0x0373 – тоже самое в 16-чном виде становится неузнаваемым.

На самом деле эти числа неправильные, правильный лэйаут, например, 12 = 0x0C, но, как ни странно, являются допустимыми.

С появлением AppleALC этот параметр приобрел новую жизнь. Теперь это действительно номер лейаута, и вам нужно его подбирать из списка рекомендованных для вашего звукового кодека. Смотрите документацию к этому кексту.

ResetHDA — если звуковой чип почему-то не включается, то этот ключ может помочь с начальным запуском. Воздействует также и на Виндоус. Необходимость замечена после перезагрузки из Виндоуса в Мак.

AFGLowPowerState — воздействует на драйвер AppleHDA, и вроде должен решить проблему со щелчками в динамиках при засыпании/пробуждении звуковой карты. Подтверждений эффективности патча маловато.

```
    <key>USB</key>
<dict>
   <key>Inject</key>
   <true/>
   <key>AddClockID</key>
   <true/>
   <key>FixOwnership</key>
   <true/>
        <key>HighCurrent</key>
     <true/>
</dict>
```

Inject
Можно поставить **false**, если вы почему-то хотите отказаться от инжекции свойств USB, например, если будете сами инжектировать свойства через Properties.
FixOwnership
БИОС захватывает управление ЮСБ, и перед стартом ядра мы должны оторвать ЮСБ от БИОСа. ~~Для УЕФИ загрузки он вроде и не актуален, поэтому, по-умолчанию он включен для легаси-загрузки, и выключен для УЕФИ.~~ Актуален!
AddClockID
При наличии такого свойства ЮСБ контроллер засыпает намертво, и не будит компьютер. Если же вы хотите пробуждаться от ЮСБ мышки, и ставьте здесь false. Но будьте готовы, что вам компьютер будет пробуждаться самопроизвольно, например от встроенной камеры.
HighCurrent
Повышенный ток на этом ЮСБ контроллере, нужен для зарядки айПада, но я не стал делать такое значение по-умолчанию.

Группа параметров для маскировки своих устройств под нативные для OSX. (1971)

```
<key>FakeID</key>
<dict>
     <key>ATI</key>
     <string>0x67501002</string>
     <key>IntelGFX</key>
     <string>0x01268086</string>
     <key>NVidia</key>
     <string>0x0FE210DE</string>
     <key>LAN</key>
     <string>0x436311AB</string>
     <key>SATA</key>
     <string>0x25628086</string>
     <key>WIFI</key>
     <string>0x431214E4</string>
     <key>XHCI</key>
     <string>0x1E318086</string>
     <key>IMEI</key>
     <string>0x1E3A8086</string>
 </dict>
```

В этой группе параметров можно задать перемаркировку своего неподерживаемого устройства в поддерживаемое. Примеры:

- AMDRadeonHD7850 имеет DeviceID=0x6819, который неподдерживается кекстами ATI7000Controller и ATIRadeonX3000 в системе 10.8. Зато там есть поддержка для DeviceID=0x6818. Делаем подмену. Чтобы она вступила в силу требуется этот фейк

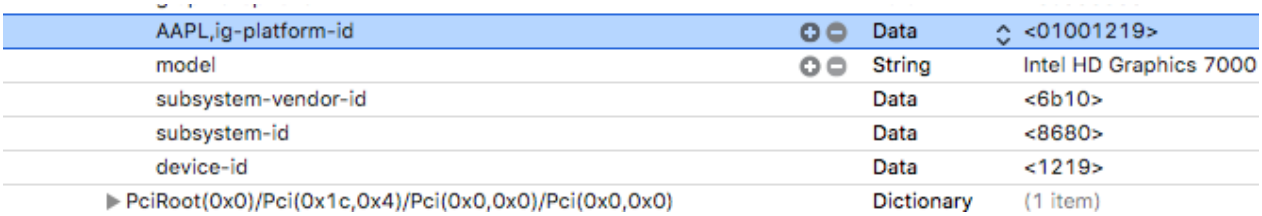
как-то проинжектировать. Для видеокарт два варианта: либо Inject->ATI=true, либо DsdtFixMask включает 0x0100 (FixDisplay).

- NVidia GTX660 имеет DeviceID=0x1183, карточка работает по-любому, но AGPM для нее не предусмотрен. Делаем подмену на 0x0fe0, и AGPM включается. Поскольку для такой карточки Inject->NVidia=false, то подстановку ID можно сделать только через патч DSDT с маской 0x0100 (FixDisplay).
- WiFi карточка в ноутбуке Dell называется Dell Wireless 1595, DeviceID=0x4315, реально это Broadcom, у которого поддерживаются чипы 4312, 4331, и ряд других. Делаем подстановку. Патч DSDT с маской 0x4000 (FixAirport).
- Распространенная сетевая карта Marvell 80E8056 DeviceID=0x4353 просто так не работает, однако работает с драйвером AppleYukon2, если сделать подмену ID на 0x4363. Патч DSDT с маской 0x2000 (FixLan).
- IMEI - это устройство работает вместе с Intel HD3000/4000, однако, не факт что ваш чипсет имеет правильный ID. Подстановки следующие:
  SandyBridge = 0x1C3A8086
  IvyBridge = 0x1E3A8086
  Haswell = 0x8C3A8086
  Работает с патчем DSDT фикс AddIMEI_80000 (AddIMEI)

Эта маскировка работает в двух случаях: при инжекте, либо при патче ДСДТ. Однако, если мы не хотим полный инжект в том варианте, как задумано Кловером, то мы можем задать следующее свойство:

```
<key>NoDefaultProperties</key>
<true/>
```

В этом случае строчка для инжекта создается, но пока не содержит ни одного нового свойства. Например таким свойством будет FakeID.
Опять-таки, это способ делать FakeID устарел, лучше делать через Properties следующим способом



Можно добавить и другие свои свойства, например model, в следующем массиве словарей
Устарело! Используйте Properties!

```
<key>AddProperties</key>
<array>
  <dict>
    <key>Device</key>
    <string>NVidia</string>
    <key>Key</key>
    <string>AAPL,HasPanel</string>
    <key>Value</key>
    <data>AQAAAA==</data>
  </dict>
...
</array>
```

Значение Value может быть <data> или шестнадцатиричная строка. Просто строку нельзя. То есть вместо <string> ABC.... надо писать <string>0x414243....
Конвертируйте через PlistEditor или через Xcode.

Первый ключ **Device** определяет, на какое именно устройство будет добавлено такое свойство. Список устройств:
ATI
Nvidia
IntelGFX
LAN
WIFI
Firewire
SATA
IDE
HDA
HDMI
LPC
SmBUS
USB
Названия должны быть именно такие, буква в букву. Я думаю, пояснения здесь не нужны. Таким образом можно инжектировать разные свойства для аналогового звука Device=HDA, и для цифрового Device=HDMI. Отличать Кловер будет, увы, не очень корректно, по вендору. Если Интел, значит HDA, если ATI или Nvidia, значит HDMI. Например на Хазвеле есть Intel HDMI sound. Этот вариант в Кловере пока не предусмотрен. Предусмотрено, что с Интел графикой на выход HDMI будет использоваться чипсетный HDA звук. Для этого служит параметр

```
<key>UseIntelHDMI</key>
<true/>
```

Влияет этот параметр на инжекцию свойств звука, передаваемому по HDMI, а также на DSDT патч. Однако, насколько мне известно, звуковые драйвера, что VoodooHDA, что AppleHDA, не совсем полноценно работают с HDMI выходом. По новым сведениям, VoodooHDA работает только с HDMI выходом NVIDIA, а что касается AMD, то Apple в системах 10.13+ создала новый драйвер AppleGFXHDA.kext. Изучайте его возможности.

```
<key>HDMIInjection</key>
<false/>
```

Совсем запретить инжекцию свойств HDMI устройства.

Начиная с ревизии 3262 вводится новый способ инжекции свойств устройств не по имени, как это было раньше, а по их расположению на PCI шине. Вот список, который выдает Кловер в бут-логе

```
4:432  0:000  PCI (00|00:00.00) : 8086 2E30 class=060000
4:432  0:000  PCI (00|00:02.00) : 8086 2E32 class=030000
4:432  0:000  Found GFX model=Unknown
4:432  0:000  PCI (00|00:02.01) : 8086 2E33 class=038000
4:432  0:000  PCI (00|00:1B.00) : 8086 27D8 class=040300
4:432  0:000  PCI (00|00:1C.00) : 8086 27D0 class=060400
4:432  0:000  PCI (00|01:00.00) : 10DE 01D1 class=030000
4:432  0:000  Found NVidia model=Gigabyte GeForce 7300 LE
4:432  0:000  PCI (00|00:1D.00) : 8086 27C8 class=0C0300
4:432  0:000  PCI (00|00:1D.01) : 8086 27C9 class=0C0300
4:432  0:000  PCI (00|00:1D.02) : 8086 27CA class=0C0300
4:432  0:000  PCI (00|00:1D.03) : 8086 27CB class=0C0300
4:432  0:000  PCI (00|02:05.00) : 10EC 8167 class=020000
```

Здесь видим две видеокарточки и четыре USB UHCI устройства. Тип устройства опознается по его классу class=040300 — это звуковое устройство стандарта HDA. В данном случае

расположено по адресу 00:1B.00, а сетевая карточка класса 020000 расположена по адресу 02:05.00
Bus= 02
Device = 05
Function = 00
Используем это значение для инжекции свойств
Устарело! Используйте Properties!

<key>Arbitrary</key>

```
<array>
   <dict>
      <key>PciAddr</key>
      <string>02:05.00</string>
      <key>Comment</key>
      <string>Realtek LAN 8167</string>
      <key>CustomProperties</key>
      <array>
         <dict>
            <key>Disabled</key>
            <true/>
            <key>Key</key>
            <string>model</string>
            <key>Value</key>
            <string>Realtek 8169 Gigabit Ethernet Controller</string>
         </dict>
         <dict>
            <key>Key</key>
            <string>built-in</string>
            <key>Value</key>
            <data>AQAAAA==</data>
         </dict>
      </array>
      </dict>
      <dict>
         <key>PciAddr</key>
         <string>01:00.00</string>
         <key>Comment</key>
         <string>Nvidia Geforce card in PCIe slot</string>
         <key>CustomProperties</key>
         <array>
            <dict>
               <key>Key</key>
               <string>model</string>
               <key>Value</key>
               <string>Gigabyte GeForce 7300 LE</string>
            </dict>
            <dict>
               <key>Disabled</key>
               <true/>
               <key>Key</key>
               <string>AAPL,boot-device</string>
               <key>Value</key>
               <data>AQAAAA==</data>
            </dict>
         </array>
      </dict>
   </array>
```

Таким образом секция Arbitrary представляет собой массив словарей, каждый из которых соответствует одному устройству с заданным адресом, и для описания каждого устройства используется массив CustomProperties состоящий из пар Key-Value. Также конкретное свойство может быть выключено ключом **Disabled.**
Включить или выключить свойство можно динамически в меню Кловера.
Key должен быть строкой <string>
Value может быть строкой, числом или данными <string>, <integer>, <data>

```
<key>ForceHPET</key>
<true/>
```
Оказывается, еще встречаются компьютеры, где HPET по умолчанию выключен, и в БИОСе нет галки для его включения. Поможет этот ключ (ревизия 2789+)

```
<key>SetIntelBacklight</key>
<false/>
```
Ключ введен в ревизии 3298. В прежних системах яркость экрана поднимали специальными кекстами IntelBacklight или ACPIBacklight, в Капитане они не работают, но оказалось очень просто это сделать в Кловере на этапе старта системы, и не нужен дополнительный кекст, просто поставьте <true/>. Трюк от Ramalama и Rehabman.
И дополнительные ключи

```
<key>SetIntelMaxBacklight</key>
<true/>
```
В регистр чипа будет прописано значение из следующего ключа:

```
<key>IntelMaxValue</key>
<integer>1808</integer>
```
либо дефолтное значение, наиболее подходящее для этого чипа. Шерлок внес в Кловер значения почти для всех интелловских чипов, на основе анализа дампов с настоящиков. Всегда ли это правильно, не знаю, но дефолтные значения Кловер предложит.

```
<key>DisableFunctions</key>
<string>0x18F6</string>
```
Это маска, которая накладывается в RCBA 0x3418 — выставляет дополнительные биты, запрет некоторых устройств в интелловском чипсете. Для очень серьезных хакеров.

```
<key>LANInjection</key>
<false/>
```
По-умолчанию для сетевой карты инжектируется свойство built-in. Эти параметром можно запретить такую инжекцию.

RtVariables

Следующие два параметра введены начиная с ревизии 980 и предназначены для разрешения регистрации в сервисе iMessage.
Начиная с ревизии 1129 параметры берутся из СМБИОС, и здесь не нужны.
MLB = BoardSerialNumber
ROM = последние цифры SmUUID либо Mac address.

```
<key>MLB</key>
<string>XXXXXXXXXX</string>
```
Цифры и буквы, длиной 17 знаков, означающие серийный номер материнской платы. Закономерности нету. Самое надежное, взять реальный номер, и поменять средние цифры, например, написать ...SLICE... У кого какая фантазия.

```
<key>ROM</key>
<data>AAAAAAAA</data>
```
Шесть пар 16-чных цифр, часто совпадающих с Мак-адресом сетевой карты. Есть, однако, сообщения, что сервис работает с произвольными цифрами. Начиная с ревизии 3051 можно

написать **<string>UseMacAddr0</string>**, и Кловер сам определит МакАдрес вашей сетевой карты. Процедура работает не у всех, поэтому проверяйте.
Ну и самое главное, регистрация в iMessage подразумевает платный сервис, вы должны указать реальную банковскую карту, с которой у вас будет списано 1$. Кто пытается войти нахаляву, получают сообщения типа «Позвоните в Эппл».

В 2015 году появилась система 10.11 ElCapitan с новыми требованиями по безопасности. SIP = System Integrity Protection. По умолчанию защита включена и не позволяет грузить свои кексты и устанавливать свои системные утилиты. Чтобы ее отключить, Кловер дает возможность выставить новые параметры в NVRAM

```
    <key>CsrActiveConfig</key>
  <string>0x3E7</string>

    <key>BooterConfig</key>
  <string>0x28</string>
```

Это битовые маски с возможными значениями битов

```
/* Rootless configuration flags */
#define CSR_ALLOW_UNTRUSTED_KEXTS       (1 << 0)
#define CSR_ALLOW_UNRESTRICTED_FS       (1 << 1)
#define CSR_ALLOW_TASK_FOR_PID          (1 << 2)
#define CSR_ALLOW_KERNEL_DEBUGGER       (1 << 3)
#define CSR_ALLOW_APPLE_INTERNAL        (1 << 4)
#define CSR_ALLOW_DESTRUCTIVE_DTRACE    (1 << 5) /* name deprecated */
#define CSR_ALLOW_UNRESTRICTED_DTRACE   (1 << 5)
#define CSR_ALLOW_UNRESTRICTED_NVRAM    (1 << 6)
#define CSR_ALLOW_DEVICE_CONFIGURATION  (1 << 7)
#define CSR_ALLOW_ANY_RECOVERY_OS       (1 << 8)
#define CSR_ALLOW_UNAPPROVED_KEXTS      (1 << 9)
#define CSR_ALLOW_EXECUTABLE_POLICY_OVERRIDE    (1 << 10)


/* Bitfields for boot_args->flags */
#define kBootArgsFlagRebootOnPanic    (1 << 0)
#define kBootArgsFlagHiDPI            (1 << 1)
#define kBootArgsFlagBlack            (1 << 2)
#define kBootArgsFlagCSRActiveConfig  (1 << 3)
#define kBootArgsFlagCSRPendingConfig (1 << 4)
#define kBootArgsFlagCSRBoot          (1 << 5)
#define kBootArgsFlagBlackBg          (1 << 6)
#define kBootArgsFlagLoginUI          (1 << 7)
```

Значения по-умолчанию соответствуют выключению защиты.
Если, в силу какой-то паранойи, вам нужно включить защиту, ставьте значение 0 (ноль).

Следующие три параметра исключены, ибо должны выставляться системой из Контрольной Панели Кловера, чтобы не было конфликта.

```
    <key>MountEFI</key>
    <string>Yes</string>
```

Этот параметр сообщает стартовому скрипту, что при входе в систему следует смонтировать раздел ESP (EFI System Partition). Этот параметр для большинства людей является ненужным или временным, стоит в конфиге прописать No, а в меню при необходимости ставить Yes. Еще возможное значение disk1, если у вас несколько дисков, и на каждом есть свой раздел EFI.

```
    <key>LogEveryBoot</key>
    <string>Yes</string>
```

Лог загрузки нужен разработчикам, а простым пользователям можно и No поставить. Здесь вместо Yes может быть число, сколько логов хранить в системе.

```
<key>LogLineCount</key>
<string>3000</string>
```

Количество строк в этом логе, дальше идет вытеснение старых строк новыми, чтобы не было неограниченного роста этого файла.

Реально копить эти логи неинтересно. Всегда можно получить последний лог командой

```
$ bdmesg > ~/Desktop/boot-log.txt
```

DisableDrivers

```
<key>DisableDrivers</key>
<array>
   <string>CsmVideoDxe</string>
   <string>VBoxExt4</string>
</array>
```

Суть этой секции в том, чтобы иметь разные config.plist в разных OEM папках, но, поскольку папка drivers общая, надо как-то различать, какой набор драйверов используется на той или иной конфигурации. На одной, к примеру, нужен OsxAptioFixDxe, на другой нужен EmuVariableDxe. Устарело! Сейчас на всех платах используется общий набор.

Quirks

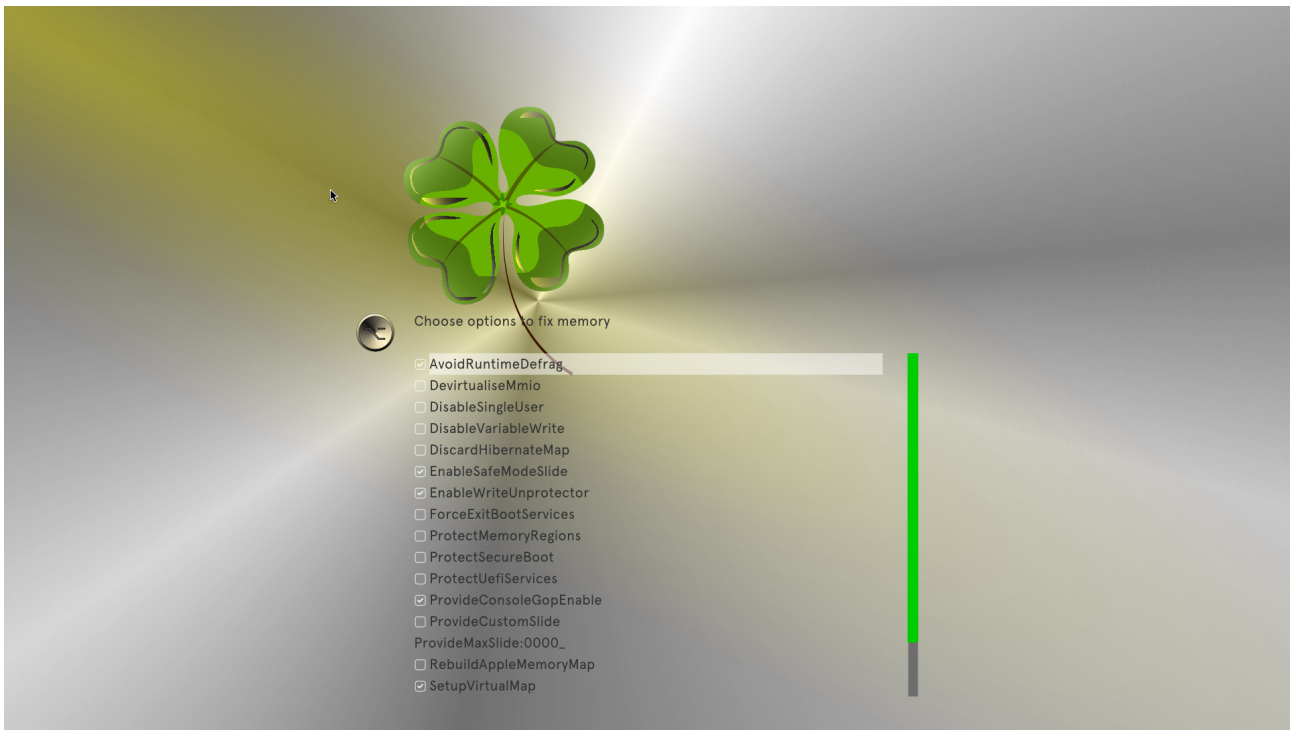
История этой секции такова. Начало разработки Кловера для УЕФИ загрузки Дмазаром это было разработка драйвера для корректировки памяти, которую оставляет за собой UEFI BIOS Aptio (American Megatrend). Дело в том, что функция Allocate в этом БИОСе выделяет память в нижней части, а для загрузки macOS нужно иметь нижнюю память свободной. Конфликт задевает не только память, там еще boot.efi делает виртуализацию адресов, и это влияет на указатели, на функции и так далее. Подробнее не буду, это не моя работа, это Дмазар шаг за шагом нашел все конфликты, и придумал, как их разрешить. Это стал драйвер OsxAptioFixDrv.efi. Варианты 32 и 64 бита со всеми отличиями адресации.

Замечу, что с легаси Кловером этой проблемы не было, ибо в этом случае используется Allocate из EDK2, а он выделяет память сверху вниз. Легаси Кловер работает без этого драйвера.

Долгое время после ухода Дмазара никто этого драйвера не касался, кроме, может быть, отдельных однострочных добавок типа Free2000. И вот vit9696 взялся за переделку драйвера капитально. В первую очередь он сделал изменение, позволяющее использовать нативный NVRAM на многих чипсетах (БИОСах), с которыми раньше это не работало. А дальше он разбил драйвер на смысловые части (quirks), которые теперь можно было включать и выключать по желанию пользователя, если используется загрузчик OpenCore.

Но нашелся и программист ReddestDream, который решил выделить все эти наработки из OpenCore в отдельный драйвер OcQuirks.efi, который работает совместно с драйвером OpenRuntime.efi, а все настройки записаны в файле OcQuirks.plist, чтобы использовать все это с загрузчиком Кловер.

А дальше мой ход. Мне нужно иметь все исходники в одном репо, чтобы можно было делать бисекцию. А поскольку лицензия на все это дело позволяет использовать исходники по своему разумению, да и исторически все вернулось на историческую родину, я их скопировал, и модернизировал таким образом, чтобы вместо отдельного файла .plist использовался тот же самый кловеровский config.plist, а эти настройки еще и можно менять просто из ГУИ Кловера.



То есть, если вы не можете сразу загрузиться, вы можете попытаться зайти в это меню и поменять какую-то установку да-нет.
Теперь подробности про каждый пункт. Указаны значения по-умолчанию.

<key>AvoidRuntimeDefrag</key>

```
<true/>
```

Предотвращает дефрагментацию памяти для runtime-services, например поддержка NVRAM. Рекомендуется для всех, кроме Apple и VMware.

<key>DevirtualiseMmio</key>

```
<false/>
```

Снимает атрибут runtime с некоторых известных областей MMIO. Не рекомендуется для систем старше Sandy Bridge. Список известных регионов еще можно дополнить в секции кирков MmioWhitelist. Однако, по моим наблюдениям всегда отключено. Утверждается необходимость для Z390 чипсета.

<key>MmioWhitelist</key>

```
<array/>
```

Список регионов задается как массив словарей

```
<dict>
  <key>Comment</key>
  <string>Это такой-то регион</string>
  <key>Address</key>
  <string>0xffe00000</string>
  <key>Enabled</key>
  <true/>
</dict>
```

<key>DisableSingleUser</key>

```
<false/>
```

Запрещает использование режима командной строки, потому что это небезопасно. ;)

```
<key>DisableVariableWrite</key>
<false/>
```
Запрещает доступ из МакОС к HBPAM по записи, для безопасности.

```
<key>DiscardHibernateMap</key>
<false/>
```
При пробуждении из гибернации использовать старую карту памяти. Использовать, только если вы точно уверены, что это ваш случай. Практически всегда нет.

```
<key>EnableSafeModeSlide</key>
<true/>
```
По умолчанию загрузка в безопасном режиме дает slide=0. Этот патч позволяет использовать другое значение, заданное в кирке ProvideCustomSlide.

```
<key>ProvideCustomSlide</key>
<false/>
```
Использовать или нет автоматические вычисление значения slide=***. Необходимость видна из лога, если есть сообщение `OCABC: Only N/256 slide values are usable!`

```
<key>ProvideMaxSlide</key>
<integer>0</integer>
```
Значение от 1 до 254, для случая указанного выше. Значение 255 выставляется по-умолчанию, и это может приводить к ошибкам выделения памяти.

```
<key>EnableWriteUnprotector</key>
<true/>
```
Снимает бит защиты записи в странице Runtime сервисов. Ввиду небезопасности этого кирка есть другой RebuildAppleMemoryMap, но если ваш набор ACPI содержит MATS.

```
<key>ForceExitBootServices</key>
<false/>
```
Повторить попытку ExitBootServices с новой картой памяти. Это тот момент, когда происходят патчи ядра и кекстов. Используйте только если вы уверены, что делаете.

```
<key>ProtectMemoryRegions</key>
<false/>
```
Меняет атрибуты некоторых регионов, когда имеется конфликт между понятиями макОС и БИОС. Этот кирк включает в себя несколько фиксов, в том числе разработанный лично мной, тот, который защищает CSM регион. В частности, если ваша видеокарта не имеет UEFI VBIOS, этот кирк следует включить. С другой стороны, в драйвере OsxAptioFix3Drv он безусловно включен, и никогда не вызывал нареканий. Тем не менее пока выключен по дефолту.

```
<key>ProtectSecureBoot</key>
<false/>
```
Защищает ключи шифрования от перезаписи операционной системой. Технология Secure Boot, которая нам как-то не особенно нужна. Необходимость этого кирка наблюдается на некоторых БИОСах Insyde. Остальным не нужен.

```
<key>ProtectUefiServices</key>
<false/>
```

Некоторые БИОСы, такие как VMware пытаются переписать указатели на Runtime сервисы. Нам это не нужно, поэтому защищаем их. Редискин утверждает, что это нужно на Z390 чипсете.

<key>ProvideConsoleGopEnable</key>

```
<true/>
```

Создает протокол GOP для режима консоли, то есть, чтобы вывод текста был не в текстовом режиме, как привыкли делать в PC BIOS, а в графическом, как это делает Apple. Кирк безусловно необходим.

<key>RebuildAppleMemoryMap</key>

```
<false/>
```

Генерировать карту памяти, совместимую с макОС. Дело в том, что у Apple несколько другие представления, как и что делать, чем в нашем UEFI BIOS. Проблема, однако, в том, что наша карта памяти соответствует нашему железу. Поэтому кирк отключаем. Другие кирки выполняют необходимую часть этой работы. Этот кирк вроде заменяет EnableWriteUnprotector для систем, поддерживающих MAT.
Также этот кирк требует включения SyncRuntimePermissions. Впрочем, у нас он включен по-умолчанию.

<key>SetupVirtualMap</key>

```
<true/>
```

Кирк имеет дело с порядком присвоения виртуальных адресов и их использования. Некоторые БИОСы, такие как OVMF, не поддерживают этот кирк. Хм-м-м, а зачем этот драйвер OcQuirks.efi в системе с OVMF?! Она без него работает, как и легаси Кловер.
Так что включаем его.

<key>SignalAppleOS</key>

```
<false/>
```

Сообщает БИОСу, что мы грузим систему макОС, хотя мы грузим Виндоус. Нужен на некоторых макбуках, но не нам.

<key>SyncRuntimePermissions</key>

```
<true/>
```

Обновляет флаги разрешений в области Runtime. Разумеется нужно.

Общая ситуация такова, что для моих не очень новых компьютеров именно эти установки по-умолчанию оказались достаточными. У Редискина список другой, и он утверждает, что его список соответствует поведению AptioMemoryFix. Однако, это неправда. С его набором мой компьютер не грузится, тогда как со старым AptioMemoryFix все в порядке. Так что в Кловере стандартный набор кирков отличается от оригинала.

ACPI

Группа параметров, регулирующих коррекцию различных ACPI таблиц.
И дело не только в том, что у Мака свои требования, но и просто разные версии АЦПИ спецификации, и элементарная лень производителей, и просто в БИОСе системной платы нет сведений об установленных картах и о ЦПУ (а динамически определить слабо? Кловер же это делает!).

Параметры для таблицы FADT

```
<key>ResetAddress</key>
<string>0x64</string>
<key>ResetValue</key>
<string>0xFE</string>
```

Эти два параметра служат для одного очень ценного фикса – исправление рестарта. Эти значения должны быть в таблице FADT, но почему-то они там не всегда есть, более того, бывает и сама таблица короче необходимого, короче настолько, что эти значения оказались отброшены. По умолчанию идет значение, уже присутствующее в FACP, однако, если там ничего нет, то используется пара **0x64/0xFE,** что означает рестарт через PS2 контроллер. Практика показала, что это не у всех работает, другая возможная пара значений **0x0CF9/0x06,** что означает рестарт через PCI шину. Эта пара используется и на нативнике, но не всегда работает на хакинтошах. Разница понятна, на хакинтошах есть еще и PS2 контроллер, который может помешать рестарту, если его не сбросить. Еще вариант **0x92/0x01**, не знаю, может кому-то поможет.

```
<key>HaltEnabler</key>
<true/>
```

А это исправление проблемы с выключением/уходом в сон при UEFI-загрузке. Фикс производится однократно, перед вызовом boot.efi, поэтому 100% эффективности не гарантируется. Тем не менее он достаточно безопасный, во всяком случае на Интел-чипсетах. Лично для меня это спасение!

```
<key>smartUPS</key>
<false/>
```

Вообще-то, этот параметр предназначен для того, чтобы прописать в таблице FADT профиль питания=3. Логика следующая:
PM=1 – desktop , питание от сети
PM=2 – notebook, питание от сети или от батарейки
PM=3 – server, питание от SmartUPS, про который MacOSX тоже что-то знает.
Выбор между 1 и 2 Кловер сделает на основе анализа бита мобильности, но также есть и параметр **Mobile** в секции SMBIOS. Можно, к примеру, сказать, что у нас МакМини, и что он мобильный. Значение же 3 будет подставлено, если **smartUPS**=Yes.

Корректировка MADT (APIC)

```
<key>PatchAPIC</key>
<false/>
```

На некоторых компьютерах можно загрузить систему только с cpus=1, либо со специальным патченным ядром (Lapic NMI patch). Простейший анализ показал, что у них неправильная таблица MADT, а именно, в ней отсутствуют разделы NMI. Этот параметр служит для корректировки таких таблиц на лету. Для здорового компьютера ничего плохого не произойдет. ~~Впрочем, я не видел и отчетов, что кому-то с чем-то помогло~~. Да, есть разработчик, которому это надо, и который поправил этот патч в новых версиях Кловера. Есть соответствующий патч в секции KernelAndKextPatches, тоже для решения этой проблемы, но другими средствами.

Другие АЦПИ таблицы:

```
<key>DropTables</key>
<array>
   <dict>
      <key>Signature</key>
      <string>DMAR</string>
   </dict>
   <dict>
      <key>Signature</key>
      <string>MCFG</string>
   </dict>
   <dict>
      <key>Signature</key>
      <string>SSDT</string>
      <key>TableId</key>
      <string>CpuPm</string>
      <key>Length</key>
      <string>0x0fe1</string>
   </dict>
</array>
```

В этом массиве мы перечисляем таблицы, которые хотим отбросить.
DMAR — потому что Мак не дружит с технологией VT-d. Вернее, там другая таблица.
MCFG — потому что задав модель MacBookPro или MacMini мы получаем жестокие тормоза. Уже придуман более правильный метод.

```
<key>FixMCFG</key>
<true/>
```

При этом таблица не отбрасывается, а корректируется. Автор патча опять vit9696. Тем не менее метод отбрасывания таблицы пока остается в запасе.
Возвращаемся к рассказу о DropTables
SSDT бывают разные, и мы указываем дополнительно TableId, какие будем отбрасывать, потому что собираемся генерировать свои таблицы SSDT, построенные по правилам Apple, а не Gigabyte, или, прости Господи, ASUS. Посмотреть можно в заголовке таблицы, или в бут-логе Кловера. Вот, к примеру, таблица, которую не стоит отбрасывать.

```
DefinitionBlock ("SSDT-0.aml", "SSDT", 1, "SataRe", "SataTabl", 0x00001000)
```

При этом на сохраненные таблицы будет распространяться правило для бинарных патчей DSDT, то есть эти таблицы так же будут модифицированы, что логично.
Если все SSDT таблицы почему-то имеют один и тот же TableID, то можно указать длину таблицы, которую хотим дроппануть. Длину можно задать в хексе, как выше, можно в <integer> как десятичное число.

```
<key>DisableASPM</key>
<false/>
```

Это влияет на настройки самой ACPI системы, типа того, что эпповское управление ASPM работает не так, как предусмотрено у нас. Например ненативный чипсет. В каких случаях нужно применять, и на что это влияет я не помню.

```
<key>SSDT</key>
```

```
<key>DropOem</key>
<true/>
```

Поскольку мы собираемся создавать или подгружать динамически свои таблицы SSDT, то нужно избежать ненужного пересечения интересов. Этот параметр позволяет отбросить **все**

родные таблицы, в пользу новых. Или же вы категорически хотите избежать патча таблиц SSDT. У вас есть такой вариант: подложить родные таблицы с небольшими правками в папку EFI/OEM/xxx/ACPI/patched/, а ~~непатченные дропнануть~~. (фу!) неисправленные таблицы отбросить. Лучше, однако, воспользоваться указанным выше способом выборочного Дропа.

<key>Generate</key>

```
<dict>
   <key>CStates</key>
   <true/>
   <key>PStates</key>
   <true/>
</dict>
```

Здесь мы определяем, что будут сгенерены две дополнительные таблицы для C-states и для P-states, по-правилам, разработанным хак-сообществом.
Для C-states, таблица с учетом параметров C2, C4, C6, Latency, упомянутых в секции CPU. Также можно указать параметры и в секции SSDT, что логично

```
<key>EnableC7</key>
<true/>
<key>EnableC6</key>
<true/>
<key>EnableC4</key>
<false/>
<key>EnableC2</key>
<false/>
<key>C3Latency</key>
<integer>67</integer>
```

То, что эта генерация вступила в силу, контроллируется по кернел-логу. Без этого метода присутствует ошибка *ACPI_SMC_PlatformPlugin::pushCPU_CSTData - _CST evaluation failed*.
Отдельное слово про `C3Latency`. Эта величина фигурирует в настоящих Маках, для айМака порядка 200, для МакПро порядка 10. По-моему, айМаки регулируются П-стейтами, МакПро — Ц-стейтами. И еще это зависит от чипсета, будет ли ваш чипсет адекватно реагировать на Ц-стейт команды от МакОС. Самый простой вариант — не пишите этого параметра, все будет работать и так.
Для P-states, таблица дополняющая процессорную секцию методами _PPC, _PCT и _PSS.
_PCT – Performance Control – контроль за управлением спидстепом.
_PPC – Performance Present Capabilities – возможности спидстепа, Эта функция возвращает одно число, которое означает ограничение частоты. Подробности ниже, в параметре **PLimitDict**.
_PSS – Performance Supported States – набор возможных состояний процессора – P-states. Этот массив формируется на основе данных о процессоре, которые Кловер уже вычислил, а также с учетом параметров пользователя:

<key>PLimitDict</key>

```
<string>1</string>
```

Суть параметра очень проста – ограничить максимальную частоту процессора. Значение 0 – работа до максимума, 1 – на одну ступень меньше максимума, 2 – на две ступени. Пример: Core2Duo T8300 2400MHz работает на максимальной частоте 2000, если ограничить на две ступени. Зачем? Да чтобы ноутбук не перегревался, там возможности ЦПУ намного превышают возможности по охлаждению. Точно такой же параметр присутствует в платформ-плистах, например:
System/Library/Extensions/IOPlatformPluginFamily.kext/Contents/PlugIns/ACPI_SMC_PlatformPlugin.kext/Contents/Resources/MacBook5_1.plist
Далее мы еще обсудим эти плисты.

Для некоторых процессоров, например Core2Quad, замечено, что PlimitDict работает наоборот, и лучший вариант =1. Вполне возможно, что это просто ошибка в ДСДТ. Например, потому что не захотели делать патч Дарвина

<key>UnderVoltStep</key>

```
<string>1</string>
```

Дополнительный параметр для снижения температуры процессора путем снижения его рабочего напряжения. Возможные значения 0, 1, 2, 3 … чем больше, тем сильнее охлаждаем, пока компьютер не повиснет. В этом месте работает защита против дурака, Кловер не позволит поставить значение вне допустимого диапазона, вернее пишите, что хотите, а работать будет только то, что дозволено. Впрочем и дозволенные значения могут давать неустойчивую работу. Эффект от этого параметра реально наблюдается. Однако, только для Пенрина.

<key>DoubleFirstState</key>

```
<true/>
```

Найдено, что для успешного спидстепа нужно в таблице P-states продублировать первый стейт. После введения других параметров необходимость этого стала сомнительной. Еще это верно только для ИвиБриджа, остальным отменить безусловно.

<key>MinMultiplier</key>

```
<integer>7</integer>
```

Минимальный множитель процессора. Сам он рапортует, что 16, и предпочитает работать на частоте 1600, однако, для спидстепа следует задать в таблице стейты вниз до 800 или даже 700. Эмпирика.

<key>MaxMultiplier</key>

```
<integer>30</integer>
```

Введено по аналогии с минимальным, но, похоже, зря. Его не стоит вписывать. Впрочем, он как-то влияет на количество П-стейтов, так что можете поэкспериментировать, однако, без особой нужды этого делать не стоит.

<key>Generate</key>

```
<dict>
   <key>CStates</key>
   <true/>
   <key>PluginType</key>
   <false/>
   <key>APLF</key>
   <false/>
   <key>APSN</key>
   <false/>
   <key>PStates</key>
   <true/>
</dict>
```

В новом Кловере эта группа параметров объединена в одну секцию, и PluginType теперь просто true или false. Потому что других вариантов нет. Параметры APLF и APSN вроде влияют на спидстеп, но для тех, кто знает, для чего они нужны.

<key>PluginType</key>

```
<integer>0</integer>
```

Для процессоров IvyBridge, Haswell (и выше?) нужно ставить 1, для остальных 0.

Большая секция по настройке и патчам DSDT.

```
<key>DSDT</key>
<dict>
```

```
<key>Debug</key>
<false/>
```

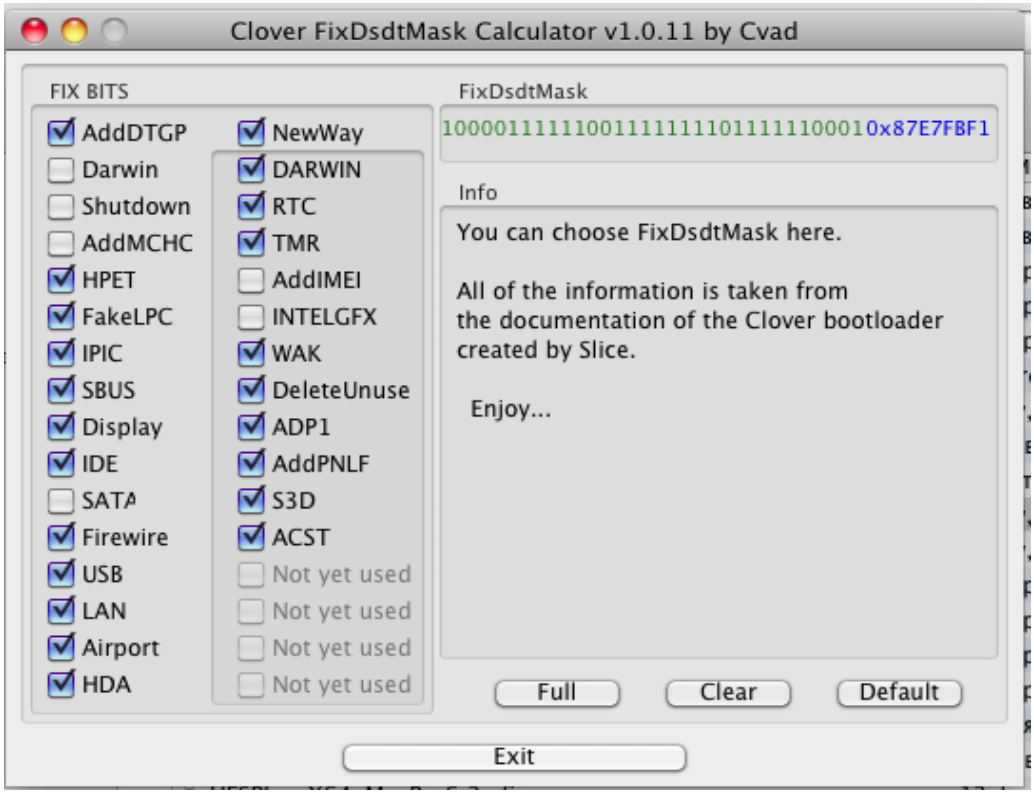
этот параметр позволяет видеть, что происходит с DSDT в процессе его патча, если мы после этого не можем загрузить систему. Сначала сохраняется исходный вариант /EFI/CLOVER/ACPI/origin/DSDT-or.aml, затем проделывается процедура всех патчей (кстати она оставляет немало сообщений в debug.log, если он тоже подключен), и затем сохраняется файл /EFI/CLOVER/ACPI/origin/DSDT-pa0.aml, если такой файл уже существует с предыдущей попытки, то будет создан следующий по номеру DSDT-pa1.aml, DSDT-pa2.aml... , они не будут перезаписывать друг друга. Не забудьте в конце всех упражнений почистить папку.

```
<key>Name</key>
<string>DSDT.aml</string>
```

У вас может быть несколько версий файла DSDT с произвольными именами, а можете написать несуществующее имя, например стандартно BIOS.aml, и тогда будет взят за основу тот DSDT, который имеется в БИОСе. Приоритеты файлов следующие:
1. Высший приоритет - файл DSDT.aml, лежащий в корне загружаемой системы. Логика в том, чтобы одной флешкой загружать разные компьютеры, у каждого из которых есть свой ДСДТ.
2. Если такого файла нет, ищем на флешке в секции OEM: /EFI/CLOVER/OEM/p8b/ACPI/patched/DSDT.aml
3. Если и там нет, то смотрим в общей папке /EFI/CLOVER/ACPI/patched/DSDT.aml

```
<key>FixMask</key>
<string>0xFFFFFFFF</string>
```

Под этим параметром скрывается сразу 32 патчей для таблицы DSDT, по числу битов в маске.
Чтобы подсчитать, как сумма битов складывается в ту или иную маску, можно вызвать системный калькулятор, перевести его в вид для программиста, и переключить на 16-чные числа. И теперь щелкая мышью по битам с 0 по 31 мы наберем нужную маску.
Есть более наглядный вариант: CloverFixDsdtMaskCalculator by Cvad
http://www.applelife.ru/attachments/cloverfixdsdtmaskcalculator-app-zip.43973/



Начиная с ревизии 2184 патчи можно (да и нужно) вносить побитно в следующей секции

<key>Fixes</key>

```
<dict>
 <key>AddDTGP</key>
 <true/>
 <key>FixDarwin</key>
 <true/>
 <key>FixShutdown</key>
 <true/>
 <key>AddMCHC</key>
 <false/>
 <key>FixHPET</key>
 <true/>
 <key>FakeLPC</key>
 <false/>
 <key>FixIPIC</key>
 <true/>
 <key>FixSBUS</key>
 <true/>
 <key>FixDisplay</key>
 <true/>
 <key>FixIDE</key>
 <false/>
 <key>FixSATA</key>
 <false/>
 <key>FixFirewire</key>
 <true/>
 <key>FixUSB</key>
 <true/>
 <key>FixLAN</key>
 <true/>
 <key>FixAirport</key>
 <true/>
 <key>FixHDA</key>
 <true/>
```

```
    <key>FixDarwin7</key>
    <true/>
    <key>FixRTC</key>
    <true/>
    <key>FixTMR</key>
    <true/>
    <key>AddIMEI</key>
    <true/>
    <key>FixIntelGFX</key>
    <false/>
    <key>FixWAK</key>
    <true/>
    <key>DeleteUnused</key>
    <true/>
    <key>FixADP1</key>
    <true/>
    <key>AddPNLF</key>
    <true/>
    <key>FixS3D</key>
    <true/>
    <key>FixACST</key>
    <true/>
    <key>AddHDMI</key>
    <true/>
    <key>FixRegions</key>
    <true/>
  </dict>
```

При наличии этой секции фиксов ключ фикса по маске будет проигнорирован.
А вот для того, чтобы рассказать, что означают эти фиксы, придется открыть новую главу.

Еще ключи, помогающие решить некоторые проблемы с автоматическим патчингом.

```
<key>ReuseFFFF</key>
<false/>
```

В некоторых случаях попытка сделать патч дисплея упирается в наличие в оригинальном ДСДТ устройства типа

```
Device (PEGP)
{
   Name (_ADR, 0xFFFF)
   Name (_SUN, One)
}
```

Ему можно поменять адрес на 0, но это не всегда работает. <true/> - пытаемся поменять адрес, <false/> - уходим и не пытаемся его патчить.

Это фикс исключен из нового Кловера начиная с 5116.

```
    <key>DropOEM_DSM</key>
  <dict>
    <key>ATI</key>
    <true/>
    <key>NVidia</key>
    <true/>
    <key>IntelGFX</key>
    <true/>
    <key>HDA</key>
    <true/>
    <key>HDMI</key>
    <true/>
    <key>LAN</key>
    <true/>
    <key>WIFI</key>
    <true/>
    <key>SATA</key>
```

```
      <true/>
      <key>IDE</key>
      <true/>
      <key>USB</key>
      <true/>
      <key>LPC</key>
      <false/>
      <key>SmBUS</key>
      <false/>
      <key>Firewire</key>
      <true/>
   </dict>
```

В некоторых случаях устройство, которые мы хотим автоматически пропатчить, уже имеет OEMный метод _DSM. Дублировать нельзя, поэтому два варианта:
<true/> - оригинальный метод будет откинут, и вместо него сгенерен наш,
<false/> - встретив оригинальный метод мы ретируемся, ничего не сделав.
Что там, в оригинальном методе? Да вряд ли то, что мы хотели бы увидеть, и вряд ли то, что нужно для OSX. Обычно производители БИОСов думают только о Windows.
Если же вам кажется, что в том методе что-то важное, то инжектируйте свои свойства в данное устройство с помощью стрингов (см. главу Devices->Inject). Так что я бы рекомендовал дроппать все эти OEM DSM, за исключением того варианта, когда вы подкладываете свой кастомный ДСДТ, и применяете к нему еще фиксы из числа автоматических, но вы не желаете подменять свои _DSM методы на автоматически сгенеренные.

В моем новом компьютере ДСДТ оказался безумно плохой с точки зрения macOS. В нем много методов DSM, причем как в родительском устройстве, так и в дочках. Результат — паника. DropOEM_DSM здесь уже бессильно. Помог патч следующего вида
Patches → find :_DSM, replace: ZDSM. См. ниже.

```
<key>SuspendOverride</key>
<false/>
```

Патч шатдауна работает только на состояние типа 5 - shutdown, однако, мы можем захотеть распространить этот патч на состояния 3 и 4, ставим SuspendOverride = true.
Мне это помогло с заходом в сон при UEFI-загрузке. Иначе экран гас, а лампочки и вентиляторы продолжали работать.

Более продвинутые хакеры могут делать собственные патчи ДСДТ, используя замену на бинарном уровне:

```
<key>Patches</key>
   <array>
   <dict>
         <key>Find</key>
         <data>UFhTWAhfQURSAAhfUFJXEgYC</data>
         <key>Replace</key>
         <data>UFhTWAhfQURSAAhfU1VOCgQIX1BSVxIGAg==</data>
   <key>TgtBridge</key>
   <data>UFhTW</data>
   </dict>
<dict>
   <key>Comment</key>
   <string>Rename oem _DSM to ZDSM</string>
   <key>Disabled</key>
   <false/>
   <key>Find</key>
   <data>
      X0RTTQ==
   </data>
```

```
        <key>Replace</key>
        <data>
           WkRTTQ==
        </data>
      </dict>
       </array>
```

TgtBridge определяет, что патч будет работать только внутри одного блока, определяемого этим именем
Device (RP02) { .. }
однако, этот метод неточен, потому что блоки бывают фрагментированные, лучше пользуйтесь методом RenameDevices (ниже)
Конкретные цифры — ваши, из ваших разработок, если вы знаете, что делать. Длины строк могут не совпадать, Кловер корректно учтет изменение длины, за одним исключением: чтобы это не произошло внутри оператора If или Else. Если вам требуется такое изменение, заменяйте оператор целиком.

Некоторые пояснения. Ключ `Comment` служит не только для напоминания Вам, что здесь написано, он также используется в меню Кловера для составления меню по подключению/отключению этих фиксов. Первоначальное значение включено или нет определяется строчками `Disabled=false`. Значение по-умолчанию — разрешено. Если у вас чужой набор патчей, то лучше их сначала прописать как `Disabled=true,` а потом в меню их разрешать по одному.

Другие ACPI таблицы

Кловер имеет сервис по подгрузке других таблиц. В частности, очень распространено изготовление целой библиотеки разных SSDT-xxx.aml для разных устройств и для спидстепа. Начиная с ревизии 3088 правило подгрузки таких таблиц изменилось.
Таблицы помещаем в папку
`/EFI/CLOVER/OEM/xxx/ACPI/patched/`
если такой папки нет, то рассматривается общая папка
`/EFI/CLOVER/ACPI/patched/`
Из этой папки будут грузиться все файлы с расширением ".aml", не начинающиеся с точки "." и не содержащие в своем имени строки "DSDT", потому что DSDT одного из разных вариантов грузится по другому алгоритму.
Порядок загрузки не гарантирован. Если же мы хотим строго определенного порядка загрузки, мы должны написать это явно в конфиге

```
  <key>ACPI</key>
  <dict>
    <key>SortedOrder</key>
    <array>
      <string>SSDT-3.aml</string>
      <string>SSDT-1.aml</string>
      <string>SSDT-2.aml</string>
    </array>
```

И если такой массив присутствует, будут загружены только эти таблицы, и строго в этом порядке.
Еще одна проблема с таблицами, это имя. OEM не чурается ни использование национального алфавита, ни просто отсутствие имени, а вот для Apple это неприемлимо. Имя должно быть 4 символа латинского алфавита. Следующий фикс этот поправит

```
        <key>FixHeaders</key>
        <true/>
```

Точно такой же фикс был в секции DSDT Fixes, однако, поскольку он относится отнюдь не к DSDT, он вынесен в отдельный пункт секции ACPI.

```
        <key>RenameDevices</key>
        <dict>
               <key>_SB.PCI0.RP02.PSXS</key>
               <string>ARPT</string>
               <key>_SB.PCI0.EHC1</key>
               <string>EH01</string>
               <key>_SB.PCI0.POP2.PEGP</key>
               <string>GFX0</string>
        </dict>
```

В отличие от бинарных патчей ACPI, которые, кстати, действуют не только на DSDT, но и на SSDT также, и родные, и загружаемые, этот метод служит заменой для патча типа Find->PXSX, Replace->ARPT. Но если в секции DSDT->Patches такая замена действует на всем пространстве, то в методе RenameDevices алгоритм будет выискивать только те девайсы, которые лежат на указанном мостике.

Вот такой сложный пример для первой замены

```
Scope(\_SB)
{
  Device (PCI0)
  {
    Device(RP02)
    {
      Device(PSXS)  <- здесь менять
      {
        Method(_ON)
        {
        }
        Method(_OFF)
        {
        }
       }
       PSXS._ON() <- здесь менять
    }
    Scope(RP02)
    {
        PSXS._OFF() <- здесь менять
    }
    Device(RP03)
    {
        Device(PSXS) <- здесь не менять
        {
        }
        PSXS._ON() <- здесь не менять
    }
```

Корректировка DSDT

DSDT - Differentiated System Description Table – самая большая и самая сложная АЦПИ таблица. Минимальная длина 36 байт, реальная – 20кб и больше, 400кб. Эта таблица описывает устройства и методы доступа к ним. Методы доступа могут содержать арифметические и логические выражения, и, таким образом, представляют собой программу на некоем языке программирования, похожим на С своими фигурными скобками. Исправлять эту таблицу значит что-то понимать в программировании. Кловер предлагает некий вариант автоматической правки, но надо понимать, что искусственный интеллект еще не создан, и автоматическая корректировка программ пока еще далека до совершенства. Человек сделает лучше.

А зачем ее нужно исправлять? Весь патч ДСДТ, со времен его основания был нацелен в первую очередь на исправление устройства HPET – High Precision Events Timer. Дело в том,

что в системе OSX присутствует кекст AppleIntelCPUPowerManagement, который служит чтобы управлять питанием процессора (спидстеп), и которому строго необходимо, чтобы в системе был HPET, имеющий прерывания IRQ. Без этого условия кекст уходит в панику. Работать можно только запретив, или удалив этот кекст. Но есть и другой вариант – скорректировать DSDT, и устройство HPET включится как положено! ~~Впрочем, в системе 10.9 требования изменились.~~ Чушь! Патч по-прежнему нужен, и имеет ту же силу. В новых системах, на новом железе уже потерял актуальность, в первую очередь из-за того, что функциональность кекста ушла в ядро.

Это – патч **№1**, жизненная необходимость. Только ли МакОСу нужен этот HPET? Нет, конечно, но производители БИОСов только еще начинают это осознавать и прописывать правильные параметры, до сих пор редко встретишь, чтобы ДСДТ работал без патча.

Момент **№2**. В ДСДТ можно разглядеть некоторые зависимости от операционной системы, "Windows 98", "Windows 2001", "Windows 2006", "Linux", МакОС имеет идентификатор "Darwin", и, как правило, на него ДСДТ не рассчитан. А даже если и рассчитан, то на версию типа FreeBSD. MacOSX является серьезной ACPI системой, т.е. использует ДСДТ по максимуму, так, как его использует Windows 2001, но не Linux, не Windows 98, и не Windows 2006. Правильнее всего сделать мимикрию под Windows 2001. И даже если у вас уже есть "Darwin", добейтесь, чтобы он работал как "Windows 2001". **На многих БИОСах этому соответствует значение OSYS = 0x07D2. Но никак не 0x2410, как это прописано в нативнике. Хотя есть сообщения, что 7D6 или 7D9 для каких-то конфигураций в чем-то лучше.** Нужно смотреть по алгоритму.

На моем скайлейке 2017 лучшее значение оказалось 0x7DA. Потому сделан дополнительный фикс ДСДТ FixDarwin7_10000, который делает мимикрию под Windows 2009, то есть Windows 7. У меня ему соответствует 0x7D9, и это влияет на работу USB3.0, почему-то в БИОСе решили его выключить для системы WindowsXP. Опять же, патч может и не нужен, если далее в ДСДТ эта идентификация не используется. А вы это проверяли?

Момент **№3**. Производитель системной платы, а с ней и его БИОСа, и его ДСДТ, не может предусмотреть, какой процессор будет установлен, какая видеокарточка и другие PCI устройства. А ведь их стоит прописать в ДСДТ! И наоборот, исключить из ДСДТ такие устройства как спикер, флоппи дисковод, параллельный порт. Драйверов на них нет и не нужны. Также часто необходимо добавлять или убавлять коннекторы/порты у каких-то устройств, например, у видеокарты, или у SATA контроллера.

DSDT лежит в БИОСе и используется в системе в бинарном коде AML, существует компилятор/декомпиллятор IASL, который переводит коды в понятный для человека язык DSL. Человеческий путь правки такой AML->DSL->edit->DSL->AML. И тут возникает момент **№4**. Последняя компиляция становится невозможной из-за ошибок, синтаксических и логических, изначально присутствующих в OEM DSDT. В процессе правки требуется и их исправлять. Ну а заодно исправить и смысловые ошибки, из-за которых, например, компьютер не может заснуть, или не может проснуться. А может еще и новые устройства прописать. (А вообще странно, но компиляция/декомпиляция не являются строго обратными операциями, туда-обратно меняет таблицу, а то и вообще, туда идет, обратно нет — требуется вмешательство. Взглядом с моей колокольни это означает, что декомпилятор написан с ошибками, такие уж программисты над ним работали. А несоответствие стандартам нужно отмечать как предупреждения (warning), а не ошибки (error). Если же в исходном АМЛ творится что-то неправильное, то нужно понимать, что компьютер работает, значит это неправильное как-то можно интерпретировать).

Когда мы проделали весь этот путь, мы можем подсунуть загрузчику наш исправленный ДСДТ, положив его в папку /EFI/CLOVER/OEM/xxx/ACPI/patched, или, если OEM имя компьютера еще не известно, в папку /EFI/CLOVER/ACPI/patched, или загружаемая система сама имеет свой вариант ДСДТ, лежащий в корне системного диска.

Где взять исходный ДСДТ, который необходимо патчить? Есть варианты добыть его используя Windows, Linux или даже OSX. Если Кловер как-то удалось запустить, то теперь он и сам предоставляет такую возможность. Надо войти в графическое меню и нажать клавишу **F4**. Если Кловер установлен на раздел FAT32, то ему удастся сохранить все OEM ACPI таблицы, включая нетронутые DSDT и FADT.
Будьте терпеливы, если сохранение происходит на флешку, и таблиц много, процесс может занять заметное время. В текущей ревизии Кловер при таком способе извлекает набор таблиц, ранее недоступных другими способами, в том числе в AIDA64. Есть также способ сохранить на диске вариант препатченного DSDT. Для этого, в интерфейсе Кловера входим в Options Menu, меняем маску DSDT, затем выходим из меню и нажимаем **F5**. Кловер сохранит ваш ДСДТ, поправленный на текущую маску, с именем типа **DSDT-F597.aml**, т.е. патченный с маской 0xF597. Можно сделать несколько вариантов, чтобы потом сравнивать.

Теперь можно брать DSDT файл и редактировать… Ну а для тех, кто не силен в языке ASL, Кловер предлагает проделать некоторые фиксы автоматически. Сразу отвечу и на такой вопрос: «А почему после правки Кловером DSDT все равно компилируется с ошибками?». Да, ДСДТ представляет собой набор описаний и методов, многие из которых нам не нужны. Кловер их не трогает, даже если мы поставим максимальную маску. Ошибки могут быть заложены в тех местах, нетронутых, и они там остаются. Но это не мешает работать всему остальному, ибо ДСДТ работает не как единое целое, а просто как набор описаний и методов.

Рассмотрим фиксы подробнее. В ревизии 4300+ описание фиксов упростилось.

AddDTGP bit(0):

Для описания свойств устройства, кроме варианта DeviceProperties, рассмотренном выше, есть вариант с методом _DSM, прописанным в DSDT.
_DSM - Device Specific Method – хорошо известна заготовка этого метода, который работает в MacOSX начиная в версии 10.5, этот метод содержит массив с описанием устройства и вызов универсального метода **DTGP**, который един для всех устройств.
Данный фикс просто добавляет этот метод, чтобы потом его применять для других фиксов.
Самостоятельного значения не имеет. Видел я совет поставить маску 0x31, мол, остальные фиксы не нужны. Но тогда и (1) не нужна!

FixDarwin bit(1):

Мимикрия ОС Дарвин под систему Windows XP. Многие проблемы со сном и с яркостью растут ногами из неверной идентификации системы.

FixDarwin7 bit(16)

Аналогично, только мимикрия под систему Windows 7. В старых ДСДТ может и не быть проверки под такую систему. У вас есть варианты.

FixShutdown bit(2):

В функцию _PTS добавляется условие: если аргумент = 5 (выключение), то никаких других действий делать не надо. Странно, а почему? Тем не менее, есть неоднократные подтверждения эффективности это патча для плат АСУС, может и для других. В некоторых ДСДТ такая проверка уже есть, в этом случае такой фикс следует отключить. Если в конфиге заложено SuspendOverride=true, то этот фикс будет расширен на аргументы 3 и 4. То есть уход в сон (Suspend). С другой стороны, если стоит HaltEnabler=true, то этот патч наверно уже не нужен.

AddMCHC bit(3):

Такое устройство класса 0x060000, как правило, отсутствует в ДСДТ, но для некоторых чипсетов это устройство является обслуживаемым, а поэтому его нужно прописать, чтобы правильно развести управление питанием PCI шины. Вопрос о необходимости патча решается экспериментально. Еще опыт, это устройство понадобилось на маме с чипсетом Z77, иначе паника ядра на начальной стадии запуска. И наоборот, на чипсете G41M (ICH7) этот фикс вызывает панику. К сожалению, общего правила не видно.

FixHPET bit(4):

Как уже сказано, это главный фикс, необходимый.
Таким образом, минимально необходимая маска патча ДСДТ выглядит как 0x0010

FakeLPC bit(5):

Подменяет DeviceID LPC контроллера, чтобы кекст AppleLPC прицепился к нему. Нужен для тех случаев, когда чипсет не предусмотрен для OSX (например ICH9). Впрочем, родной список чипсетов Intel и NForce настолько большой, что необходимость такого патча очень редка. Проверяется в системе, загружен ли кекст AppleLPC, если нет – патч нужен. Хотя, это тоже еще не факт. Бывает, что кекст сам выгружается из памяти за ненадобностью, хотя чипсет поддерживаемый.

FixIPIC bit(6):

Удаляет прерывание из устройства IPIC. Этот фикс влияет на работу клавиши Power (выскакивающее окошко с вариантами Reset, Sleep, Shutdown).

FixSBUS bit(7):

Добавляет SMBusController в дерево устройств, тем самым удаляя предупреждение об его отсутствии из системного лога. И также создает правильную разводку управления питанием шины, это также влияет на сон.

FixDisplay bit(8):

Производит ряд патчей для видеокарточки. Инжектирует свойства, и сами девайсы, если их нет. Инжектирует FakeID если заказан. Добавляет кастомные свойства. Этот же фикс добавляет устройство HDAU для вывода звука через HDMI. Если задан параметр FakeID, то он будет инжектирован через метод _DSM. (1974)
Патчи для всех видеокарточек, только для не-Интел. Для встроенных интелловских другой бит. FIX_INTELGFX_100000
Также добавляется HDMI устройство (HDAU) если необходимо.

FixIDE bit(9):

В системе 10.6.1 появилась паника на кекст AppleIntelPIIXATA. Два варианта решения проблемы – использование исправленного кекста, либо исправить устройство в ДСДТ. А для более современных систем? Пусть будет, если есть такой контроллер.

FixSATA bit(10):

Фиксирует какие-то проблемы с SATA, и убирает желтизну иконок дисков в системе путем мимикрии под ICH6. Вообще-то спорный метод, однако без этого фикса у меня DVD-диски не проигрываются, а ля ДВД привод не должен быть съемным. Т.е. просто замена иконки — это не вариант!
Есть альтернатива, решаемая добавлением фикса с кексту AppleAHCIport.kext.
Смотрите главу про патч кекстов.

И, соответственно, этот бит можно не ставить! Один из тех немногих битов, которые я рекомендую не ставить.

FixFirewire bit(11):

Добавляет свойство "fwhub" к контроллеру Firewire, если он есть. Если нет, то ничего не произойдет. Можете ставить, если не знаете, нужно или нет.

FixUSB bit(12):

Попытки решения многочисленных проблем с USB. Для контроллера XHCI, при использовании родного или патченного IOUSBFamily, такой патч DSDT незаменим. Эппловский драйвер конкретно использует ACPI, и пропись в DSDT должны быть правильная. Со стрингами пропись в ДСДТ не конфликтует.

FixLAN bit(13):

Инжектирование свойства "built-in" для сетевой карточки – необходимо для правильной работы. Также инжектируется модель карточки – для косметики.

FixAirport bit(14):

Аналогично LAN, кроме того, создается само устройство, если еще не прописано в DSDT. Для некоторых известных моделей производится подмена DeviceID на поддерживаемую. И аэропорт включается без прочих патчей.

FixHDA bit(15):

Корректировка описания звуковой карточки в ДСДТ, чтобы работал родной драйвер AppleHDA. Производится переименование AZAL -> HDEF, инжектируется layout-id и PinConfiguration.

FixRTC

Удаляет прерывание из устройства _RTC. Это необходимо, и очень странно, что кто-то отказывается от такого патча. Если в оригинале прерывания уже нет, то и ничего страшного этот патч не сделает. Однако, возник вопрос о необходимости правки длины региона. Чтобы не было сброса CMOS, нужно длину ставить 2, но при этом в кернел-логе появляется фраза типа ...only single bank...
Не знаю, что плохого в этом сообщении, его можно исключить, если длину поставить 8. Но в этом случае будет риск получить сброс биоса после сна. Поэтому вводится дополнительный ключ для разрешения такого трюка:

```
<key>ACPI</key>
<dict>
    <key>DSDT</key>
    <dict>
        <key>Rtc8Allowed</key>
        <false/>
    </dict>
```

true — длина региона останется 8 байт, если была таковой,
false — будет поправлена на 2 байта, что более надежно предохраняет от сброса ЦМОС.
Как исследовал vit9696 длину региона нужно все-таки оставлять 8, потому что она нужна для сохранения ключа гибернации. А сам по себе фикс полезен. Ну а на десктопах гибернация не нужна, так что может подумать о сбросе ЦМОС.

FixTMR

Аналогично удаляет прерывание из таймера _TMR. Он устарел и Маком не используется.

AddIMEI

Необходимый патч для SandyBridge и выше, заключающийся в добавлении устройства IMEI в дерево устройств, если его еще не было.

FixIntelGfx

Патч для встроенной графики Интел отделен от остальных графических карт, то есть можно поставить инжекцию для Интела и не ставить для Нвидии.

FixWAK

Добавляет Return к методу _WAK. Он обязан быть, но почему-то часто ДСДТ его не содержат. Видимо авторы придерживались каких-то других стандартов. В любом случае этот фикс совершенно безопасен.

DeleteUnused

Удаляет неиспользуемые устройства типа флоппи. А чего, казалось бы, беспокоиться? На самом деле здесь еще удаляются устройства CRT и DVI - совершенно необходимое условие запуска IntelX3100 на ноутбуках Dell. Иначе черный экран, проверено сотнями пользователей.

FixADP1

Корректирует устройство ADP1 (блок питания), что необходимо для правильного сна ноутбука когда он воткнут в розетку, или когда вынут из нее.

AddPNLF

Вставляет устройство PNLF (Backlight), что необходимо для правильного управления яркостью экрана, и, как ни странно, помогает решить проблему со сном, в том числе для десктопа.

FixS3D

Аналогично, этот патч решает проблему со сном.

FixACST

В некоторых DSDT встречается устройство, или метод, или переменная с именем ACST, но это имя используется MacOSX 10.8+ для управления c-state. Совершенно неявный конфликт с очень неясным поведением. Этот фикс переименовывает все вхождения такого имени в нечто безопасное.

Не могу понять, как можно игнорировать эту серию патчей?! Вам что, не нужен хорошо работающий компьютер?

AddHDMI

Добавляет устройство HDAU в DSDT, соответствующее HDMI выходу на видеокарте ATI или Nvidia. Понятно, что раз карточку прикупили отдельно от материнской платы, то в родном DSDT такого устройства просто нет. Кроме этого в устройство инжектируется свойство hda-gfx=onboard-1 или onboard-2 по обстоятельствам:

-1 если `UseIntelHDMI=false`

-2 если присутствует Интелловский порт, который занял порт 1.

FixRegions

А вот это совершенно особый патч. Если остальные патчи предназначались для правки BIOS.aml, для создания хорошего DSDT из ничего, то данный фикс предназначен для окончательной юстировки хорошо сделанного кастомного DSDT.aml, и для BIOS.aml он бесполезен. Дело вот в чем.

В ДСДТ есть регионы, которые имеют свои адреса, например:

```
OperationRegion (GNVS, SystemMemory, 0xDE6A5E18, 0x01CD)
```

Проблема в том, что адрес этого региона создается БИОСом динамически, и он может быть различным от загрузки к загрузке. В первую очередь это было замечено при изменении общего количества памяти, затем при изменении настроек БИОСа, а на моем компьютере зависит даже от предыстории загрузок, например от объема занятого NVRAM. Понятно, что в кастомном DSDT.aml это число фиксированное, а значит, может не соответствовать истине. Самое простое наблюдение - отсутствие сна. После исправления региона сон появляется, но до следующего смещения.

Данный фикс исправляет все регионы в кастомном ДСДТ до значений в BIOS DSDT, и таким образом маска

```
<key>Fixes</key>
<dict>
   <key>FixRegions</key>
   <true/>
</dict>
```

является достаточной маской, если у вас есть хорошо сделанный DSDT со всеми вашими необходимыми фиксами.

Есть еще один патч, но он касается не DSDT, а вообще всех ACPI таблиц, и занимает здесь место незаконно.

FixHeaders

Он будет проверять заголовки не только DSDT, но и вообще всех ACPI таблиц, решая проблему китайских иероглифов в названиях таблиц, которые МакОС не переносит, сразу паникуя. Есть у вас проблема с таблицами, или нет, безопаснее включить этот фикс.

Выбор патчей

Как выбрать, какие патчи необходимы, какие безвредны, а какие опасны? Ну компьютер вы не погубите ни в каком случае. Все это происходит только в оперативной памяти, и будет забыто после перезагрузки. Можно испытывать набор фиксов, исправляя маску в графическом меню, и сохраняя результат по клавише F5 – "Сохранить DSDT-xxxx.aml, скорректированный по текущей маске".
Можно попытаться и загрузиться с текущей маской. Чтобы не мешался настоящий, патченный ДСДТ, уже присутствующий в системе, можно указать в меню

```
DSDT name: BIOS.aml
```

Не найдя такого файла система возьмет OEM DSDT из БИОС и проделает над ним фиксы, согласно установленной маске. В случае неудачи после перезагрузки компьютера текущие установки будут утеряны, и в силу вступят установки по умолчанию, которые у вас работоспособные.

Маска 0xFFFFFFFF соответствует включению всех фиксов, и если OS после этого загрузится, труд программистов потрачен не зря. По описанию выше вы уже сообразили, что некоторые фиксы вам просто ни к чему (например WIFI). Начиная с ревизии 1992 проделана работа по предотвращению паники на двойном патче, так что не бойтесь задавать лишние

биты. Два фикса на настоящее время я бы не рекомендовал использовать: FIX_SATA, бит 0x0400, лучше использовать бинарный патч кекста, и FIX_SHUTDOWN, бит 0x04, ибо вместо него почти всегда работает установка HaltEnabler=true, работающая более корректно. Также опасный патч AddMCHC_0008, кому-то он обязательно нужен, а кому-то категорически противопоказан.

Чтобы увидеть, как DSDT патчи повлияли на результат, предположим, вы не смогли загрузиться, вы можете прописать в конфиге, в секции ACPI, следующий ключ:

```
<key>DSDT</key>
<dict>
  <key>Debug</key>
   <true/>
```

При этом, перед стартом системы на диск будет сохранено два файла в папке
/EFI/CLOVER/ACPI/origin/
DSDT-or.aml
DSDT-pa15.aml
origin — это ваш DSDT загруженный с диска, либо взятый из БИОСа, до применения патчей.
patched - после применения патчей.
Поскольку вы не смогли загрузиться, вы будете делать еще и еще попытки, и патченные файлы будут нумероваться последовательно, не затирая старую информацию. 15 в данном случае — это 15-я попытка загрузки, видимо удачная, нужно смотреть, в чем была проблема 14-й попытки.
И все-таки, рекомендую избегать двойного патча. Может возникнуть и ситуация, когда двойной патч происходит из-за того, что в OEM DSDT уже присутствует метод _DSM, когда мы хотим проинжектировать свой. Значит, вам нужно выставить биты в маске DropOEM_DSM. Смотрите главу «Конфигурирование аппаратной части» → ACPI → DSDT → DropOEM_DSM. Нет, вам нужно переименовать все OEM DSM.

Ручная правка DSDT

Пререквизиты

Нужно иметь какой-то компьютер с какой-то операционной системой, чтобы можно было заниматься редактированием текстовых файлов. Или вы уже на этот компьютер поставили macOS как-то, и теперь желаете усовершенствовать DSDT.

Для любой операционной системы есть компилятор командной строки iasl. В том числе для Виндоус, iasl.exe запускается в командной строке как и в маке, имеет те же функции, и дает те же результаты. Редактировать тексты в Виндах неудобно, notepad не имеет подсветки синтаксиса и нумерации строк, лучше взять Notepad+. В Маке полно вариантов, и Xcode, и BBEdit, и другие. Новую версию компилятора для Мака можно взять здесь Оптимизация Dsdt. Новейший Компилятор.
Для виндоуса на сайте acpica.org Windows Binary Tools
Еще вы должны запастись описанием языка АЦПИ ACPI_6_2.pdf. Лучше брать новую версию, поскольку разбираться нужно с тем ДСДТ, что вам подсунул свежий БИОС.

Создание заготовки

Сначала сделайте себе флешку с Кловером, загрузочную, пусть даже без системы. Важно, чтобы она была в формате FAT32. Грузимся на этом компьютере с этой флешки до интерфейса Кловера. Жмем клавишу "O" (латинская буква Оу) или выбираем в меню значок Options. Заходим в раздел ACPI-> там находим DSDT Name: и вписываем BIOS.aml. Это

точно тот DSDT, который вы смогли бы получить в виндах через Аиду. Спускаемся по меню и выбираем DSDT fixes -> там можно ставить галочки, клавой или мышкой. Ставим почти все галочки, которые не вызывают у Вас отторжения. К примеру не нужны Firewire, Airport, IDE если вы знаете, что у вас таких устройств нет. Все незнакомое лучше поставить. Возвращаетесь из этого подменю, и нажимаете последовательно клавиши F2, F4, F5.

Можете выключать испытуемого, и нести флешку на тот компьютер, на котором будете редактировать файлы. Интересующие нас файлы находятся на флешке в папках

EFI\CLOVER\ACPI\origin
EFI\CLOVER\misc

Копируем эти папки в свою рабочую папку на компьютере, где есть iasl. Хоть с Маком, хоть с Виндой. Инструкции одинаковые, за исключением тонкостей типа наклона слеша.

Декомпиляция

Полученный DSDT.aml есть бинарный файл, его текстовым редактором не посмотришь. Категорически не советую специализированные редакторы maciASL, DSDTEditor и тому подобные, поскольку цель темы не в том, что добрый дядя рехаб за тебя все сделает, а в том, чтобы самому видеть, что и как происходит.

Запускаем командную строку: cmd.exe в винде, Terminal.app в маке, что там в Линуксе не в курсе, вроде bash называется.

```
> cd РабочаяПапка\origin
```

то есть переход в ту самую копию папки, которую сняли с флешки. Положите в эту же папку iasl.exe, если вы в виндах, или установите iasl в систему, если вы в маке

```
$ sudo cp ~/Downloads/iasl /usr/local/bin
```

Теперь, наконец, можно декомпилировать, то есть получить DSDT на читабельном языке.

```
$ iasl -da SSDT*.aml DSDT-*.aml
```

Опция -da делает волшебную вешь, она ищет символы сразу по всем файлам, так что в итоге не должно оставаться неизвестных символов.

Мы декомпилируем тот DSDT, который нам пропатчил Кловер, и таким образом уже имеем массу полезных фиксов. Аналогично можно декомпилировать оригинальный файл

```
$ iasl -da SSDT*.aml DSDT.aml
```

И иметь возможность сравнить, что было и что стало. А получился у нас файл с длинным именем DSDT-1234567.dsl, у вас будут другие цифры и буквы. Это исходная заготовка, которую нужно переименовать в DSDT.dsl, а оригинальный в DSDT-origin.dsl, редактировать и компилировать в бесконечном итерационном процессе:
1. Редактируем текстовым редактором.
2. Компилируем, получаем DSDT.aml

команда компиляции
```
$ iasl -ta DSDT.dsl
```
3. Тестируем, то есть кладем в папку EFI\CLOVER\ACPI\patched и запускаем систему.
4. Возвращаемся к пункту 1.

К сожалению декомпиляция может завершиться со следующей ошибкой
Код:

```
Input file SSDT-10x.aml, Length 0x37F (895) bytes
ACPI: SSDT 0x0000000000000000 00037F (v02 PmRef  Cpu0Cst
00003001 INTL 20120913)
Pass 1 parse of [SSDT]
ACPI Error: [C3ST] Namespace lookup failure, AE_ALREADY_EXISTS
(20160729/dswload-462)
ACPI Exception: AE_ALREADY_EXISTS, During name lookup/catalog
(20160729/psobject-310)
Could not parse external ACPI tables, AE_ALREADY_EXISTS
```

Это означает что описание символа C3ST встретилось в двух разных SSDT, последняя из них это SSDT-10x.
В моем случае она оказалась сходной с SSDT-5x, отличаясь тем, что 5x это ACPI1.0, а 10x это ACPI2.0. А имена одинаковые! То-то у меня в кернел-логе сыплет, что-то вроде этого. Это ошибка БИОСа!
Так же в ноутбуке две одинаковые таблицы. Я проследил, они действительно обе присутствуют в БИОСе, это не моя ошибка.
Что делать? Перед компиляцией удалить дублера, а в реальной работе сделать в конфиге Кловера дроп лишних таблиц. В случае дублей улетят обе.

Что исправлять

1. Ошибки синтаксиса, допущенные производителем данного компьютера
2. Смысловые ошибки
3. Трюки, найденные в интернете. Насчет этого пункта, все думают, что это единственное, что нужно сделать. Нет, первые два пункта тоже важны.

Ошибки синтаксиса

Смотрим сам файл **DSDT**.dsl и в самом файле видим первые проблемы

```
External (_SB_.PCI0.PEG0.VID_.LCD_, UnknownObj)
```
Это означает, что где-то в тексте есть ссылка на объект, а самого объекта нигде нет. Ищем. Он есть в SSDT-7. Получается, что он оттуда не экспортировался. Именно по этой причине к этому ноутбуку предлагают эту **SSDT** включать внутрь общей DSDT. Простым копированием текста, из той SSDT от первой фигурной скобки до последней в хвост DSDT.dsl перед последней скобкой.

Вторая ошибка с отсутствующими символами
```
External (HNOT, MethodObj) // Warning: Unknown method, guessing 1 arguments
```

Поиск показывает, что метод упоминается в такой конструкции

Код:

```
      If (CondRefOf (HNOT))
       {
         HNOT (Arg0)
       }
       Else
       {
         Notify (GFX0, 0x80) // Status Change
       }
```

Тогда все в порядке, если метод неопределен, то и не выполняется. Мне непонятно, а как тогда компилируется? Лучше удалить это, оставить только Notify...

Запускаю компиляцию только что полученного файла (я переименовал DSDT-1F2C3B4D.dsl в более простое DSDT1.dsl, как первая попытка)

```
$ iasl -ta DSDT1.dsl
1 errors, 14 warnings, 91 remarks, 109 optimisations
```

Ошибку надо исправлять, иначе нет результата.
Варнинги тоже надо исправлять, вдумываясь, почему компилятор ругается.
Это еще хорошая ситуация, я встречал и сотни ошибок при первой компиляции.

В данном случае ошибка некритичная, в строке
`Name (_HID, "*pnp0c14")`
формат строки символов недопустимый, исправляется по науке так:
`Name (_HID, EisaId ("PNP0C14") /* Windows Management Instrumentation Device */)`
Исправлять нужно, иначе не скомпилируем, но на работу это не влияет, это виндусовая примочка.

Варнинги реально более критичны, вот образцы исправлений.
- что было
+ что сделал

```
- CreateDWordField (BUF0, \_SB.PCI0._Y0F._LEN, MSLN) // _LEN: Length
+ CreateQWordField (BUF0, \_SB.PCI0._Y0F._LEN, MSLN) // _LEN: Length
```

Подсказка была в логе компиляции
`ResourceTag larger then field (size mismatch tag 64bit, Field 32 bit)`
Правило простое, tag - то, что нужно, field - что нужно исправить

```
tag=1 => CreateBitField
tag=8 => CreateByteField
tag=16 => CreateWordField
tag=32 => CreateDWordField
tag=64 => CreateQWordField
```

Исправлять нужно, чтобы в реальной работе посылаемое значение не заехало на соседнее

поле. Такой глюк никак не отловишь.

Такой варнинг
```
Not all control path return a value
```
Ошибка логики, метод должен что-то возвращать, а получается, что в каких-то случаях ничего не вернет. И? Мак, конечно, не упадет, но и адекватности не ждите.
А что же там написать? Return(Zero) или Return(Local0)?
Чтобы успокоился компилятор, это сойдет, а вообще нужно смотреть логику.

Аналогичный глюк
```
Reserved method should not return a value
```
Код такой
Код:
```
      Method (_SRS, 1, Serialized)  // _SRS: Set Resource
Settings
      {
        Return (BUF2) /* \_SB_.PCI0.A_CC.BUF2 */
      }
```
Открываем ПДФ, упомянутый выше, спецификация АЦПИ, находим поиском метод _SRS, и читаем, что он должен делать. Возврат значений там не предусмотрен.

Поэтому переделываем следующим образом
Код:
```
      Method (_SRS, 1, Serialized)  // _SRS: Set Resource
Settings
      {
        BUF2 = Arg0
      }
```
Так вроде логичнее.

Ну и всем известный варнинг
```
Name (_T_0, Zero)
Use of compiler reserved name _T_0
```

Новый компилятор iasl прекрасно разбирается с этой конструкцией, и не нужно ничего делать. Старый компилятор требовал переименования в TT_0

Смысловые ошибки

Извините, но тут надо быть хоть немного программистом, чтобы их отлавливать. Подсказок ждать неоткуда.

В исходном **DSDT** видим
```
External (CFGD, IntObj)
```
И эту переменную находим в **SSDT** CpuPm. И тут пора вспомнить, что мы эту таблицу дроппаем, вместе с этой переменной!
Надо ее скопировать в DSDT. Как и другие, которые могут пригодиться.
Код:

```
+         Name (CFGD, 0x0066F6FF)
+         Name (PDC0, 0x80000000)
+         Name (PDC1, 0x80000000)
+         Name (PDC2, 0x80000000)
+         Name (PDC3, 0x80000000)
+         Name (PDC4, 0x80000000)
+         Name (PDC5, 0x80000000)
+         Name (PDC6, 0x80000000)
+         Name (PDC7, 0x80000000)
+         Name (SDTL, Zero)
```

Общая ошибка это OEM методы _DSM.
Это не ошибка производителя, это то, что он писал под виндоус, а у нас хакинтош.
Пример

Код:

```
-          Method (_DSM, 4, Serialized)  // _DSM: Device-
Specific Method
            {
-                             Name (_T_0, Zero)  // _T_x:
Emitted by ASL Compiler
               If (Arg0 == ToUUID ("a5fc708f-8775-4ba6-bd0c-
ba90a1ec72f8"))
               {
                 While (One)
                 {
```

Видите UUID? Это из виндусового реестра, в Маке такого нет., и ничего выполняться не будет.
Это было бы пол беды, но если для виндов нормально иметь _DSM и для устройства, и для его мостика, то в Маке это вызывает краш.
Убиваем все чужие _DSM! Простой вариант

Код:

```
-   Method (_DSM, 4, Serialized)  // _DSM: Device-Specific
Method
+   Method (ZDSM, 4, Serialized)
```

Сам по себе метод сохранился, но никто к нему уже не обратится, и паники не будет.

Вот такой любопытный кусок

Код:

```
      OperationRegion (DXHC, SystemMemory, 0xFED1F418, 0x04)
        XHCD,   1
      }

     If ((OSYS < 0x07D6) && (OSYS > 0x03E8))
      {
        XHCD = One
```

```
        Notify (XHC, Zero) // Bus Check
      }
```

Приглядевшись к адресу, я понял, что это Function Disable Bit.
Смысл операции в том, что для систем ниже, чем Windows Vista запретить USB3.
По-моему этот кусок надо вообще вырезать.

А вот это уже лажа писателей
Код:

```
If ((OSYS > 0x07D0) || (OSYS < 0x07D6))
```

Диапазоны складываются и перекрывают вообще любое значение
Скорее там должно быть
Код:

```
If ((OSYS > 0x07D0) && (OSYS < 0x07D6))
```

Тогда диапазоны пересекаются.
Но дальше нужно смотреть, а что там в If, и что в Else.
У меня получилось, что правильнее в Then, поэтому исходную конструкцию
Код:

```
          If ((OSYS > 0x07D0) || (OSYS < 0x07D6))
          {
            Notify (PCI0, Arg1)
          }
          Else
          {
           Notify (GFX0, Arg1)
          }
```

Я сократил до вообще одного оператора
`Notify (PCI0, Arg1)`
Дело в том, что этот If проверяет, является ли система WindowsXP, и делает то, что в Then, для систем типа Windows7,8,10 делает Else.
В чем отличие? В новых системах работает Optimus.
В **macOS** нам нужна нотификация первого типа, на всю шину. И аналогично нужно смотреть в других местах DSDT.

Небольшая детективная история по теме, может кого-то наведет на правильную мысль.
Итак, задача.
Ноутбук спит и просыпается, если не вставлен в розетку. Но тогда батарейка садится.
Если же он в розетке, то сразу, через секунду как заснул, он просыпается. В чем дело?
Именно в ДСДТ!
Итак, ищем, что отличает **ACPI** состояние в розетке и нет. Устройство
Код:

```
    Device (AC)
    {
      Name (_HID, "ACPI0003" /* Power Source Device */)  //
_HID: Hardware ID
```

Декомпилятор нам услужливо подсказал, что это и есть блок питания.
Метод _PSR определяет, является ли это устройство источником питания, либо отдыхает.

Код:

```
    Method (_PSR, 0, NotSerialized)  // _PSR: Power Source
    {
      Local0 = ECG5 ()
      Local0 &= One
      If (Local0 != PWRS)
      {
        PWRS = Local0
        PNOT ()
      }

      Return (Local0)
    }
```

Прочитать об этом можно в книге ACPIspec.pdf, любого года издания

> *An Integer containing the power source status*
> *0 - Off-line (not on AC power)*
> *1 - On-line*

То есть, в моем случае какой-то хардверный метод ECG5() выдает некое число, в котором бит 0 означает воткнутость в розетку.
И сохраняет в переменной PWRS, если там значение отличается.
Теперь по всему коду смотрим, где упоминается PWRS, и находим в методе _PTS (Prepare To Sleep), в том самом, который отвечает за засыпание.

Код:

```
Method (_PTS, 1, NotSerialized)  // _PTS: Prepare To Sleep
{
  P80D = Zero
  P8XH (Zero, Arg0)
  PTS (Arg0)
  If (AOAC & One) {}
  If (Arg0 == 0x03)
  {
    If (PWRS == Zero)
    {
      \_SB.PCI0.XHC.PMEB = Zero
      \_SB.PCI0.EHC1.PMEB = Zero
      \_SB.PCI0.EHC2.PMEB = Zero
      If (\_SB.PCI0.XHC.PMST == One)
      {
        \_SB.PCI0.XHC.PMST = One
      }

      If (\_SB.PCI0.EHC1.PMST == One)
```

```
        {
          \_SB.PCI0.EHC1.PMST = One
        }

        If (\_SB.PCI0.EHC2.PMST == One)
        {
          \_SB.PCI0.EHC2.PMST = One
        }
      }
```

Интересненько здесь!
Условие: если аппарат не воткнут в розетку, то что-то сделать в ЮСБ2 и ЮСБ3.
А если воткнут, то ничего не надо.
В логе неожиданного пробуждения мы и видим XHC, EHC2.
То есть, создатели этого ДСДТ считают, что ноутбук может быть воткнут в розетку только во время работы. Иначе вытаскивай.
Убрав это условие If (PWRS == Zero) я получил решение своей задачи. У меня теперь ноутбук спит и просыпается, и адаптер питания его не тревожит.

Успехов в создании минимально правильного ДСДТ!

Нативный спидстеп

Правильнее говорить Управление Питанием и Частотой Процессора (УПиЧП), по-английски это будет EIST – Enhanced Intel Speedstep Technology, откуда и русское слово "Спидстеп".

Собственно эта тема не столько для загрузчика, как вообще для настройки ХакОС, но поскольку Кловер делает некоторые шаги, то опишем отдельной главой.
Кловер делает не все, что нужно, требуется и немного поработать руками.

Для чего это вообще нужно? Смысл такой: процессор в бездействии работает на минимальной частоте с минимальным напряжением, под нагрузкой скорость и напряжение растут. (А напряжение-то зачем? А потому что фронт импульса становится круче, и потому быстрее набирает уровень, быстрее переходит из состояния 0 в состояние 1).

УПиЧП можно осуществить двумя способами: специализированной утилитой, типа КулБукКонтроллер, или ДженерикЦПУПМ, или же понять нативный спидстеп, благо МакОС это умеет делать.

Следующие шаги необходимы:

1. В ДСДТ обязательно должен быть поправлен HPET, что успешно делается Кловером при маске 0x0010.
2. Должна быть правильная процессорная секция, что делается Кловером при ключе GeneratePStates=Yes (ну и вдобавок DropSsdt)
3. Должна быть выбрана MacModel как образец вашего SMBIOS, для которого предусмотрена технология EIST. Оказывается, не для всех моделей. К примеру для модели MacBook1,1 спидстеп работать не будет, а для MacBook5,1 – будет.

Пункт 3 можно переосмыслить следующим образом: пусть, все-таки модель будет более похожа по конфигурации к настоящей, но исправим ее платформ-плист так, чтобы спидстеп появился.

Для каждой модели существует свой плист, смотрите здесь
`System/Library/Extensions/IOPlatformPluginFamily.kext/Contents/PlugIns/ACPI_SMC_PlatformPlugin.kext/Contents/Resources/MacBook5_1.plist`
Смотрим сходства и различия разных плистов, и исправляем свой в правильную сторону.

ConfigArray

```
<key>ConfigArray</key>
<array>
   <dict>
      <key>WWEN</key>
      <true/>
      <key>model</key>
      <string>MacBook4,1</string>
      <key>restart-action</key>
      <dict>
         <key>cpu-p-state</key>
         <integer>0</integer>
      </dict>
   </dict>
</array>
```

Этот ключ restart-action означает на какой P-State должен свалиться CPU при рестарте. Только при наличии этого ключа заработали сон и выключение компьютера!

CtrlLoopArray

```
<key>CtrlLoopArray</key>
<array>
   <dict>
      <key>Description</key>
      <string>SMC_CPU_Control_Loop</string>
…
      <key>PLimitDict</key>
      <dict>
         <key>MacBook4,1</key>
         <integer>0</integer>
      </dict>
```

Этот ключ PLimitDict уже упоминался в генерации P-states. Повторим: это ограничение максимальной скорости процессора. 0 – скорость максимальна, 1- на одну ступень ниже максимальной. Если же этот ключ здесь отсутствует, то процессор застрянет на минимальной частоте.

CStateDict

```
<key>CStateDict</key>
<dict>
   <key>MacBook4,1</key>
   <string>CSD3</string>
   <key>CSD3</key>
   <dict>
      <key>C6</key>
      <dict>
         <key>enable</key>
         <true/>
```

Практика показывает, что эту секцию лучше всю удалить, чтобы работало управление питанием именно по PState, а не по CState. Хотя, кому как, может и этот вариант стоит проработать.
Симптом – процессор стоит на максимальной частоте, не падает. После удаления секции начинает варьировать частоту.

С процессорами Skylake появилась новая методика Управлением Частоты процессора, по-ихнему называется SpeedShift. Я пока не разобрался, предварительные результаты:

В секции CPU прописываем два ключа (изобретатель - goodwin_c)

```
<key>CPU</key>
<dict>
   <key>HWPEnable</key>
   <true/>
   <key>HWPValue</key>
   <string>0x30002a01</string>
```

Первый ключ ставит 1 в регистр MSR 0x770.
Второй ключ записывает заданное значение в регистр MSR 0x774.

Проблема сна

А что проблема сна? Когда все вышесказанное будет сделано, компьютер будет ложиться спать и просыпаться, как послушный ребенок. Самое главное, необходимое для этого, Кловер уже проделал: скорректировал FADT и FACS. Осталось только поправить ДСДТ, завести спидстеп, пользоваться только хорошими кекстами, и будет вам счастье.

Хорошему сну может помешать любое устройство, в том числе незаведенное PCI устройство, или заведенное частично. К примеру, AppleHDA. Сну категорически мешает NullCPUPM.kext. Вам, может, спидстеп и не нужен, но вы должны так сделать патч HPET, чтобы запустился родной AppleCPUPM, и нуль был не нужен. А тем, у кого процессор не позволяет использовать AppleCPUPM, можно попробовать SleepEnabler — иногда помогает, либо патченное ядро.

В ДСДТ есть группа методов _GPE с нотификациями на каждое устройство, которое нужно пробудить после сна. Сам-то компьютер проснулся, а вот может оказаться, что видео/сеть/звук/мышь забыли проснуться. Смотрите ДСДТ, учите теорию, как это делать.

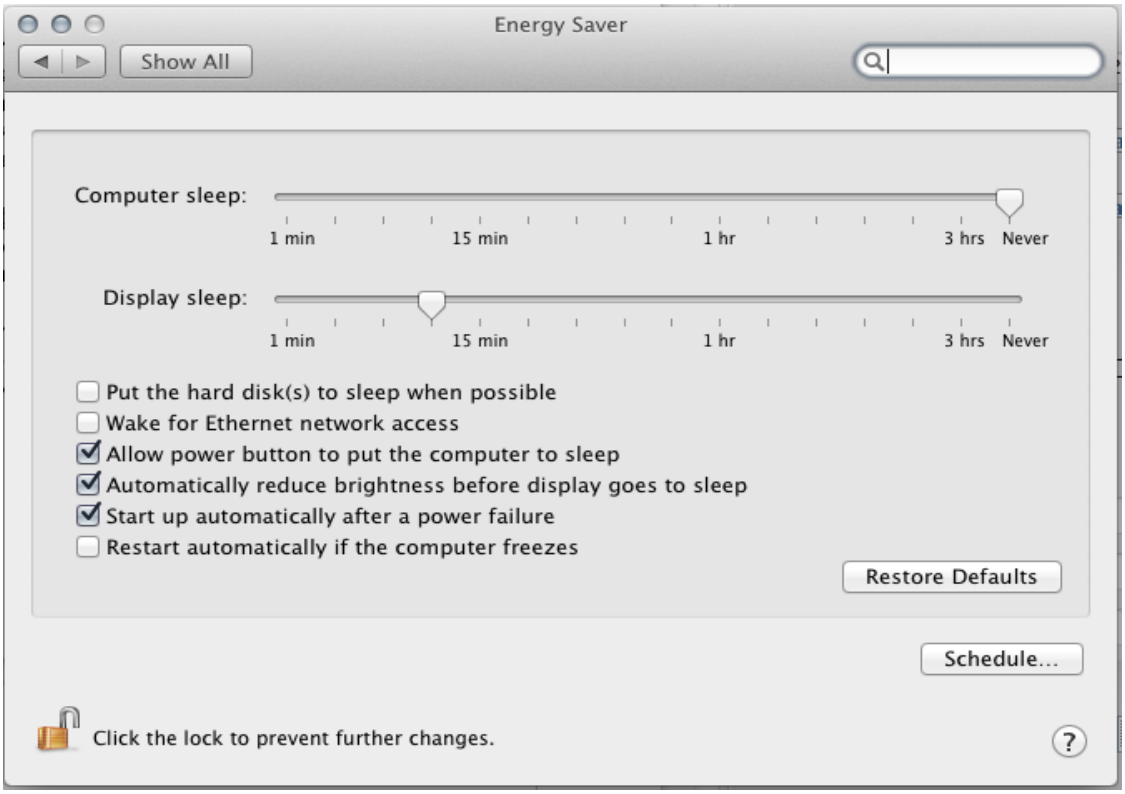
Еще была проблема со сном при UEFI загрузке в систему 10.8.
Проблема решена в ревизии 1942. Изменение в драйвере OsxAptioFixDxe: сон/пробуждение работает даже в 10.8, даже с CsmVideoDxe. Нынче это кирк ProtectMemoryRegions.

Следующий трюк для UEFI загрузки

```
<key>ACPI</key>
<dict>
   <key>HaltEnabler</key>
   <true/>
```

Это поправляет состояние чипсета, неправильно проинициализированного УЕФИ-биосом. Для легаси загрузки CloverEFI делает все корректно, там такой проблемы не встречалось. Симптомы — не уходит в сон, экран гаснет, а вентилляторы нет.

И еще один трюк.



Вот без галочки "Start up automatically..." я никак не мог добиться пробуждения после сна.

Гибернейт

Его еще называют глубокий сон, а вообще больше похоже на клиническую смерть. Суть в том, что систему вроде отправляют в сон, но она сохраняет свое состояние в файле sleepimage, и просто выключает компьютер, чтобы потом, при включении, он просто восстановил свое состояние и проснулся. Для ноутбуков это имеет решающее значение. При обычном сне компьютер выключается не полностью, и продолжает использовать электроэнергию, хоть и меньше, чем в рабочем состоянии, но все же заметно, чтобы аккумулятор полностью разрядился за некоторое время сна. При гибернейте аккумулятор не используется, и разряжается только в силу своих утечек.

Когда-то давно с Хамелеоном гибернейт работал, но только до версии 10.7.2 (вроде), потом в силу каких-то изменений в системе такая методика перестала работать. В Кловере удалось сделать гибернейт, но при следующих условиях:

- Загрузка либо CloverEFI (legacy), либо InsydeEFI, либо Phoenix 2.3.1. В текущей ревизии 2915+ появился драйвер OsxAptioFix2Drv, который позволяет иметь гибернейт с AMI UEFI на системе 10.9.1+. Но система 10.7.5 вообще не грузится с этим вариантом. Да и бог с ней!
- Система либо 10.7.5, либо 10.9.1+. Другие системы пока не просыпаются.
- Мода 21 или, лучше, 29 или даже 57, хотя Эппл настаивает на 25.
  `sudo pmset -a hibernatemode 29`
  Начиная с Капитана так поставить 29 невозможно. Трюк следующий:
  - копируете /Library/Preferences/com.apple.PowerManagement.511CE201-1000-4000-9999-120361221216.plist на Рабочий Стол. Таких файлов несколько, выбираете тот, который соответствует UUID в Информации О Системе. Редактируете этот файл, чтобы занести туда 29

```
<key>Hibernate Mode</key>
<integer>29</integer>
```

И копируете отредактированный файл с помощью терминала обратно
```
sudo cp ~/Desktop/ com.apple.PowerManagement.511CE201-1000-4000-9999-120361221216.plist /Library/Preferences/
```
После этого необходима перезагрузка, только тогда изменение вступит в силу.

- Если же у вас работает настоящий, железный NVRAM, то можете сделать моду 25. И тогда в конфиге прописываем
```
<key>Boot</key>
<dict>
 <key>StrictHibernate</key>
 <true/>
```
- Для системы 10.13 и выше нужно включить
```
<key>RtcHibernateAware</key>
<true/>
```
потому-что ключ шифрования может быть прописан в CMOS, и потому что Кловер должен сформировать дополнительные NVRAM переменные. (насчет последнего еще надо подумать.

Работает это следующим образом:
1. Ставим моду 29 (или 25), если еще не выставлена. Повторять нет необходимости.
2. Отправляем компьютер в сон либо через меню, либо закрыв крышку, либо нажав кнопку питания, если на это настроено. Через минуту компьютер полностью погаснет.
3. Чтобы пробудиться, просто включаем его, как обычно. Видим заставку БИОСа, входим в меню Кловера. А вот здесь мы видим, что наша система имеет пометку `(hibernated)`

Тогда как другие системы - нет. При нажатии на эту иконку происходит загрузка системы из имиджа, несколько секунд, внизу виден прогресс, и система включается. Это гораздо быстрее, чем нормальная загрузка системы, особенно для ноутбуков, и особенно при большом количестве открытых приложений.
Надо заметить, что если файловая система тома подвергалась модификации после гибернейта, например из системы, загруженной со второго раздела, то возникнет серьезная опасность повреждения файловой системы, ибо в спящей системе есть кеш с другой структурой. Для системы 10.9 эта сложность преодолевается автоматически путем сравнения даты модификации. В системе 10.7.5 это не работает, следите за правильностью вручную.

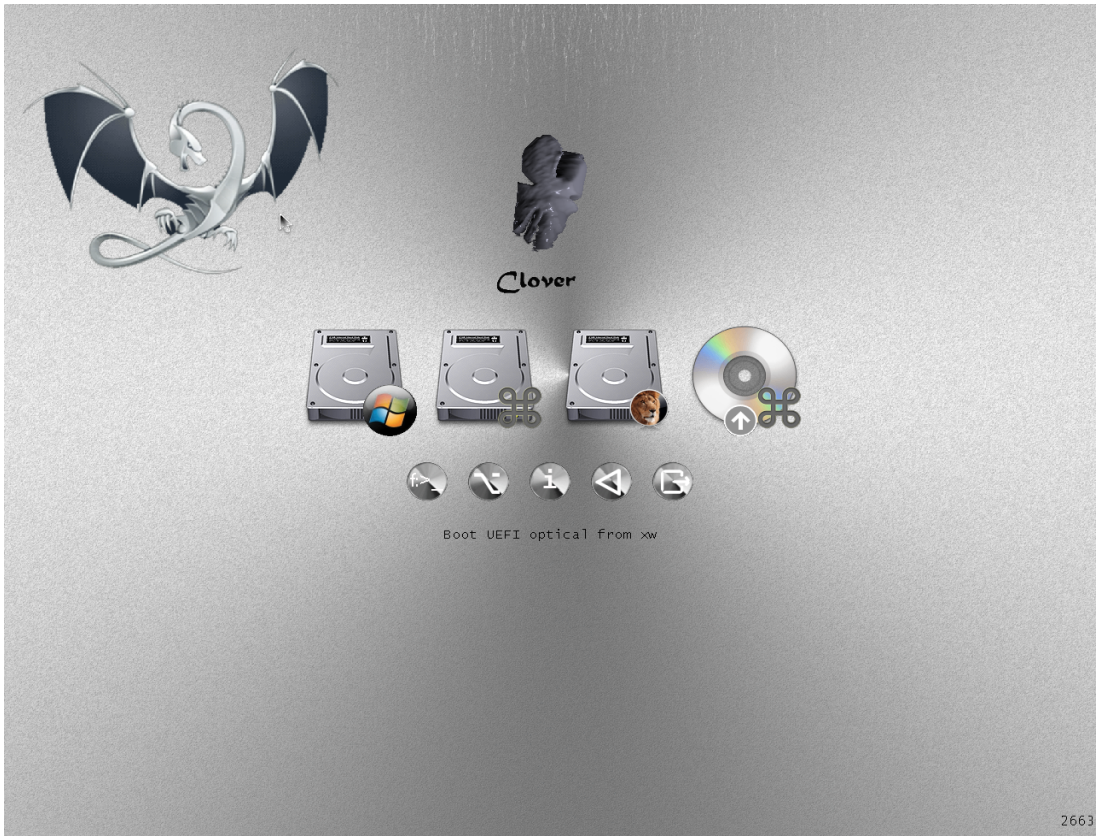
Вы можете отменить пробуждение из имиджа путем нажатия на пробел на этой иконке, и выбрав пункт "Cancel hibernation".
Если же система и дальше продолжает считать, что надо просыпаться, то придется прописать в конфиг:

```
<key>Boot</key>
<dict>
  <key>NeverHibernate</key>
  <true/>
```

Как пользоваться

Первое знакомство

Во-первых, загрузитесь в GUI Кловера и попробуйте, для начала, пожить здесь, понажимать разные клавиши, подвигать мышью.



Верхний ряд кнопок — это предполагаемые операционные системы, которые можно загрузить. На данной картинке их две, Lion и Windows, что видно по картинкам. Реально, я напоминаю, Кловер не является загрузчиком операционных систем, он является менеджером их собственных загрузчиков. А именно, для Мака загрузчиком является /System/Library/CoreServices/boot.efi. Для виндоус, в данном случае, /EFI/microsoft/boot/bootmgfw.efi

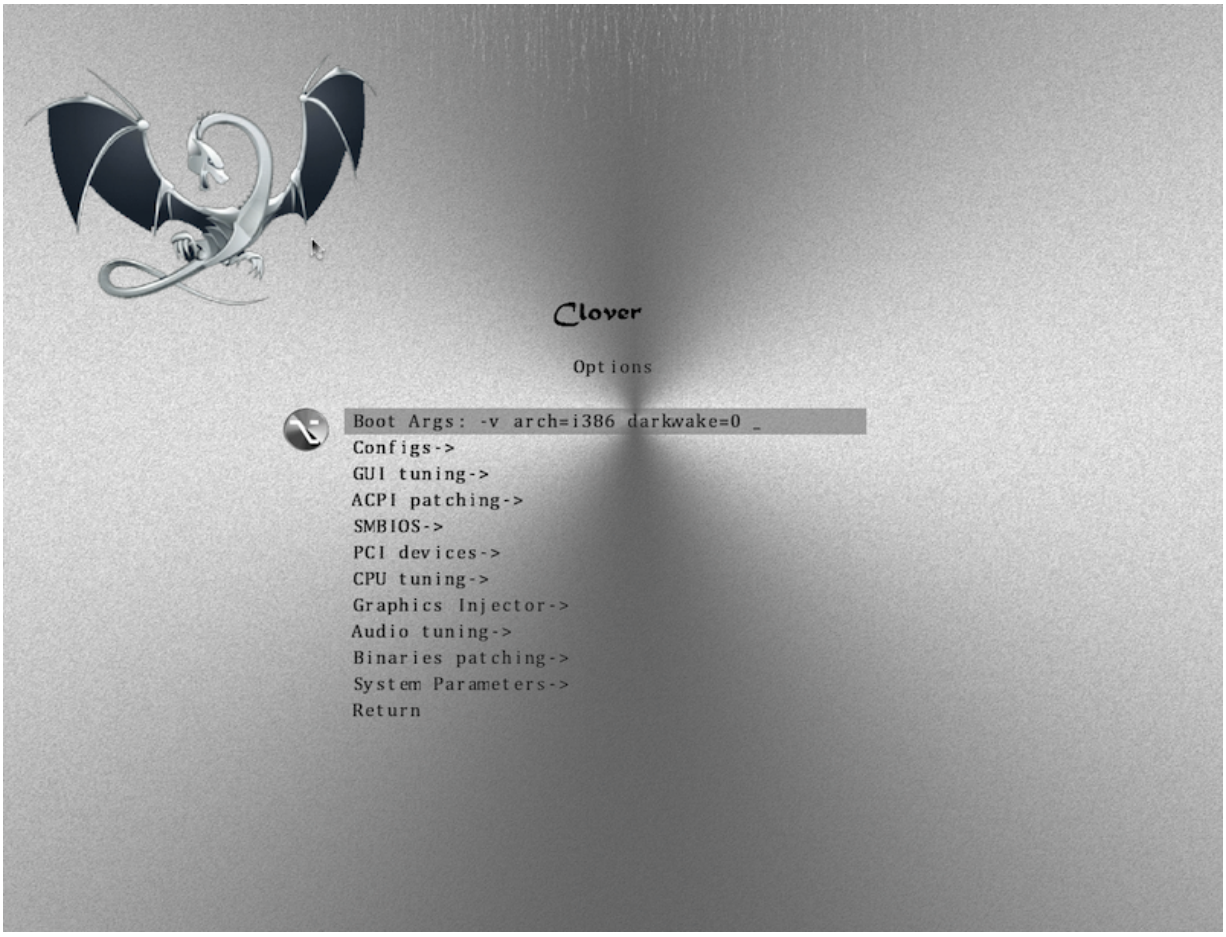
Нижний ряд кнопок — дополнительные функции: командная строка (Shell), меню Options, информация о загрузчике и среде, рестарт и выход из Кловера. Выход куда? Обратно в среду EFI, в УЕФИ БИОС или в CloverEFI соответственно.

Очень полезно для начального знакомства нажать F1 (кто бы мог подумать?!). Если в конфиге указано

```
<key>GUI</key>
<dict>
   <key>Language</key>
   <string>ru:0</string>
```

то и справка будет на русском языке
Командная строка это нечто наподобие ДОСа, с возможностью копирования и удаления файлов. Как и для чего - это выходит за рамки этой книги. Это Shell.efi со своим help.

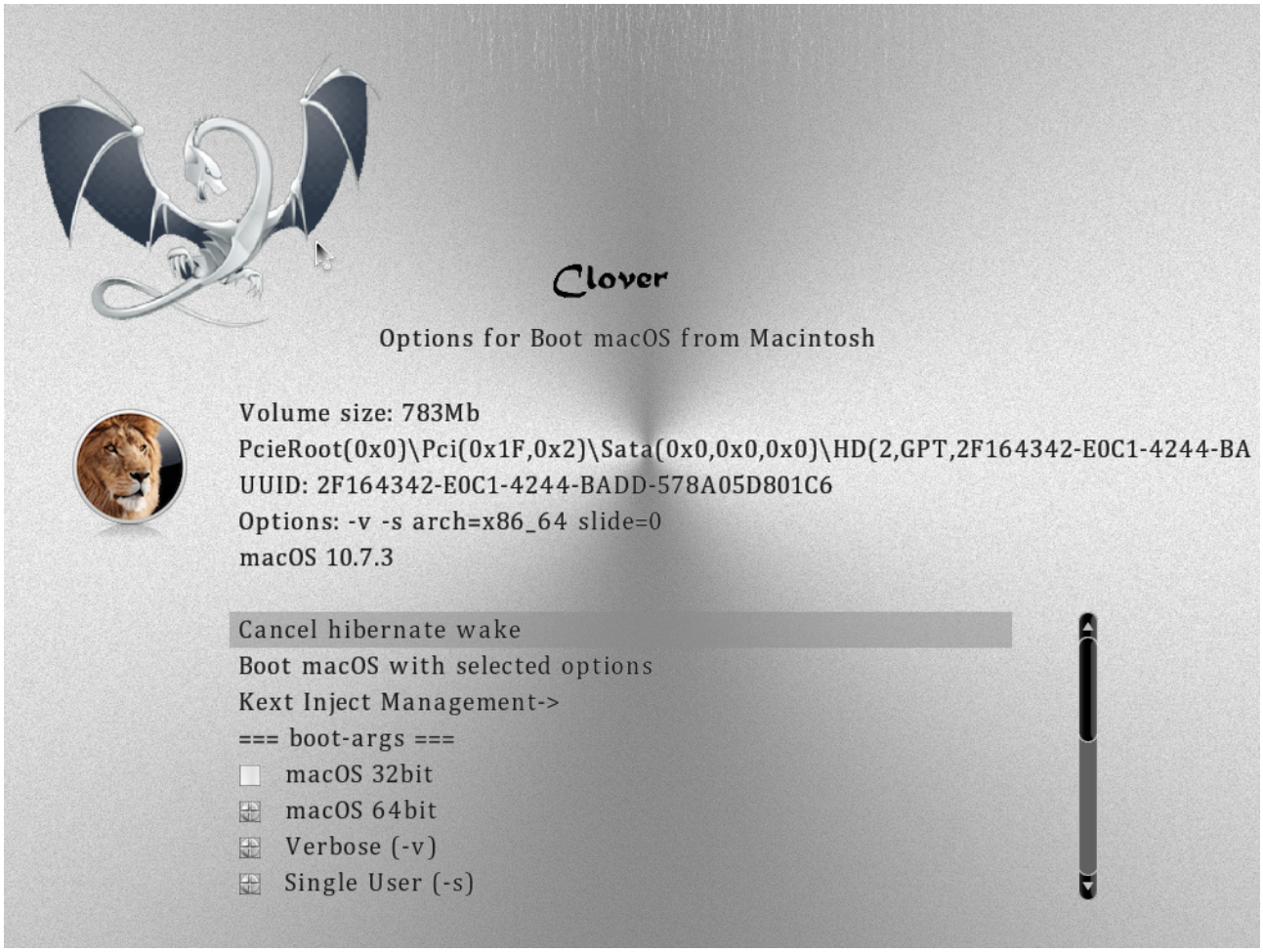
Меню Options (настройки) позволяют менять некоторые настройки, которые повлияют на ход загрузки системы.



Часть из них задана в файле config.plist, но вполне могло получиться, что там написано неправильно, и, чтобы не редактировать пока этот файл, настройки можно поменять уже при запущенном Кловере.

Что именно менять и для чего, это зависит от задачи, что именно надо получить. Очень раздражают просьбы типа «у меня хдд вестерн дигитал и память корсар, помогите настроить конфиг». Конфиг настраивается не по железу, а по результату. Если не можете сразу загрузиться, попытайтесь определиться, в чем может быть проблема, и исправляйте в этом меню.

Несколько специальных способов загрузки можно получить, нажав пробел на иконке загрузчика. (еще раз, ENTER - загрузка системы, SPACE — вход в дополнительное меню загрузок).



В частности, только здесь можно предотвратить выход из гибернейта, если он нежелателен.

Почему Кловер так медленно запускается?

Некоторые даже не могут дождаться запуска, сообщают, что Кловер не работает. Посмотрим на этот вопрос повнимательнее.

1. **Выставлен debug.log**.

```
<key>Boot</key>
<dict>
   <key>Debug</key>
   <true/>
```

Да, это очень ценная информация, чтобы узнать, что не работает и почему. Но имейте ввиду, что это очень тормозит запуск. Если Кловер установлен на ЮСБ (для пробы, так сказать, мы ведь не верим, что Кловер вообще способен работать?!), то запуск с дебагом может длиться 10 минут. Да, реально. Это происходит потому, что дебаг-лог открывается, закрывается и перезаписывается на флешке на каждой строке. Это гарантирует, что вы получите информацию об успешности запуска даже при насильственном резете, так что, если вы пошли на дебаг, ждите! Или для первой попытки поставьте <false/>.

2. **Слишком много дисков, разделов и файлов на них**.

Кловер, чтобы составить меню запуска, должен просканировать все диски, все разделы, и все файлы на них, чтобы найти, какие системы можно предложить для запуска. Ну ждите! Либо отмените сканирование,

```
<key>GUI</key>
```

```
   <dict>
      <key>Scan</key>
      <dict>
         <key>Entries</key>
         <false/>
         <key>Legacy</key>
         <false/>
         <key>Tool</key>
         <false/>
      </dict>
```
и составьте меню вручную
```
   <key>GUI</key>
   <dict>
      <key>Custom</key>
      <dict>
         <key>Entries</key>
         <array>
            <dict>
```
Правда, это требует какого-то умственного усилия, чтобы понять, что туда писать. Либо оставьте пока сканирование загрузчиков, чтобы хоть куда-то загрузиться.
```
      <key>Scan</key>
      <dict>
         <key>Entries</key>
         <true/>
```
3. **Громадный Windows или Linux раздел, и даже не один**.
Сканирование виндоус-раздела производится при наличии NTFS драйвера. А в этом разделе, как правило, миллион файлов, и среди них мы собираемся искать bootmgr.efi.
Я бы рекомендовал устанавливать Windows так, чтобы этот файл лежал на разделе EFI, а драйвер NTFS.efi вообще удалить, и, таким образом, не сканировать виндоус-разделы. Аналогично с Линукс-разделом и драйвером VBoxExt2.efi.
4. **Слишком много драйверов в папке /EFI/CLOVER/drivers/**
Я предвижу ситуацию, когда люди начнут производить свои драйвера такого типа, и найдутся желающие попробовать их. На данный момент подумайте, нужны ли вам лишние драйвера клавиатуры, мыши, LowMemoryFix... Дело в том, что если вы запускаетесь с ЮСБ, то чтение всех этих файлов может занять время.
5. **Неподдерживаемая мышь**.
К сожалению, не все мыши поддерживаются EFI драйвером, который у нас имеется. Плохая мышь может вести себя на экране неправильно, либо дать жесткие тормоза.
Запретите ее для проверки, а то и навсегда, если она неправильная
```
      <key>Mouse</key>
      <dict>
         <key>Enabled</key>
         <false/>
         <key>Speed</key>
         <integer>0</integer>
      </dict>
```
6. **Медленный драйвер HFS+**.
В комплекте с официальным Кловером идет драйвер VboxHFS.efi, который лицензионно чистый и понимает линки, но работает медленнее, чем эппловский HFSplus.efi. Скачайте где-нибудь этот неофициальный, но быстрый драйвер, и положите его в папку **/EFI/CLOVER/drivers/UEFI/.** Это касается и легаси (..**drivers/BIOS/**) загрузки.
7. **Выбрана чудовищно красивая тема оформления**.
Чем тема богаче красками и анимацией, тем дольше она грузится. Выберите встроенную тему, она самая скоростная
```
   <key>GUI</key>
   <dict>
      <key>Theme</key>
      <string>embedded</string>
```

8. **Пользуйтесь самым новым Кловером**.
Что бы там не шептали добрые советчики, но новая версия лучше старой, и в ней исправлены баги, в частности, вызывавшие зависания Кловера. А начиная с ревизии 3063 появились сообщения прямо на экране о процессе загрузки.

Надписи кривоваты, потому что хороший шрифт еще не загружен, и это вызвало немало нелестных откликов "Как удалить надписи на экране?". Поставьте timeout=0, и надписей не будет. А для начала они очень полезны, немало новых пользователей поняло, что Кловер все-таки работает, просто медленно. Для них и написана эта глава.
Или так

```
<key>Boot</key>
<dict>
   <key>NoEarlyProgress</key>
   <true/>
```

Далее рассмотрим некоторые приемы, специальные патчи и методы работы, собранные по принципу обратного словаря. Есть проблема → вот решение.

Анализируем Debug.log/Preboot.log

Зачем? Кловер даст вам информацию о железе, которую вы не увидите из характеристик на сайте, и еще расскажет о том, как он справляется с конфигурированием, прежде чем запустить систему.

Как его получить? Сразу оговорюсь, что это одно и то же, с разницей в успехе и скорости. Чтобы получить preboot.log, зайдите в оболочку Кловера (ГУИ) и нажмите клавишу F2. Файл preboot.log будет сохранен в папке Кловера EFI/CLOVER/misc/. Файл заканчивается на фразе Enter GUI, что логично. Однако, если вы загрузите систему, то файл будет заканчиваться на запуске самой системы, то есть все, что можно.
Если же Кловер виснет, то поможет debug.log. Чтобы его получить, выставляем в конфиге

```
<key>Boot</key>
<dict>
   <key>Debug</key>
   <true/>
```

В этом случае файл создается построчно, и гарантируется, что вы найдете в нем точку остановки. Пример:

```
222:890  0:001  DefaultIndex=-1 and MainMenu.EntryCount=7
222:892  0:001  SetScreenResolution: 1366x768 - already set
0:100  0:100  MemLog inited, TSC freq: 2591578780
```

```
0:100  0:000  CPU was calibrated with ACPI PM Timer
```
Здесь мы видим, что после сообщения о смене разрешения экрана начался новый лог. То есть ошибку надо искать в тексте Кловера после этого сообщения, и до следующего сообщения, которого не было. Сообщите эти строчки разработчику, он будет искать.

Разберем теперь, что мы видим в логе.

```
MemLog inited, TSC freq: 2591583140
```
Здесь начальная тактовая частота процессора. По стандарту процессор запускается с одним ядром на максимальной нетурбо частоте. Если это не так, срочно бейте в колокола! Работать не будет. Такие ситуации встречаются редко, и с каждым разбираемся индивидуально с особыми поправками Кловера. Цифра некруглая, и это правильно! Круглой она будет только в утилитах, которые производят округление для простоты показа.

```
Now is 14.7.2019,  8:47:7 (GMT)
```
Дата и время запуска по Гринвичу. В Москве местное время будет +3. Отличаем старый лог от нового.

```
0:100  0:000  Starting Clover revision: 5120 (master, commit dddceaae3) on American
Megatrends EFI
```
Здесь и ниже мы узнаем, какой версией Кловера воспользовался юзер, а также у него UEFI загрузка, как в данном случае, или легаси (было бы Clover EFI).

```
=== [ Get Smbios ] ====
```
Дальше идет расшифровка модулей памяти, как их распознал БИОС. Обычно правильно, поэтому ставим в конфиге
```
   <key>SMBIOS</key>
   <dict>
      <key>Trust</key>
      <true/>
```
А если вам значения не понравились, то false. Кловер тогда сам будет определять путем чтения SPD или XMP. В SMBIOS значения тоже берутся из чтения SPD, по алгоритмам БИОСа. У нас хорошие алгоритмы, но универсальные, а в БИОСе конкретные под эту материнскую плату, так что не факт, кто точнее.
```
   <key>Boot</key>
   <dict>
      <key>XMPDetection</key>
      <string>-1</string>
```
Смотрите в главе про конфигурирование.

```
Running on: 'Latitude E6430' with board '0H3MT5'
```
А это информация, как называется ваш компьютер.

```
=== [ GetCPUProperties ] ====
```
Здесь информация о процессоре, и CPUID, и его собственное имя
```
BrandString = Intel(R) Core(TM) i5-3320M CPU @ 2.60GHz
```
И другие важные характеристики.
```
MSR 0xE2 before patch 1E008404
```
Видите восьмерку? Это залочка, ужасный грех для Мака. Если есть возможность, перепрошейте БИОС на разлоченный. Если нет, то жить вполне можно, поставив патчи
```
   <key>KernelAndKextPatches</key>
   <dict>
      <key>KernelPm</key>
      <true/>
      <key>AppleIntelCPUPM</key>
      <true/>
```
Кловер, по идее, сделает это автоматом, но лучше если вы вручную это сделаете.

```
Turbo: 31/31/31/33
```
Эти цифры означают, что на одном ядре достижима частота 3300МГц, на двух ядрах и более только 3100. Поскольку в Маке включены все ядра, то не надо выдвигать претензию, что процессор в маке не разгоняется до максимально возможной частоты 3300.
У меня, однако, есть и еще одна строчка
```
 The CPU not supported turbo
```
Это, скорее всего, БИОС запретил использование турбо частот. Надо смотреть настройки БИОСа.

=== [GetDevices] ===
Очень люблю эту секцию, она рассказывает, какие PCI (PCIe) устройства найдены в компьютере, вместе с их адресами и DeviceID/VendorID. Очень помогает для выяснения, можно ли найти драйвера под это устройство
```
0:100  0:000  PCI (00|00:00.00) : 8086 0154 class=060000  MCH
0:100  0:000  PCI (00|00:01.00) : 8086 0151 class=060400
0:100  0:000  PCI (00|00:02.00) : 8086 0166 class=030000  VideoCard
0:100  0:000   - GFX: Model=Intel HD Graphics 4000 (Intel)
0:100  0:000  PCI (00|00:14.00) : 8086 1E31 class=0C0330  USB3.0
0:100  0:000  PCI (00|00:16.00) : 8086 1E3A class=078000  IMEI
0:100  0:000  PCI (00|00:16.01) : FFFF FFFF class=FFFFFF
0:100  0:000  PCI (00|00:16.03) : 8086 1E3D class=070002  SerialPort
0:100  0:000  PCI (00|00:19.00) : 8086 1502 class=020000  LAN
0:100  0:000   - LAN: 0 Vendor=Intel
0:100  0:000  PCI (00|00:1A.00) : 8086 1E2D class=0C0320  USB2.0
0:100  0:000  PCI (00|00:1B.00) : 8086 1E20 class=040300  HDA
0:100  0:000  PCI (00|00:1C.00) : 8086 1E10 class=060400
0:100  0:000  PCI (00|00:1C.01) : 8086 1E12 class=060400
0:100  0:000  PCI (00|03:00.00) : 14E4 4353 class=028000  WiFi
0:100  0:000   - WIFI: Vendor=Broadcom
0:100  0:000  PCI (00|00:1C.02) : 8086 1E14 class=060400
0:100  0:000  PCI (00|00:1C.03) : 8086 1E16 class=060400
0:100  0:000  PCI (00|00:1C.05) : 8086 1E1A class=060400
0:100  0:000  PCI (00|0C:00.00) : 1217 8221 class=080501  SD-reader
0:100  0:000  PCI (00|00:1D.00) : 8086 1E26 class=0C0320  USB2.0
0:100  0:000  PCI (00|00:1F.00) : 8086 1E55 class=060100  LPC
0:100  0:000  PCI (00|00:1F.02) : 8086 1E03 class=010601  SATA AHCI
0:100  0:000  PCI (00|00:1F.03) : 8086 1E22 class=0C0500  SMBUS
0:100  0:000  PCI (00|00:1F.06) : FFFF FFFF class=FFFFFF
```
некоторые устройства еще и прокомментированы.
FFFF означает, что устройство неподключено, хотя как-то присутствует.
Class устройства это Video, Audio, USB2.0, USB3.0, LAN, WiFi и так далее. Все расшифровки известны из спецификаций PCI. Я в правом столбце дописал расшифровки класса, в логе их нет. Класс 060400 это мостик, слот, куда можно вставить устройство.
Здесь нет USB устройств. Кловер не дорос, чтобы еще и их сканировать.
А вот еще момент, из чужого лога
```
- GFX: Model=GeForce GTX 760 family CE (Fermi)
```
Что мы видим?! 760 должна быть Кеплер! Почему Кловер говорит, что она Ферми?
Кловер прав, его не обмануть перешивкой БИОСа, информация о семействе видеокарты берется не из БИОСа, а из функционирования видеоядра.

```
0:128  0:027  EFI\CLOVER\#.plist not loaded with name from LoadOptions: Not Found
0:143  0:015  EFI\CLOVER\config.plist loaded: Success
```
Эта возможность Кловера практически никем не используется. Штука в том, что в БИОСе есть место, куда прошить название файла config.plist, и таким образом можно прямо в БИОСе выбрать, с каким конфигом загружать Кловер. В данном случае такого имени не найдено, и потому грузится стандартный config.plist. А вообще можно пользоваться, если вспомнить инструкции 2014 года от Дмазара.
Ниже в логе есть и другой перечень

```
0:346  0:003  === [ Found config plists ] ================================
0:403  0:057  - config0.plist
0:403  0:000  - config1.plist
0:403  0:000  - config.plist
```
Это уже для меню Кловера, чтобы оттуда поменять конфиг. Но секции Boot, GUI и KernelAndKextPatches уже не поменяются. Поздно!

```
0:143  0:000  === [ GetListOfThemes ] ==================================
0:162  0:018  - [00]: embedded
0:176  0:014  - [00]: random
0:177  0:001  - [00]: metal
0:188  0:010  - [01]: BGM
0:223  0:035  - [02]: CESIUM
0:285  0:061  - [03]: METAL@2X
0:298  0:012  - [04]: Clovy
0:343  0:044  - [05]: BOOTCAMP
```
Это понятно, перечень всего установленного. Разве что убедиться, что в конфиге выбрана тема, которая реально есть.

```
KextsToPatch: 13 requested
KernelToPatch: 1 requested
```
Небольшой перечень, что в конфиге выставлено для патчей кекстов и ядра. Смотрим в чужой конфиг, и критикуем, зачем он сделал те или иные патчи.

```
0:749  0:002  === [ LoadDrivers ] ======================================
0:861  0:111  Loading ApfsDriverLoader.efi  status=Success
0:883  0:021   - driver needs connecting
0:885  0:002  Loading AudioDxe.efi  status=Success
0:893  0:007   - driver needs connecting
0:895  0:002  Loading DataHubDxe.efi  status=Success
0:917  0:022  Loading EnglishDxe.efi  status=Success
0:926  0:009  Loading Fat.efi  status=Success
0:935  0:008   - driver needs connecting
0:937  0:002  Loading FSInject.efi  status=Success
0:944  0:007  Loading OsxAptioFix3Drv.efi  status=Success
0:952  0:007  Loading SMCHelper.efi  status=Success
0:959  0:007  Loading VBoxHfs.efi  status=Success
0:966  0:007   - driver needs connecting
0:968  0:002  4 drivers needs connecting ...
0:970  0:002  PlatformDriverOverrideProtocol not found. Installing ... Success
0:974  0:004  APFS driver loaded
0:978  0:003  Searching for invalid DiskIo BY_DRIVER connects: not found, all ok
```
И следующий повод для критики, зачем юзер загружает эти драйвера, и почему не загружает другие.

```
SetScreenResolution: 1366x768 - already set
```
Для своего экрана я заказал это разрешение. И оно успешно выставлено.
У кого желание расходится с реальностью, смотрим главу конфигурирование.

```
4:481  0:002  === [ GetMacAddress ] ====================================
4:561  0:080  MAC address of LAN #0= D4:BE:D9:6C:86:CD:
```
Кловер умеет читать МАК-адрес практически любой сетевой карты. Используем эту информацию, чтобы выставить свое значение переменной ROM. На некоторых UEFI BIOS это не работает, ищем другие способы.

```
=== [ ScanSPD ] ====
```
Проверка модулей памяти, если мы не доверяем БИОСу.

```
=== [ GetAcpiTablesList ] ====
```

Список таблиц ACPI, найденных в БИОСе. Пригодится, если захотите выбросить какую-то (drop)
```
- [06]: SSDT  CpuPm len=2850
```
Тут есть и ID = CpuPm и длина 2850. Дроппать можно и так, и так, в зависимости от уникальности.

```
=== [ GetUserSettings ] ===
```
некоторая выборочная информация, что именно выставлено в конфиге, особенно полезно для чтения чужих логов.

```
=== [ ScanVolumes ] ===
```
Перечень томов с их адресами и UUID. Том это или раздел, или целый диск.
Полезно посмотреть, когда "Кловер не видит моего раздела!".

```
=== [ InitTheme ] ====
```
Дальше информация об успешном создании графического интерфейса с выбранной темой.
Я, к примеру, вижу
```
OSicon os_mav not parsed
```
то есть в моей выбранной теме нет иконки Маверикса, будет использована просто иконка Мака.
Здесь также информация о стартовом звуке, который зависит от темы.

```
6:511  0:002  === [ Dump SMC keys from NVRAM ] ==========================
6:570  0:059  found AppleSMC protocol
6:584  0:014  Registered 17 SMC keys
```
В большинстве случаев SMC ключи на старте не играют роли. Они строго необходимы для FileVault2 и для Гибернации. Обеспечиваются драйвером SMChelper.efi и инфраструктурой Кловера. (есть еще вариант VirtualSMC со своей инфраструктурой).
Для инсталляции системы вроде не нужны, но… кто его знает!

```
=== [ ScanLoader ] ===
```
А вот тут уже перечень, что откуда можно загрузить. А также информация, если какая-то система находится в состоянии гибернации.

```
=== [ GetEfiBootDeviceFromNvram ] ====
```
От успеха этой операции зависит, будет ли автостарт Кловера по таймауту. Смотрим инструкции с соответствующей главе.
Успех выглядит так
```
Boot redirected to Entry 3. 'Boot macOS from HighHD'
```
То есть у меня по таймауту загружается система с диска HighHD.

```
=== [ StartLoader ] ===
```
Начинаем загрузку системы
```
GetOSVersion: 10.13.6 (17G7024)
```
На посмотреть, о загрузке какой системы говорит юзер.

Далее идет информация, что именно производит Кловер перед загрузкой выбранной системы, какие патчи, какие свойства сгенерированы, какие кексты загружены, и последняя строчка разблокировка USB2.0, если необходима.
```
USB EHCI Ownership for device 1E26 value=1000001
```

В версии 5120 лог продолжается со значениями из процедур патча ядра и кекстов. Это для разработчиков.

Запуск OSX на неподдерживаемом железе

Вообще-то вся книга про ЭТО. Я здесь расскажу частично, отталкиваясь от вопроса.

Неподдерживаемый БИОС. Еще бы! Именно про Хакинтоши мы и идем речь. И в первую очередь это данные в DMI, которые содержат имя производителя (должно быть Apple inc.), модель и серийный номер, цифры и буквы в котором неслучайны, они что-то означают, в частности модель и дату производства. В простейшем варианте, еще со времен Неткаса, модель всем ставили MacPro3,1, и некий серийник, один на всех, который работал. Сейчас Кловер, проанализировав железо, предлагает десятки вариантов, которые работоспособны. Тем не менее, рекомендуется сгенерить свои серийники, а может и взять модель, отличную от модели по умолчанию.

Неподдерживаемый процессор. Да, разные версии МакОС поддерживают разные наборы ЦПУ, и ваш процессор может оказаться неподдерживаемым.
Вот такая таблица по старым системам:

CPU name	CPUID	10.4	10.5.8	10.6.3	10.6.8	10.7.2	10.7.5	10.8.5	10.9.5
Yonah	0x0006E6	1	1	1	1	1	0	0	
Conroe	0x0006F2	1	1	1	1	1	1	1	
Penryn	0x010676	0	1	1	1	1	1	1	
Nehalem	0x0106A2	0	1	1	1	1	1	1	
Atom	0x0106C2	0	0	0	0	0	0	0	
XeonMP	0x0106D0	0	0	0	1	0	0	0	
Linnfield	0x0106E0	0	0	1	1	1	1	1	
Havendale	0x0106F0	0	0	1	1	1	1	1	
Clarkdale	0x020650	0	0	0	1	1	1	1	
AtomSandy	0x020660	0	0	0	0	0	0	0	
Lincroft	0x020670	0	0	0	0	0	0	0	
SandyBridge	0x0206A0	0	0	0	1	1	1	1	
Westmere	0x0206C0	0	0	0	1	1	1	1	
Jaketown	0x0206D0	0	0	0	1	1	1	1	
NehalemEx	0x0206E0	0	0	1	1	1	1	1	
WestmereEx	0x0206F0	0	0	0	1	1	1	1	
Atom2000	0x030660	0	0	0	0	0	0	0	
IvyBridge	0x0306A0	0	0	0	0	0	1	1	
Haswell	0x0306C0	0	0	0	0	0	0	1	
IvyBridgeE5	0x0306E0	0	0	0	0	0	0	0	
HaswellMB	0x0306F0	0	0	0	0	0	0	1	
HaswellULT	0x040650	0	0	0	0	0	0	1	
CrystalWell	0x040660	0	0	0	0	0	0	1	
и так далее									

То есть, поддержка Yonah и XeonMP прекращена; чем новее процессор тем новее система требуется; Атом не поддерживался никогда, хотя с виду обычный Интел процессор. Табличка устарела, смотрите по исходникам XNU. Skylake, к примеру, поддерживается в 10.11.6 и выше.

При запуске системы на неподдерживаемом процессоре вы получаете панику ядра. Для ее предотвращения служит патч KernelCpu=true. Он просто заменяет вызов паники на пустой оператор, и все продолжает работать. Насколько корректно? Ну хотя бы работает! В новых ревизиях Кловера я сделал патч FakeCPUID=0x010676. Или другие цифры, подходящие для вашей системы, и близкие к вашему процессору (примерно того же поколения, например Атом стоит подменить Пенрином, или даже Конроем). Подмена происходит в ядре на уровне вызова процедуры get_cpu_info() и таким образом окажет влияние на те кексты, которые обращаются к ЦПУ за информацией, вместо того, чтобы самим вызывать CPUID. Например так работает AppleIntelCPUPowerManagement.kext, и на него действует этот патч.
Пример:

```
<key>KernelAndKextPatches</key>
<dict>
   <key>FakeCPUID</key>
```

```
<string>0x010676</string>
```

Неподдерживаемая видеокарта.

Интел. Поддерживаются: GMA950, X3100, HD3000, и выше. Увы, никакие подмены не помогают. Для каждого варианта существует свой набор патчей, и если видеокарта другая, то в лучшем случае вы будете иметь картинку, без возможности смены разрешения, и без всяких 3D эффектов. Жить, в принципе, можно, но вот невозможность калибровки цвета экрана меня не устраивает, потому что даже с фотографиями на таком компьютере работать невозможно. В системе 10.14 поддерживаются HD4000 и выше. Но HD4000 не поддерживает 10 бит/цвет, а вот Skylake HD530 уже да!

Nvidia. Карточки 7300-7600 поддерживаются только до системы 10.7.5 в 32-битном режиме. Про более старые карты говорить наверно бесполезно.

Есть некоторые вопросы к карточкам Ферми серии 4xx/5xx. Их тоже следует отнести к частично поддерживаемым, и только до системы 10.11.6. В случае с Nvidia также проконтроллируйте кест AppleGPUPowerManagement, в нем также может быть ID вашей или похожей карты.

Для систем 10.12 и выше работает нативно только **Кеплер**, семейство GK. Для более новых карт нужен WEB драйвер, который существует только до системы 10.13.6. Это карты GTX 6xx-7xx, но не все, среди них встречаются также Ферми, тогда облом. Для работы Нвидии требуется трюк с МакМоделью, либо с подстановкой BoardID. Для систем 10.14 и выше веб-драйверов нет, то есть Максвеллы и Паскали вообще в пролете. С другой стороны кто-то заводит в Мохаве старые Tesla, видимо Эппл оставила лазейку для своих старых компьютеров.

ATI/AMD. Целая история. И о том, как я заводил Radeon9000IGP, и о кексте от dong для X1500, и о кексте Callisto, и о сложных рецептах патча коннекторов для современных карточек. Смотрите в этой книге. Для Радеонов сделано очень много, ищите, читайте, не будьте чайниками! На системах 10.13 и выше Радеоны серий 6000 и ниже не работают. У них нет поддержки Металла, и потому драйвера в системе неполноценные, а то и вообще не включаются.

Теперь вот еще WhateverGreen от vit9696 — "драйвер всех видеокарт, просто поставьте самую свежую версию, а также самую свежую версию Лилу, и ни о чем не думайте. В Кловере нужно отключить все, что связано с видеокартами". Но я так не играю! Кто не хочет разбираться, пожалуйста, пользуйтесь. Для других мы разбираем пошагово, что нужно для завода видеокарты.

Карты 5700 работают только начиная с Каталины. Для Мохаве, к примеру, список возможных карт: 550-590 и Вега.

Звуковая карта. Профессиональные карты, как правило, имеют драйвера для Мака. Чипсетные кодеки стандарта HDA поддерживаются все с кекстом VoodooHDA. Родным кекстом AppleHDA не поддерживается ни один кодек из присутствующих на рынке. Когда-то был ALC885, но нынче он не встречается. Зато хакеры разработали методику патча AppleHDA так, чтобы он поддерживал практически любой нужный реалтековский чип (то есть ALCxxx). Кловер помогает поправить DSDT под этот кекст, и предлагает способы патча кекста на лету. Что именно патчить и как читайте на форумах. HDMI Audio работает с патчами ДСДТ, предусмотреными в Кловере, однако, не работает с некоторыми картами AMD. Зато в системах 10.13+ появился новый драйвер AppleGFXHDA.kext, изучайте!

Сетевая карта. Во-первых, эпповские драйвера поддерживают целый ряд чипов. Во-вторых, для сетевых карт программисты научились писать кексты, и драйвера существуют для большинства известных карт. В некоторых случаях достаточно сделать FakeID для карты, чтобы она попала в список поддерживаемых родными драйверами, но в большинстве случаев нужен отдельный кекст.

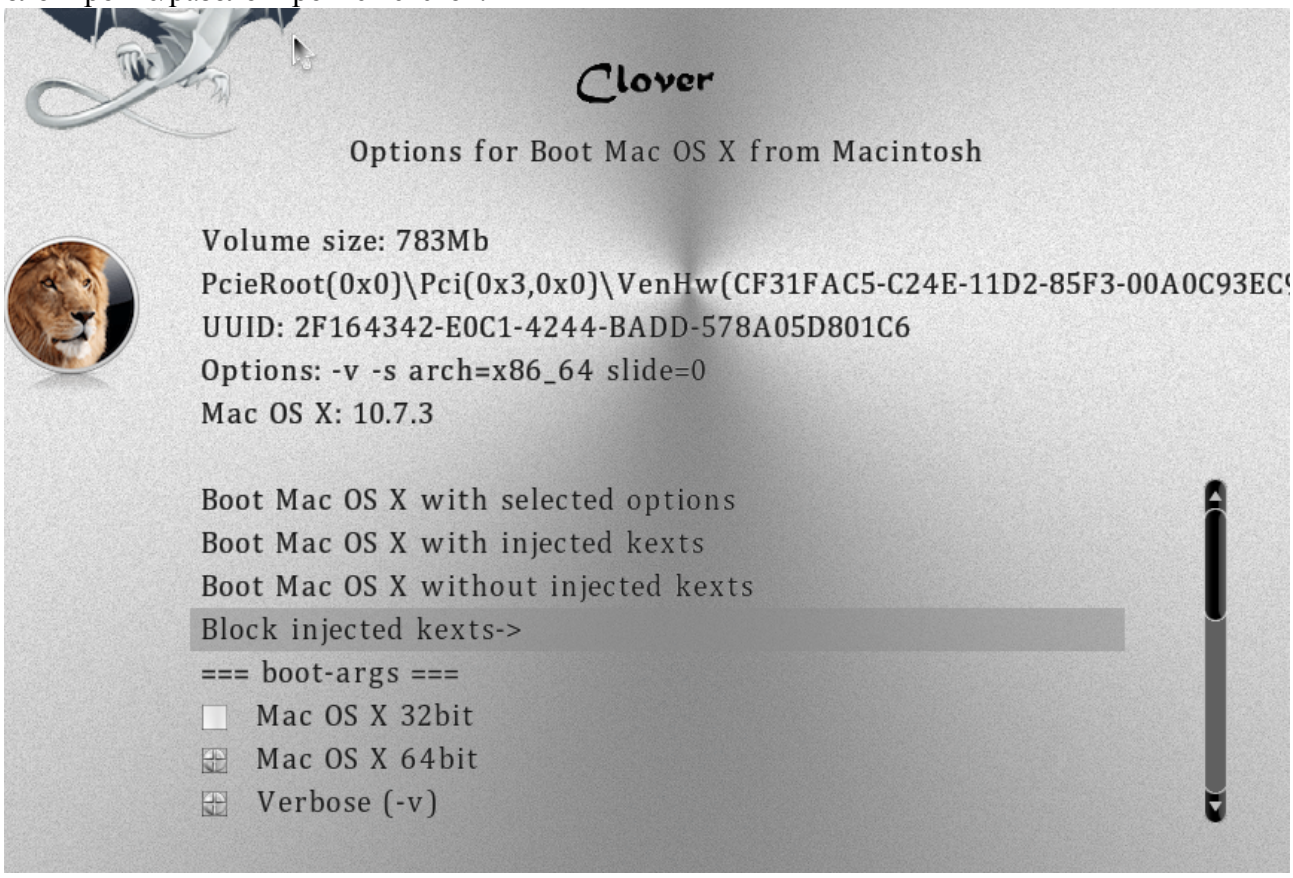
WiFi. А вот тут все очень грустно. Поддерживаются некоторые Broadcom, Atheros и Ralink. Смотрите на форумах информацию про каждую конкретную модель. ~~Intel вообще никак.~~ Кловер может помочь с FakeID, например в моем варианте подмены Boadcom4315 на поддерживаемый 4312. А также Atheros с соседними номерами.
С 2020 года в сети появился драйвер для некоторых карт Intel WiFi. Шансы есть!

Блокировка кекста

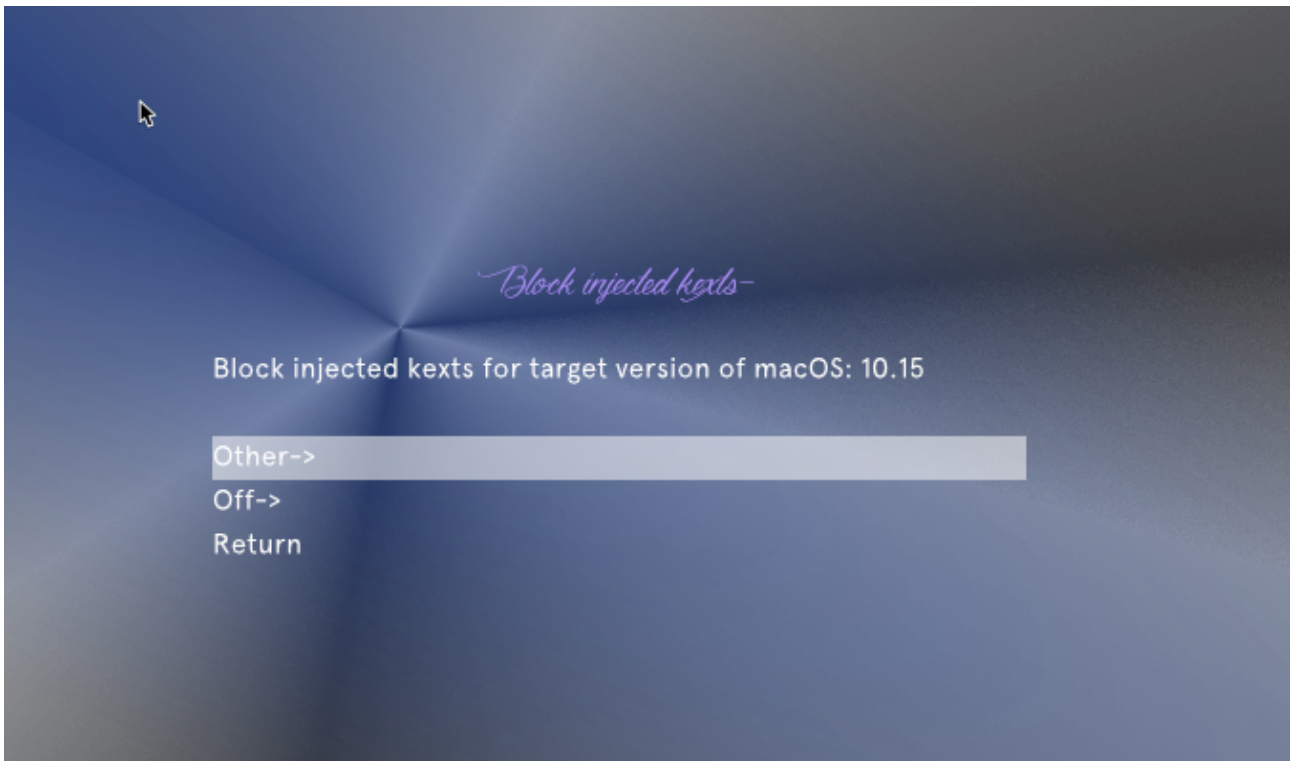
Случилось мне установить кекст Geenna.kext в системную папку SLE. На экране паника, после перезагрузки этот кекст загружается в первую очередь, и тут же паника. Ну и что теперь делать? Его надо удалить, но на этом компьютере еще нет другой системы. Для этой цели в Кловер введена дополнительная функция: в Options Menu в третьей строчке вводите **Block kext: Geenna** и спокойно загружаете систему в single user mode (пробел на иконке системы). Кекст не успеет загрузится, ибо он блокирован. В этом текстовом режиме

```
fsck -fy
mount -uw /
rm -r -v /S*/L*/Ex*/Geenna.kext
reboot
```

Ребут здесь необходим, иначе ядро все равно загрузит этот кекст следующим шагом, и опять будет паника.
В новом Кловере есть новая методика: все свои кексты кладем в папку EFI/CLOVER/kexts/Other. А в меню Кловера Details (клавиша "пробел") найдем подменю по блокировке/разблокировке кекстов.



Начиная с ревизии 5052 появилась возможность хранить пока ненужные кексты в папке *Off*, и подключать их в том же меню по необходимости



При этом кексты в папке Other подключены по-умолчанию, а в папке Off выключены, однако видны Кловером для возможности подключения.

Имя слота (AAPL,slot-name)

Это в основном косметика, хотя есть утверждения, что это обязательно надо в каких-то случаях.

Откуда система берет имя слота? По старому его пытались инжектировать через _DSM свойство «AAPL,slot-name», но это совершенно неправильный метод ибо лечит следствие вместо причины. Это свойство выставляется системным кекстом AppleSMBIOS на основании ACPI свойства _SUN и таблиц DMI. То есть _SUN задает ID в диапазоне 0-255, по которому находится таблица SMBIOS тип 9 с соответствующим ID, откуда берется имя слота и другие его свойства.
Смотрите главу про заполнение конфига, секция SMBIOS->Slots

HDMI звук

Все, что нужно для AppleHDA, расследовал Toleda, но не каждый захочет разыскивать его объяснения на английском. Я сделал патчи DSDT, при его участии, чтобы максимально приблизиться к его результату.

Принципиально есть два варианта HDMI устройства.
1. На внешней видеокарте АТИ или НВидия. В системе оно значится как звуковое устройство класса HDA = 0x0403, и обслуживается тем же звуковым драйвером. Надо только, чтобы и у видеокарты, и у HDMI было одинаковое свойство «hda-gfx=onboard-1». А может и не заработать! Неподдерживаемое устройство.
2. На встроенной карте Intel есть HDMI разъем, но устройства такого нет, используется звук от чипсетного HDA. В этом случае нужно прописать в конфиге

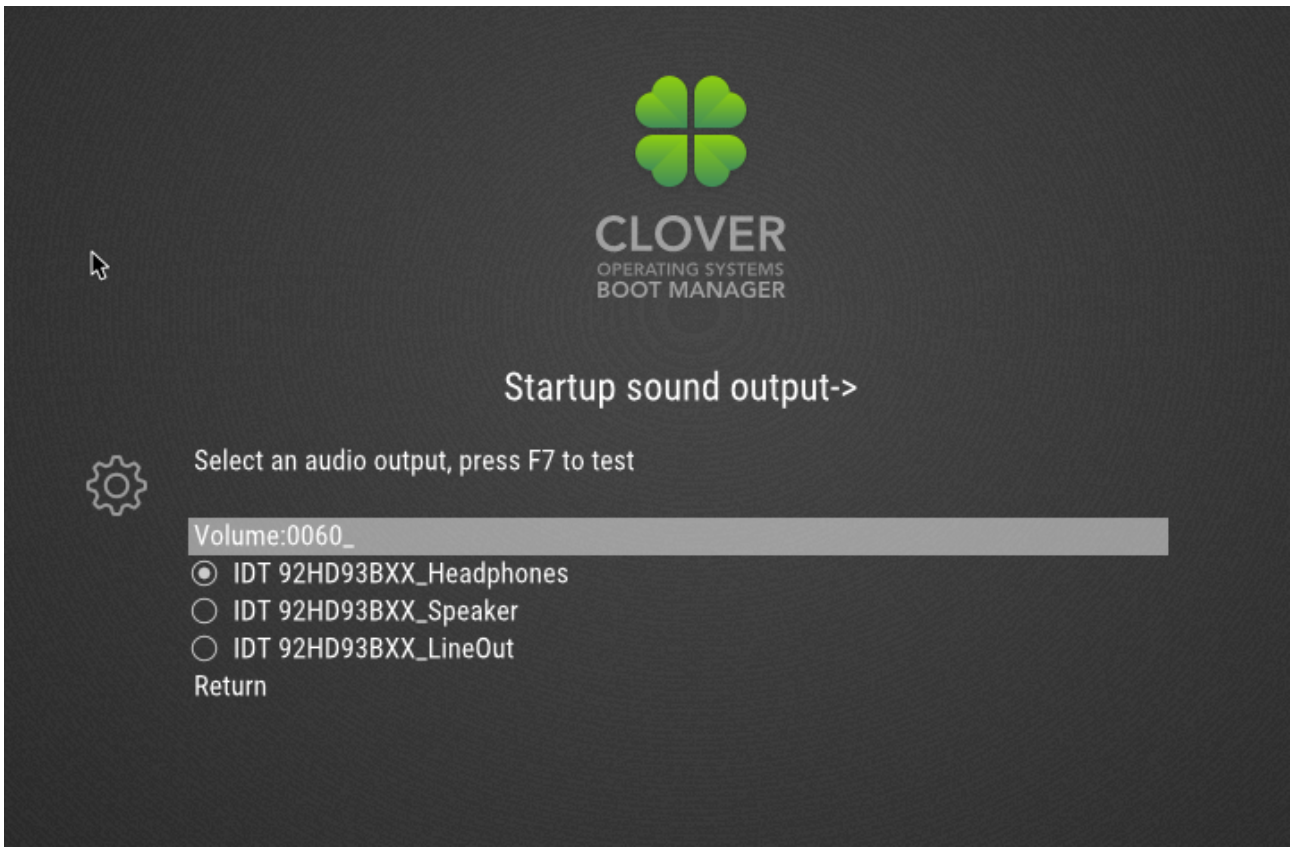
```
<key>Devices</key>
```

```
  <dict>
    <key>UseIntelHDMI</key>
    <true/>
```
При этом звук от АТИ или НВидии станет «onboard-2».
Для DSDT необходимые фиксы: FixDisplay_0100, FixHDA_8000, AddHDMI_8000000
3. Вариант, что встройка используется только для IQSV. Тогда (iMac18,3)
В устройстве HDEF есть свойство No-hda-gfx
В IGPU нет ничего (встройка для IQSV)
В GFX0 (который Радеон) стоит hda-gfx="onboard-1"
И в HDAU оно же.
Автоматически этого Кловер пока не делает, используйте массив Properties.
Замечу, что все эти дополнительные свойства нужны только для AppleHDA. Драйвер VoodooHDA не нуждается во внешних подсказках.

Стартовый звук компьютера

Это изобретение Goldfish64. Он написал драйвер EFI для звука HDA, и сделал утилиты для настройки звука, тестирования и изготовления дампа звукового кодека. https://github.com/Goldfish64/AudioPkg
Но он вставил звук на момент запуска boot.efi, перехватив его вызов загрузчиком. То есть, задумка, чтобы это работало не только с Кловером, а с любым EFI загрузчиком, не касаясь его внутренних кодов. А мне интереснее, чтобы звук работал до входа в интерфейс Кловера, или даже во время того, как я прогуливаюсь по его меню. Лицензия открыта, так что я переделал его под свои желания. Все настройки и тесты можно выполнить самим Кловером, с его графическим меню, а собственно драйвер я включил в репозитарий Кловера, чтобы он не потерялся, и чтобы в дальнейшем его можно было улучшать, не спрашивая автора, который может исчезнуть, не сегодня, так завтра.

Итак, для работы звука необходимо:
1. Использовать Кловер ревизии 4871+. Работало и в прошлых ревизиях, но в багами.
2. Положить драйвер AudioDxe.efi в папку EFI/CLOVER/drivers/BIOS или drivers/UEFI соответственно. Или в обе. Этот драйвер, поставляемый с Кловером, уже отличается от авторского оригинала, пока не принципиально, но я бы рекомендовал именно свою версию.
3. Положить звуковые файлы с именами sound.wav и sound_night.wav в используемые вами темы. Таким образом стартовый звук зависит от выбранной темы. sound_night.wav не обязателен, если его нет, то и ночью будет играть sound.wav. Эти файлы должны быть формата RIFF/WAV, 2 channels, 16bit little endian unsigned int, частота дискретизации может быть 8, 44.1, 48кГц, от этого зависит размер файла. Сама по себе звуковая поддерживает 44 и 48кГц, а то и больше. Для компактности я разрешил и 8кГц, и Кловер конвертирует такой файл на лету в 48кГц. Качество неизбежно меньше, но для такого случая оно не особо и нужно. Зато звук такого формата удалось упаковать прямо в Кловер, и он используется для тестирования выхода.
4. Зайти в интерфейс Кловера, Startup sound output→, и протестировать, какой из выходов будет играть



В первой строчке регулируем громкость звука от 0 до 100. Это проценты, больше 100 не бывает. Значение 0 означает, что звук не будет проигрываться. То есть, это не то, что кричит с закрытым ртом, а это то, что попытки мяукнуть не производится. Кстати не знаю, шкала линейная или логарифмическая.
Следующие строчки скомбинированы из модели чипа и его выхода. Если у вас несколько звуковых карт, как часто бывает кроме встроенной есть еще HDMI, то вы все увидите в этом списке, со всеми их выходами. Выбирайте, нажимайте F7, слушайте. После выхода из этого меню выбранная настройка будет сохранена в NVRAM, в том числе и эмулируемый в переменные Clover.SoundVolume, Clover.SoundDevice, Clover.SoundIndex. Здесь у меня отличие от золотой рыбки, он сохраняет настройки в приватной области со своим UUID, что невозможно для эмулируемой памяти, для легаси-Кловера. Мои настройки будут видны из системы, могут быть удалены или модифицированы из системы, а префикс Clover обеспечивает отсутствие конфликта с интересами Эппл. На данном этапе Кловер прочитает рыбьи настройки, если своих еще нет, но в дальнейшем будет использовать только свои.
На следующей перезагрузке вы услышите звук перед загрузкой оболочки Кловера, но после надписи ...scan entries…
В конфиге эти настройки не вынесены, ни к чему. В любом случае вы сначала должны протестировать, а значит уже записать настройки в HBPAM. Но в конфиге есть одна настройка, оставшаяся от тестового периода. PlayAsync=true.
Если фалс, то пока играет звук, ничего не работает. Вроде так и на настоящике.
Если тру, то звук воспроизводится на заднем плане, не мешая всему остальному. Я поставил длинный звуковой файл, и слушал музыку. Появился ГУИ Кловера, музыка играет, я зашел в меню, и выбрал грузиться в вербозе. Музыка играет. Я нажал "загрузить систему" и смотрю сообщения: вот отработал boot.efi, музыка играет, вот запустился кернел, музыка продолжает играть! И только после нескольких загруженных кекстов она заткнулась, наверно это был VoodooHDA, который переинициализировал звуковой чип. В загруженной системе никаких

проблем не обнаружил. В ревизии 4862 использовать асинхронный звук было нельзя, он вис. В ревизии 4871 проблема решена, и теперь можно использовать асинхронный звук по-умолчанию.

NVRAM, iMessage, multiboot

Вопрос об использовании системой энергонезависимой памяти NVRAM с помощью функций GetVariable() и SetVariable() собственно поднимал я еще в 2010 году http://www.projectosx.com/forum/index.php?showtopic=1504
Тогда я пытался внедрить работу с ним в Хамелеон в собственном бранче, но никакой поддержки не получил. Никому это не надо было, хотя мой аргумент насчет контрольной панели "Загрузочный Диск" оставался неотразимым. Тогда гуру объяснили мне, что это есть в загрузчике ДУЕТ, поэтому, начав проект Кловера на основе ДУЕТа я в первую очередь поставил целью обеспечить эту функциональность.

В Хамелеоне эти функции есть, но они сделаны очень просто "return Unsupported", так чтобы система, запущенная с Хамелеоном не паниковала, и просто не отзывалась на вызов этих функций. Это до поры до времени работало, за исключением панели StartupDisk. Но вот сервис iMessage уже отказался работать в таком варианте. Не сработала никакая подстановка и эмуляция. Я склоняю голову перед Меклортом, который в течении месяца все же придумал способ сделать такую функциональность в Хамелеоне, с помощью модуля FakeNVRAM.dylib и какой-то матери.

Что подразумевается под работоспособностью NVRAM? Если система хочет сохранить какую-то переменную до следующей перезагрузки, она записывает ее в NVRAM с помощью функции SetVariable(...). Мы также можем сохранять свои переменные с помощью утилиты nvram:

```
sudo nvram MyVar=qu-qa-re-ku
```

после перезагрузки эта переменная должна быть известна в системе с помощью команды чтения

```
nvram MyVar
```

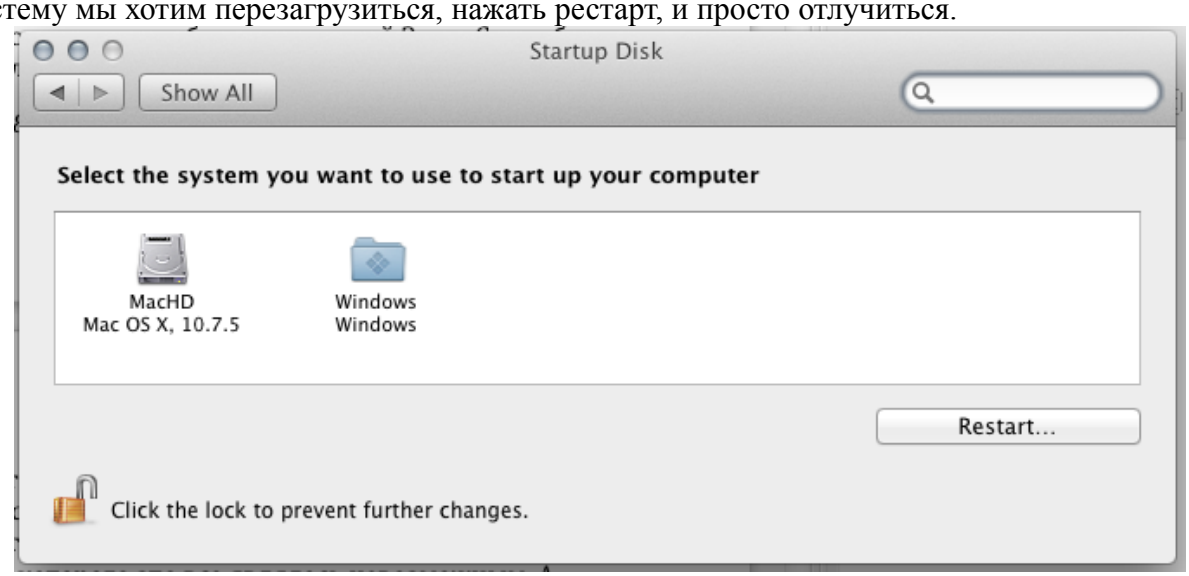
Как Кловер обеспечивает работоспособность этого сервиса?
1. Для легаси загрузки используются функции EmuVariableDxe. Это, разумеется, не настоящая энергонезависимая память, из-за того, что легаси-Кловер предназначен для тех компьютеров, где такой памяти вообще нету, как нет и собственного EFI с нужными сервисами. Этот драйвер пишет переменные просто в память, но эта память доступна для использования MacOSX в ее родном интерфейсе. При завершении работы системы вызывается скрипт rc.shutdown.local, который сохраняет всю эту память в файл nvram.plist в корне системного диска. Кловер при старте отыскивает этот файл, и заносит все переменные оттуда снова в оперативную память, эмулирующую NVRAM. Метод неполноценный, ибо таким способом сохраняются только переменные с AppleBootGuid, однако, этого достаточно для выбора Стартового Диска.
2. Для УЕФИ загрузки мы рассчитываем на собственный сервис VariableDxe, который предоставлен в OEM UEFI. В ревизии 2837 Дмазар поправил работу с этим сервисом, так что у большинства юзеров теперь оно работает по-нативному. Для тех, у кого это все-таки не работает, предусмотрен драйвер эмуляции EmuVariableUEFI, работающий аналогично легаси драйверу, и тоже требующий скриптов и файла nvram.plist. Вот и наступили новые времена! Опять-таки vit9696 поправил драйвер OsxAptioFix так, чтобы железный HBPAM работал, но вот на новых чипсетах 360, 390 и это не работает. Изменение представлено драйвером OsxAptioFix3Dxe, а сам vit9696 предлагает более продвинутый вариант AptioMemoryFix, нынче включенный в репозитарий Кловера.

EmuVariable в обоих случаях не является полноценной эмуляцией., например, не сохраняется panic.log, просто потому, что скрипт не успевает сработать. Не сохраняется

также переменная boot0082, необходимая для гибернации, но эту проблему мы обошли другими способами. А вот наличие panic.log, давняя мечта хакинтошеров, остается прерогативой Кловера с настоящим NVRAM. И, опять-таки, гибернация в моде 25 требует сохранения ключа шифрования в онлайне, то есть только с настоящим NVRAM.

iMessage — система обмена мгновенными сообщения от самой Эппл. С декабря 2012 года правила регистрации и использования сменились, и все хакинтошеры остались не у дел. С Кловером она бы работала, если бы мы в сентябре, разобравшись с сервисом iCloud, не ошиблись в количестве цифр, надо было оставить 17, а мы оставили 12. Ошибку поняли только в январе, и таким образом и хамелеоновцы поняли, в чем дело, только у них не было NVRAM, без которого это все являлось невозможным. А именно, для успешной регистрации iMessage необходимо записать в NVRAM переменные ROM и MLB, уникальные для каждого компьютера, а компьютер опознается по его HardwareUUID, который, соответственно, тоже должен быть уникальным. Для совсем чайников я сделал генерацию этих свойств на основе данных DMI, но также и рекомендации вписать соответствующие значения в config.plist, для тех, кто соображает чуть больше. При этом выяснилось, что сервис iMessage платный, и пользователю необходимо зарегистрировать свой аккаунт в аппсторе, с которого Эппл может списать 1$ для проверки, что счет в банке действующий. Из этого также следует необходимость уникальности аккаунта. Не нужно пользоваться чужими ROM, MLB и UUID, а тем более, чужой банковской картой. Когда все своё, ROM имеет 12 цифр, MLB имеет 17 цифр, UUID ненулевой, и все это уникально, аккаунт привязан к действующему счету, на котором есть деньги, iMessage будет работать. И не слушайте никаких спекуляций по поводу en0, форматирования разделов и тому подобным. Все условия я перечислил.

Загрузочный Диск — сервис, который позволяет выбрать в панели управления, в какую систему мы хотим перезагрузиться, нажать рестарт, и просто отлучиться.



Компьютер все сделает сам. Сервис этот требует, чтобы диск был размечен в GPT. Так можно переключаться между 10.9 и 10.7, к примеру.

Помните общее правило: **динамические данные имеют приоритет над статическими. Данные из NVRAM имеют приоритет над данными из config.plist.**

Использование нескольких конфигураций

Возможная проблема: у вас имеется несколько систем, но эти системы должны загружаться с разным набором патчей, прописанных в конфиге, например определение

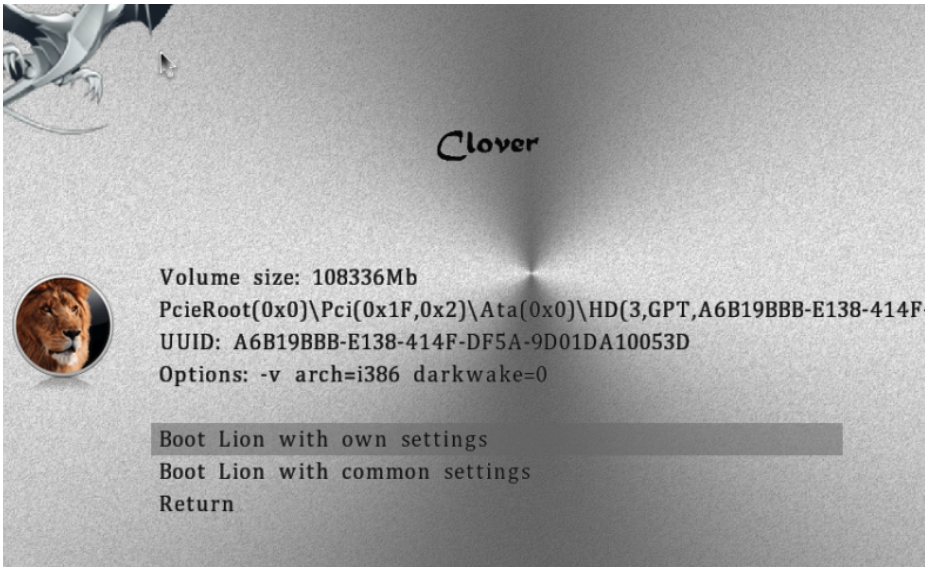
фреймбуффера Радеона в новой системе не такое, как в старой. Но как это сделать, если конфиг в кловере один? Начиная с ревизии 3266 такая возможность предусмотрена.

Вот такой конфиг

```
<key>GUI</key>
<dict>
   <key>Custom</key>
   <dict>
      <key>Entries</key>
      <array>
         <dict>
            <key>FullTitle</key>
            <string>Lion special</string>
            <key>Settings</key>
            <string>config-special</string>
            <key>Volume</key>
            <string>EE9CCC69-EE7F-358F-B120-BCD07AD78282</string>
            <key>SubEntries</key>
            <array>
               <dict>
                  <key>FullTitle</key>
                  <string>Boot Lion with own settings</string>
                  <key>CommonSettings</key>
                  <false/>
               </dict>
               <dict>
                  <key>FullTitle</key>
                  <string>Boot Lion with common settings</string>
                  <key>CommonSettings</key>
                  <true/>
               </dict>
            </array>
         </dict>
         <dict>
            <key>FullTitle</key>
            <string>Lion default</string>
            <key>Volume</key>
            <string>EE9CCC69-EE7F-358F-B120-BCD07AD78282</string>
            <key>Type</key>
            <string>OSX</string>
         </dict>
      </array>
   </dict>
```

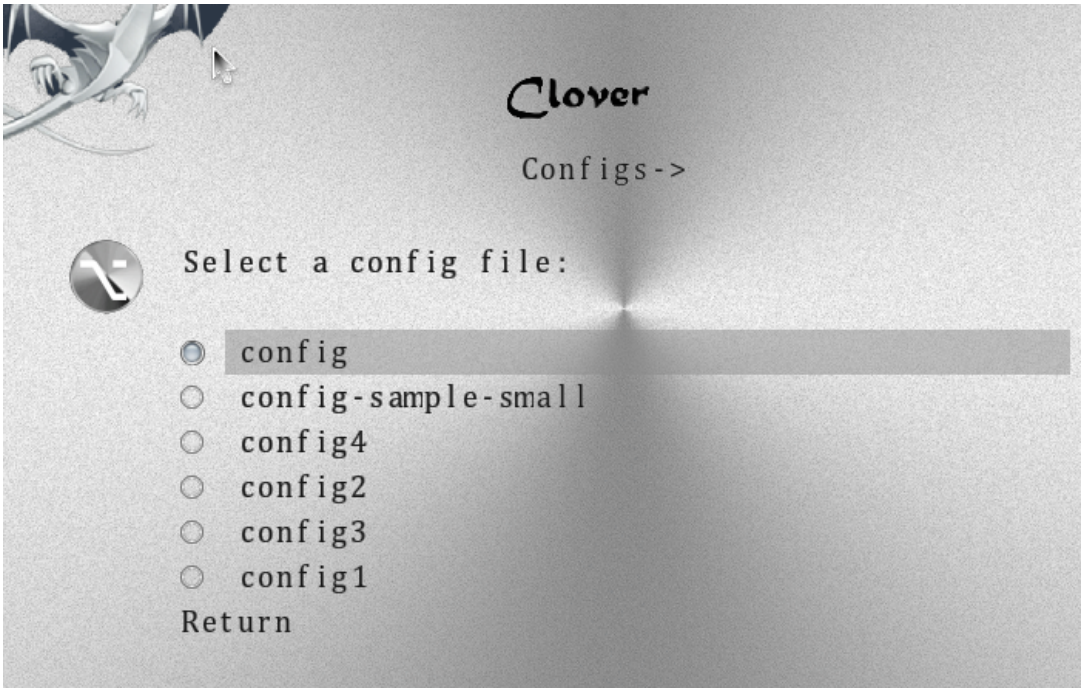
Здесь описано следующее: мы назначили свои собственные пункты главного меню (Entries) с названиями "Capitan special" и "Capitan default". Второй пункт как обычно, позволяет грузить систему с общим config.plist, с учетом изменений, сделанных в Options меню Кловера. Первый же пункт создает новую иконку для той же системы, но она будет грузиться с другим конфигом config-special.plist, как указано в ключе Settings.

Но это еще не все. Нажав пробел мы войдем в меню запуска, и тут обнаружим наши входы, прописанные как SubEntries



Сопоставьте конфиг с приведенными картинками для ясности, что происходит.
Второй пункт в этом меню означает отказ от специального конфига, чтобы использовать общий.
Понятно, что раз специальный конфиг подключается уже после того, как Кловер запущен, то разделы Boot и GUI в нем уже не нужны, они могут быть только в общем конфиге. Я, лично, использовал эту возможность чтобы тестировать новые конфиги для запуска капитана, имея один проверенный рабочий. Рабочий — special, а общий конфиг экспериментальный, потому что общий конфиг можно менять через меню, а специальный используется как есть.

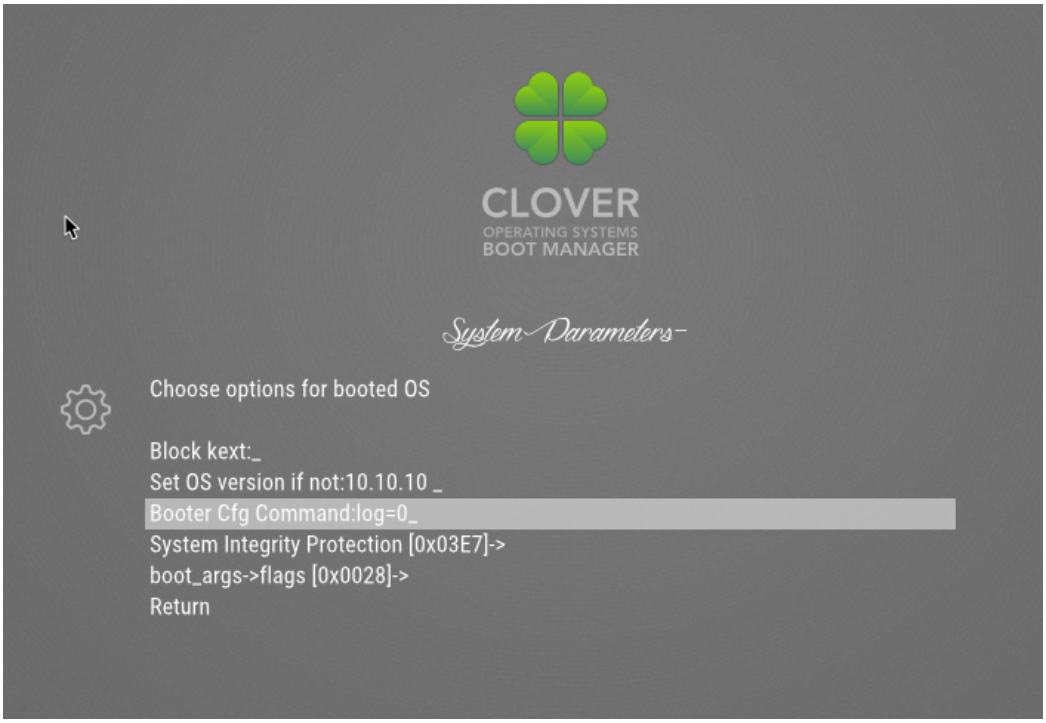
Теперь в Кловере есть возможность переключать конфиги прямо в меню. Вот картинка:



Ограничение этого метода в том, что секция Boot не меняется. Также это не меняет KernelAndKextPatches. Новый конфиг вступит в силу после выхода в основное меню.
Тут возникает недоумение, а зачем вообще менять конфиг, если не для KextPatches? Увы, не работает, причина где-то глубоко в алгоритмах Кловера. С другой стороны, пишите все патчи в один конфиг, главный, а в интерфейсе Кловера вы их можете галочкой включать или выключать. Польза от разных конфигов разве что в разных SMBIOS секциях.

Как сделать, чтобы boot.efi не слишком спаммил на экране?

Войдите в это меню



и

пропишите там log=0. Возможны другие значения, исследовал vit9696

log=value, направление вывода
1 — AppleLoggingConOutOrErrSet/AppleLoggingConOutOrErrPrint (classical ConOut or StdErr on failure)

2 —AppleLoggingStdErrSet/AppleLoggingStdErrPrint (StdErr or serial?)

4 —AppleLoggingFileSet/AppleLoggingFilePrint (BOOTER.LOG/BOOTER.OLD file on EFI partition)

debug=value
1 — enables print something to BOOTER.LOG (stripped code implies there may be a crash)

2 — enables perf logging to /efi/debug-log in the device three

4 — enables timestamp printing for styled printf calls

level=value – уровень ошибок
kc-read-size=value - размер лога
Для наших целей достаточно log=0, что в Кловере сделано по-умолчанию.

ЧаВо

Часто задаваемые **Во**просы.

В. Хочу попробовать Кловер, с чего начать?

О. С чтения этой книги.
ЗЫ. Странно писать это внутри книги, но может эти ЧаВо окажутся вне ее страниц.

В. Какая версия Кловера лучше всего подходит под мое железо?

О. Последняя. Даже не обсуждается.
ЗЫ. Вот Баг-лог по некоторым ревизиям, что, наконец, исправлено:
3514: поддержка DDR4
3471: глобальный баг с использованием va_args
3362: баг СМБИОСа когда в оригинале дублируются строки
3358: исправлено вычисление количества ядер для многих Ксеонов
3336: исправлен баг с фиксом регионов
3333: добавлены новые процессора
3259: патч кернела, позволяющий грузить кексты в ElCapitan
3168: изменение конфига в меню не воспринималось Кловером
3164: поправлен драйвер IDE, чтобы корректно работал в режиме UDMA.
3162: поправлен драйвер XHCI для выключения легаси и включения портов.
3157: увеличена скорость AHCI в разы.
3154: предотвращено зацикливание на патче InfoPlist.
3147: поправлены баги драйвера BiosBlockIO.
3144: исправлены баги с чтением конфига и установки из меню, взаимозависимости.
3138: баг с запуском Виндоус.
3128: баг с чтением SPD.
3121: запуск с раздела exFAT.
3116: ряд исправлений инсталлятора.
3100: возможность патча ядра Йоси.
3090: баг VboxHFS.efi читает не тот файл, который запрошен.
3086: баг с чтение мак-адреса, виснущего на новом чипсете.

3074: подвисание USB Legacy Support. Известен изначально, но пути решения у разных разработчиков были прямо противоположны и несовместимы.
3060: AHCI драйвер. Патч пришел от его создателей — от Интел.
3057: перекрытие таблицы адресов и EBDA, вызывающие невозможность старта на некоторых БИОСах.
3053: процедуры, использующие макросы VA_ARG должны иметь EFIAPI, иначе возможны баги в работе. И реально наблюдались! Патч от Интел.
3041: добавлена инжекция новых видеокарт.
3036: поправлен патч ядра для 10.10. Автор — Rehabman.
3035: поправлен патч ДСДТ, приводивший к зависаниям.
И так далее... Все это не считая поправок в процессы компиляции и установки, в косметику и оформление, а также поддержку новых железок и новых ОСей.

В. Не работает.

О. Сам дурак.
ЗЫ. Ну а что тут еще ответишь?

В. Установил Кловер, но получаю черный экран.

О. Загрузка ОС происходит в восемь этапов (см. Стр.6). Будьте добры, уточните, на каком именно этапе происходит остановка. И в своем отчете обязательно укажите «Устанавливал инсталлятором с выбором таких опций». Тогда и будет разговор.
Наиболее распространенные ошибки:
– с некоторыми БИОСами CsmVideoDxe не работает, удалите его;
– бывает, что PatchVBios=Yes приводит к черному экрану, попытайтесь выключить,
– стоит Boot->Debug=true. Все работает, но медленно, не хватает терпения дождаться.

Для лучшей диагностики происходящего поставьте

```
<key>Boot</key>
<dict>
   <key>Debug</key>
   <true/>
```

в файле config.plist. Загрузка будет происходит очень медленно, поскольку на каждом шаге будет обновляться /EFI/CLOVER/misc/debug.log, зато после окончательного зависания вы получите информацию, что именно произошло. Реально при загрузке с флешки речь может идти о десяти минутах до входа в ГУИ. Начиная с ревизии 3063 экран уже не черный, если КловерГУИ начал грузиться, то вы увидите надписи на экране, по которым вы поймете, что именно происходит.

В. Вижу на экране 6_ и больше ничего не происходит.

О. Это самый тяжелый случай несовместимости по железу. Сейчас уже не встречается, разве что с процессором АМД. Продиагностировать сможет только программист, который сможет вставлять в коды Кловера отладочные сообщения, и делать ребут за ребутом до полного выяснения проблемы. Увы, простым пользователям посоветовать нечего. Прочитайте главу про медленный Кловер, может подождать?
Разве что поиграться с установками БИОСа, иногда помогает. Пробуйте вместо файла boot использовать boot7 (Clover BiosBlockIO). Или сектор boot1 переустановите.

В. Происходит загрузка только до текстового аналога БИОСа с пятью пунктами, верхний – Continue>

О. Это означает, что файл boot успешно загрузился, и работает, но не находит файла CloverX64.efi. То ли того раздела не видит, то ли вообще устройства – надо разбираться далее, прогулявшись по опциям этого меню. Может, например, отсутствовать файл

HFSPlus.efi, а у вас Кловер установлен на раздел HFS+. Странно вообще-то, зачем делать UEFI загрузку с раздела HFS+.

В. Установил Кловер на флешку, загрузился с нее, и не вижу своего HDD.

О. Во-первых, HDD надо вставить в порт Sata0. В будущем может быть это ~~будет~~ уже исправлено.[5] Во-вторых я понимаю, если у вас есть хорошо работающий Хам, Химера, XPC, короче, ББХ (**Б**утер на **Б**укву **Х**), вы не хотите его убивать, но хотите попробовать Кловер, то такой поступок кажется естественным. Но, тем не менее, есть варианты установки Кловера на жесткий диск, не убивающие старого загрузчика, и в таком раскладе озвученная ошибка пропадет.
Пробуйте также файл boot7, если у вас какой-то необычный SATA/SAS/RAID контроллер. При УЕФИ-загрузке это может также означать отсутствие файлов PartitionDxe.efi и HFSPlus.efi.

В. При УЕФИ-загрузке не вижу раздела с МакОСью, только легаси.

О. Это означает, что в папке /EFI/CLOVER/drivers/UEFI отсутствует HFSPlus.efi или его легальный аналог VboxHFS.efi.

В. При УЕФИ-загрузке Виндоус выглядит как легаси, хотя он EFI.

О. То же самое, отсутствует драйвер NTFS.efi
ЗЫ. Эти два драйвера отсутствуют в репозитарии по лицензионным причинам, Вам нужно отыскать этот файл где-то на просторах сети. Сейчас существует легальный аналог GrubNTFS.efi. Имеется в инсталляторе Кловера.

В. Выставил родное разрешение в загрузчике, но экран в черной рамочке.

О. Никак не исправить. Во всяком случае разработчики Кловера ничего не смогли придумать, и на этот вопрос никто не ответит. Есть один вариант: если есть УЕФИ БИОС то надо сделать УЕФИ-загрузку, и прошить видеокарту до УЕФИ ВидеоБИОСа. В БИОСе делаем настройки:
- OS: Windows 8 WHQL
- CSM: Never
- Full screen logo: Disabled

Для легаси-загрузки сделать ничего нельзя. Не нравится траурная рамка — сделайте более низкое разрешение.

В. При попытке запуска ОСи зависает на черном экране

О. В этот момент происходит патч ДСДТ с вашей маской. Да, в идеале тут не должно виснуть. Но проблема в том, что очень много производителей БИОСов не соблюдает стандарты, не умеют программировать, и не желают отшлифовывать свой ДСДТ под нужды OSX. Очень легко убедиться, что операция декомпилировать - снова скомпилировать не проходит – ДСДТ кривой. Кловер желал бы все это исправить, но увы, количество плохих вариантов пока не поддается даже обзору. Поэтому, от вас требуется подобрать такую маску фикса ДСДТ, чтобы загрузчик не повис, а затем и чтобы ОСь не повисла, а в идеале, чтобы она еще и работала. Это – реально. Либо отказаться от автопатча (маска = 0), а ДСДТ сделать вручную. Смотрите главу про дебаг дсдт. И еще настоятельно рекомендую использовать последнюю версию Кловера, ибо такие баги время от времени отыскиваются и исправляются.
А еще есть вариант: поставьте KernelPM=true

5 Была такая ошибка в САТА драйвере от Интел, в настоящее время исправлена.

В. Ядро начинает грузиться, но паникует после десятой строки Unable To find driver for this platform \"ACPI\".

О. Это отсутствующий, или неправильный ДСДТ. Если автопатчем не получается, добавьте ДСДТ, сделанный вручную. Обратите внимание на варианты автопатча, а также на ключи ReuseFFFF и DropOEM_DSM.

В. Система начинает грузиться, но стопорится на still waiting for root device....

О. Кроме обычного для таких случаев совета включить AHCI в БИОСе, или, если такого нет, найти правильный драйвер (в смысле кекст) для вашего IDE контроллера, тут есть еще совет загрузиться с ключом WithKexts (в новых ревизиях NoCaches), тогда загрузка пойдет медленнее, и контроллер успеет включиться. Кстати, такая ошибка может возникнуть только если Кловер и система находятся на разных устройствах.

В. Система грузится до сообщения: Waiting for DSMOS....

О. Отсутствует FakeSMC. Может быть с Хамелеоном у вас этот кекст лежал в Экстре, а Кловер этой папки не видит. Для него предназначена папка /EFI/CLOVER/kexts/10.x или другие. Не забудьте также про ключ InjectKexts. По умолчанию отключен!
На второй стадии инсталляции Кловер не знает версии системы (она еще не определилась), поэтому кладите FakeSMC в папку /EFI/CLOVER/kexts/Other/
В новых версиях ключ InjectKexts имеет значение «Detect», который должен автоматически справляться с этой ситуацией, проверьте, что написано в вашем конфиге.

В. Система проходит это сообщение, но дальше ничего не меняется, хотя винчестер жужжит, как будто система грузится.

О. Типичная ситуация, когда не включилась видеокарта. Пробуйте GraphicInjector=Yes в конфиге, либо наоборот =No. Во втором варианте Радеоны запускаются на «нативной заводке», которая позволяет даже работать в системе, за небольшими исключениями, например DVDplayer не будет работать. Для полной же заводки Радеона требуется еще и коннекторы поправить. Для других случаев можно попытаться загрузить систему с ключом –x, и войти на десктоп в режиме VESA. Не очень здорово, но зато позволит что-то исправить. Еще вариант тормоза в этом месте наблюдается, если выбираете модель MacMini или MacBookPro. Проблема решается с установкой ключа DropMCFG=Yes или FixMCFG

В. Система грузится до сообщения: [Bluetooth controller....

О. Тоже самое. Смотрите предыдущий пункт. Синезуб тут ни при чем.

В. Система загрузилась, все хорошо, но в Систем Профайлере ошибки...

О. Вообще это косметика, на функциональность не влияет.
О платах PCI. См. Главу про AAPL,slot-name
О памяти. Есть две величины скорости, номинальная и фактическая, и они часто не совпадают. Какую показать в профайлере? Поставил первую – заорали, что это неверно. Поставил вторую, эти замолчали, другие пользователи заорали, что это неправильно.... Смотрите страницу 47 — как прописать свои значения памяти в конфиге.

Заключение

Кловер, конечно, еще далек до идеала, но процесс совершенствования программ никогда не бывает завершенным. Будут новые ревизии, будут новые функции, а пока так.

Самый большой недостаток Кловера в том, что он пытается быть универсальным. Программист может сделать из исходников свой вариант, подходящий именно под свое железо. Для остальных существует конфиг с сотнями настроек, и это слишком сложно для

среднего ума, несмотря на наличие автоматики, инструкций, описаний и массы советов от знатоков. Хамелеон работает за счет драйверов БИОСа, и поэтому у него больше шансов запуститься на произвольном железе, но только никто не ведет статистики, в каком проценте случаев Кловер работает правильнее.

Разработка Кловера закончена, но проект не умер, он продолжает оставаться, и еще будет развиваться.

О Хамелеоне.

Большой респект всем создателям этого проекта, который сделал возможным Мак на обычном ПиСи. Кловер позаимствовал много технологий из него, ибо создан для тех же целей (инжект видеокарт, ефи-стрингов, патч ацпи, генератор ссдт, патч смбиоса, но только все это уже на совершенно другом уровне).

Я также был среди разработчиков Хамелеона и предлагал свои патчи/улучшения, однако, админы проекта меня игнорировали. Там очень много недостатков и просто багов, которые так и не исправлены. http://www.projectosx.com/forum/index.php?showtopic=1106
Когда Хамелеон не работает, об этом не говорят, его просто игнорируют.

Первый удар пришелся на весну 2011 года, когда вышла система 10.7, и Хамелеон не смог ее загрузить. Тогда Гык обнаружил, что систему может загрузить XPC, который ЕФИ-загрузчик. Это было стартом для проекта Кловера, ЕФИ-загрузчика с открытым кодом, в отличие от приватного XPC. Причина неудачи Хамелеона была в структуре BootArgs, которая изменилась в новой системе, а также легаси прерывания. Респект netkas и cparm, которые нашли способ исправить Хамелеон, чтобы он грузил новую систему.

Второй удар произошел в январе 2013 года, когда iMessage для активации потребовал наличия переменных ROM и MLB в NVRAM. Кловер преодолел это еще в сентябре, но с небольшой ошибкой в длине строки, которая была исправлена только в январе. Тогда с Кловером заработал iMessage, а для Хамелеона это оказалось невозможным повторить. Принцип работы совсем другой. Меклорту и Космо1 потребовался месяц, чтобы преодолеть эту планку. С той зимы число пользователей Кловера впервые превысило число пользователей Хамелеона. Но Хамелеон снова полноценно работает, и остаются ярые приверженцы его. «С хамелеоном все работает!».

Третий удар хамелеоновцы проигнорировали, типа «нет и не надо». В январе 2014 года мы сделали гибернацию — глубокий сон. С хамелеоном оно работало только до системы 10.7 почему-то. Расследовать почему и как оказалось некому. Меклорт ушел от дел, остальные разработчики в команде могут только вносить новые названия видеокарт. Кловер оказался единственным загрузчиком, с которым гибернация работает хотя бы в системе 10.9.

Могу также напомнить, что с Хамелеоном не решаются проблемы плавающих регионов, имени слота, и множества не особенно нужных мелочей. Помимо этого в Хамелеоне масса ошибок, которые исправлять просто некому.

Последний удар произошел в июне 2014. Эппл выпустила систему 10.10 Yosemite, которую может загрузить Кловер, и необходимые патчи уже внесены, начиная с ревизии 2695. А вот для Хамелеона похоже наступил конец... Оглядываясь на историю, понимаешь, что зарекаться нельзя, все в этом мире возможно, возможно, что кто-то из разработчиков все же преодолеет и эту планку, а кто-то из поклонников так и останется с Хамелеоном. Счастливо оставаться!

ЗЫ: Да, разобрались и с этой проблемой, Хамелеон теперь грузит Йосю, но почему-то возникли проблемы с 10.9.4, проблемы с HBPAM, а значит и с айМесядж. И судя по активности на форуме, Хамелеон/Химеру имеют только те, кто как-то когда-то установил систему, и не собирается что-либо менять.

Еще удар, появление файловой системы apfs. Для Кловера есть родной эппловский apfs.efi, но он работает только в EFI среде, а в Хамелеоне никак. Ну и опять, спустя два года появился программер, который сделал легаси драйвер APFS для Хамелеона. Удачи!

Химера — урезанный бранч Хамелеона, со своей темой и с "другим инжектом видео". То есть для завода видеокарты нужно применять ДСДТ патч или кекст типа натита.

Ревобут — урезанный Хамелеон, в котором нужно вкомпилировать свой ДСДТ. То есть ревобут каждый должен скомпилировать под себя. По утверждению создателей (Master Chief и его "дочь" Revogirl) это позволяет сократить время загрузки на время чтения файла ДСДТ. Бред! Остальные улучшения еще более сомнительные. В настоящее время поддерживается Pike R.Alpha ("сын" шефа, брат этой девочки), который, в частности, сумел сделать загрузку Йосемита. Для себя он, разумеется может сделать, чтобы все работало (а кто-то может это подтвердить?). Но вот для других пользователей тут нечего предложить.

Да, Хамелеон имеет право на жизнь в силу того, что он является чисто легаси-загрузчиком, и может работать там, где у Кловера проблема с легаси-бутом, старые компьютеры, левый чипсет, и тому подобное. Не совсем понимаю отношения с АМД ЦПУ. У кого-то работает, а другие даже не пытаются, просто пользуются готовыми решениями с Хамелеоном. Вроде и с Кловером работает, только никто не расследует.

Короче, мне надоело говорить про Хамелеон. Я много лет доказывал, что Кловер лучше, кого-то так и не убедил, наплевать. Тема закрыта.

Другие ЕФИ-загрузчики.

Загрузчик XPC был аннонсирован в 2009 году, собралась команда и даже создали сайт проекта. Что с ними произошло я не знаю. Последнее сообщение гласит что "из-за спамеров мы не будем делать проект открытым". Какие спамеры и чем они им помешали я не понял. Проект заморозили, команда разбежалась. Остался iPhoneTom, собственно основатель, который ни на какое сотрудничество больше не шел, и исходники открывать не стал. Звездный час проекта наступил, когда весной 2011 года Гык установил 10.7 с помощью XPC, чего было невозможно с Хамелеоном, как я говорил выше. Том ожил, но сотрудничать не стал, а только позволил тестерам сообщать в ИРС свои отчеты и пожелания. У меня XPC не заработал ни на одном компьютере, поэтому я начал свой проект, это и было стартом Кловера. Итак, расклад к осени 2011 года: большинство юзеров используют Хамелеон, который преодолел эту проблему и начал бурно развиваться. Некоторые попробовали XPC, и стали его ярыми приверженцами: "Чем заниматься херней, ты бы лучше помог Тому с его загрузчиком. Он вполне адекватный парень, и слушает критику". Я, однако, программист, я могу работать сам, а не сидеть у ИРКи в ожидании, когда добрый дядя что-то исправит. И я, пока в одиночестве, стал делать загрузчик на основе ДУЕТа, и в первый же месяц получил некоторые результаты лучше, чем XPC. Война так война, я не отдал свое ноу-хау Тому. И маленькое изначальное преимущество — поддержка русских юзеров, которых больше, чем любых других.

В Кловере версии 1 был использован интерфейс от Нинзи, который "украл" его от ранней версии XPC. В такой ситуации развивать Кловер было невозможно, и в начале 2012 года, когда я понял все необходимые технологии, я начал делать интерфейс Кловера версии 2 на основе проекта rEFIt, с открытыми исходниками. Хочу заметить, что и XPC происходит из него, так что претензии скорее к нему, какое Том имеет право закрывать исходники, если сам пользуется открытыми. Теперь Кловер стал лицензионно чист, и поднялся до уровня, когда можно было говорить о конкуренции. Весна 2012 года. "XPC пока переплюнуть по функциональности никому не удалось". Однако, у него оставалась нерешенной проблема с system-type, которая в случае ноутбука мешала сну. А также board-id, которая мешала на некоторых конфигурациях с установкой системы 10.7+. А на Кловере у меня этих

проблем не было, потому что я изначально выбрал другие патчи, по другим идеям, и что из них так повлияло было совершенно неочевидно, глядя в мои исходники. Я-то знал, но твердо решил никому и никак не объяснять. Юзерам это ни к чему. Работает в Кловере, значит будете пользоваться Кловером.

Так возник проект bareBoot. Автор SunKi, ярый приверженец XPC и лучший помощник Тома в его проекте, решил таки докопаться до истины. Он неоднократно интересовался Кловером, почему и как сделано, но никогда не вносил своих предложений по улучшению Кловеру, по его дальнейшему продвижению. Поняв, что я не собираюсь рассказывать свои секреты, он открыл свой проект, я, мол, хочу объединить файлы CloverEFI+патчи в один файл, а в качестве ГУИ использовать существующий SetupBrowser, с модификациями для загрузки нескольких систем, так что получилось текстовое меню, в котором можно выбрать систему для загрузки. Согласен, была проделана работа, и не маленькая. Однако, к этому времени уже Дмазар сделал УЕФИ-загрузку, и объединение CloverEFI+GUI оказалось неприемлимым. Баребут рассчитан исключительно на легаси загрузку. Однако у Санки не было цели сделать привлекательный загрузчик, его целью была расшифровка технологий Кловера. Он начал с чистого Дуета, и стал добавлять патчи из Кловера шаг за шагом, проверяя, что на что влияет (а ведь мог начать и с готового Кловера!). Но и Кловер не стоит на месте. Мы, уже с Дмазаром, стремительно улучшали и преобразовывали коды, так что уследить за нами было непросто, как и непросто сравнить, что было и что стало. И Санки никак не мог найти, как же в Кловере сделан system-type. Тем временем Том прекратил заниматься проектом, а в баребуте не нашлось козырей, чтобы привлечь юзеров. Отсутствие графики? Хорошо, мы и в Кловере сделаем чисто текстовый интерфейс, если у кого аллергия на графику. Скорость загрузки? Давайте посоревнуемся. А тем временем в Кловере появляются новые функции, которые не так просто скопировать в баребут, в частности патчи ДСДТ, кекстов и ядра, не говоря уже про УЕФИ-загрузку. Пользователям осталось пожать плечами "А зачем баребут вообще нужен?".

Тем временем в мире хакинтоша произошло еще одно заметное событие. Некая компания QUO смастерила материнскую плату на основе Gigabyte Z77, внеся туда изменения для лучшей совместимости с Хакинтошем. Но главное, они предложили зашивать загрузчик Мака прямо в БИОС. Один из основателей этого загрузчика, THeKiNG постоянно присутствовал в теме Кловера, и старательно расспрашивал, что и как, но также ничего от себя в Кловер не вносил. И вот мы видим некий загрузчик Ozmosis, который прошивается в БИОС, и содержит модули взятые из Кловера. Прошит туда в БИОС и какой-то урезанный вариант FakeSMC. И таким образом, на этой материнке можно запустить чистую OSX, без единого хакерского файла, ни загрузчиков, ни лишних кекстов. Правда, на мой взгляд, все это справедливо, только если ничего не обновлять. Если обновлять систему, то придется и БИОС перешивать, и вообще можно дойти до кирпича. Насчет обновления фейка и сенсоров тоже огромный вопрос. Ну и, разумеется, этот загрузчик не рассчитан на другие материнские платы.

Недавно Кинг обронил и еще одну фразу "оз неприемлем для ноутбуков". А я догадываюсь, что дело не только в том, что есть опасность получить кирпич с перепрожиганием БИОСа. Реально Оз нивелировался именно для платы Гигабайт Z77, и работа на другом железе под вопросом. Счастливого плавания!

С выходом системы ElCapitan , пока еще бета, загрузчик Озмосис также испытал потрясение. В этой системе кексты извне, например из БИОСа не грузятся. Для Кловера мы эту проблему решили (спасибо solstice), но это в самом теле Кловера. А тело Озмосиса править некому, исходники закрыты. Оставайтесь, товарищи, со старыми системами!