Лист благодарностей

В первую очередь спасибо ресурсу applelife.ru. Много полезного. Спасибо за труды всех участников ресурса.

Спасибо **Slice** за помощь, консультацию и за просто то что первый отозвался и вдохновил на работу.

Спасибо **Aleox** и **Slice** за предоставленую информацию о именах устройств в Apple и в PC.

Да и просто спасибо всем, так как если бы не вы, то я бы просто не писал бы эту книгу.

ЧАСТЬ 1
С чего все начинается?

В этой книге я буду описывать способы как же все таки “допилить” всеми нами любимый Mac OS X (Hackintosh).

Чаще всего когда кто либо пытается установить Mac OS на PC, то он сталкивается с рядом проблем, такие как не правильная работа какого либо из устройств или вообще не определение устройства. Часть из этих проблем мы решим в этой книге.

Начнем с DSDT. Что же это все же такое за зверь?

DSDT (Different System Description Table) - Таблица Дифференцирования Описания Системы, т.е. это таблицы, в которых описывается как наша система и наши устройства должны работать и где их найти.

С откуда же береться этот DSDT?

В каждом компьютере этот DSDT храниться в BIOS системы, и так система понимает с чем она имеет дело. Но Mac OS X не знает что такое BIOS и поэтому просто не может получить к нему доступ и узнать, что же у нас есть в системе. Поэтому мы должны объяснить Mac OS X, и рассказать ей, что же в нашей системе есть и как же оно будет работать.

Так как у нас установлен Хакинтош, то скорее всего у нас есть загрузчик Chameleon 2 или ему подобный, который может прочитать DSDT. DSDT обычно храниться на системном диске(там где установлена Mac OS X) в папке Extra и имеет имя файла dsdt.aml.

Но этот файл нужно туда положить. Так как DSDT одного компьютера в той или иной мере отличается от DSDT другого компьютера. То есть

Ни в коем случае нельзя использовать DSDT другого компьютера у себя, так как у вас или не загрузиться Mac OS X, или же не определяться устройства, или же есть шанс повредить компьютеру.

Раз у нас нет файла dsdt.aml. Эх неужели мы не сможем улучшить нашу любимую Mac OS X? Не стоит так быстро опечаливаться. Мы всегда сможем сделать этот загадочный файл, в котором множество интересностей и он творит чудеса.

Что же нам надо для получения этого файла?

Можно взять чей либо файл и переписать его под себя. Но это долго и далеко не факт что это все будет потом работать. Так как писать очень много. Чтобы облегчить эту задачу добрые люди сделали все за нас. Но все же кое что надо сделать.

Для начала нам нужна какая либо другая операционная система, в которой мы сделаем этот файл. Достаточно загрузится с LiveCD.

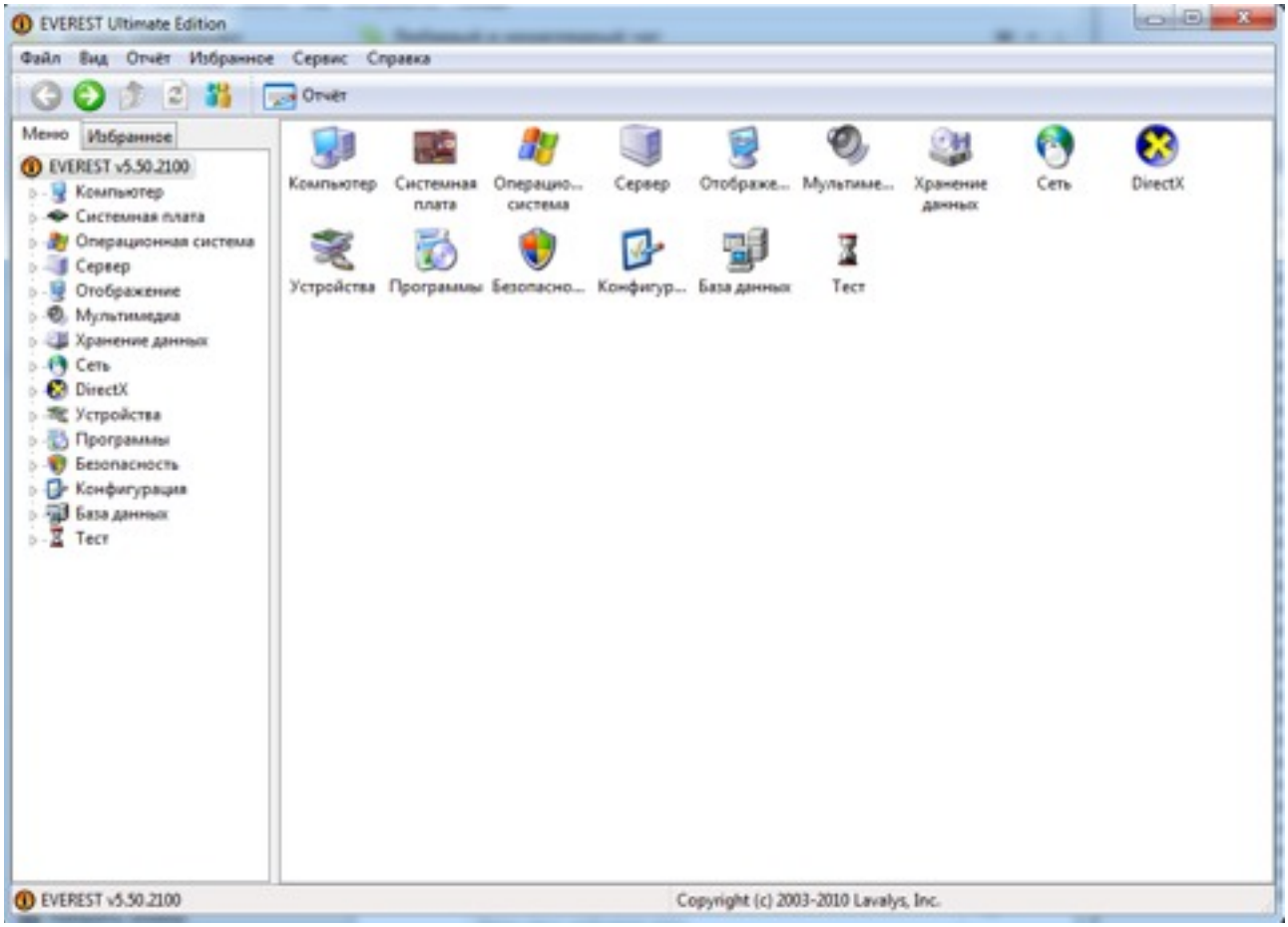
LiveCD - CD или USB, на которой уже есть операционная система и с нее можно загрузиться без установки.

Исторически так сложилось, что в наше время самые популярные операционные системы стали Windows и Linux.

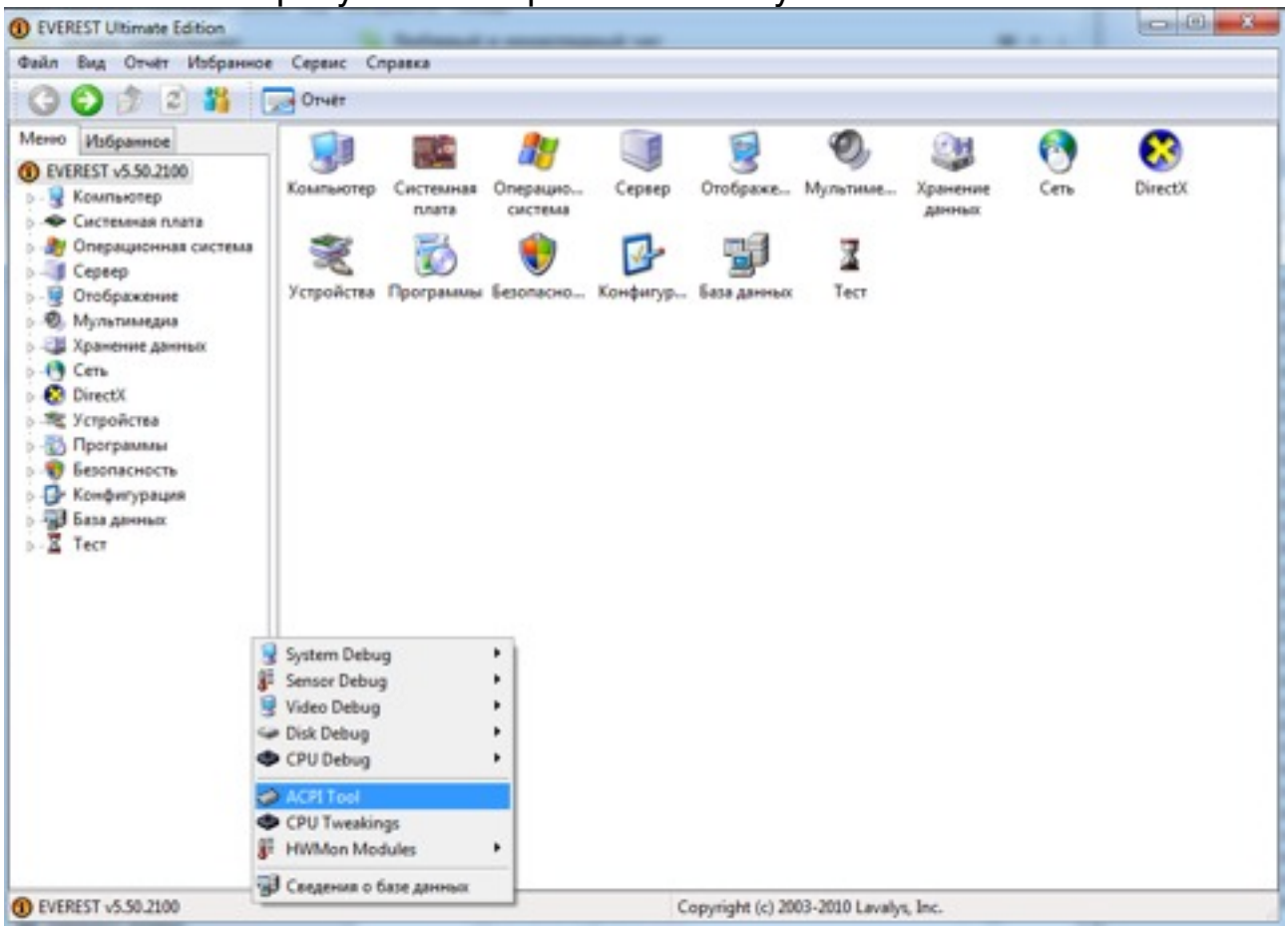
Получаем DSDT в Windows

Для этого нам необходим некоторый инструментарий. С Windows проще. Необходим всего лишь Lavalys Everest Ultimate.

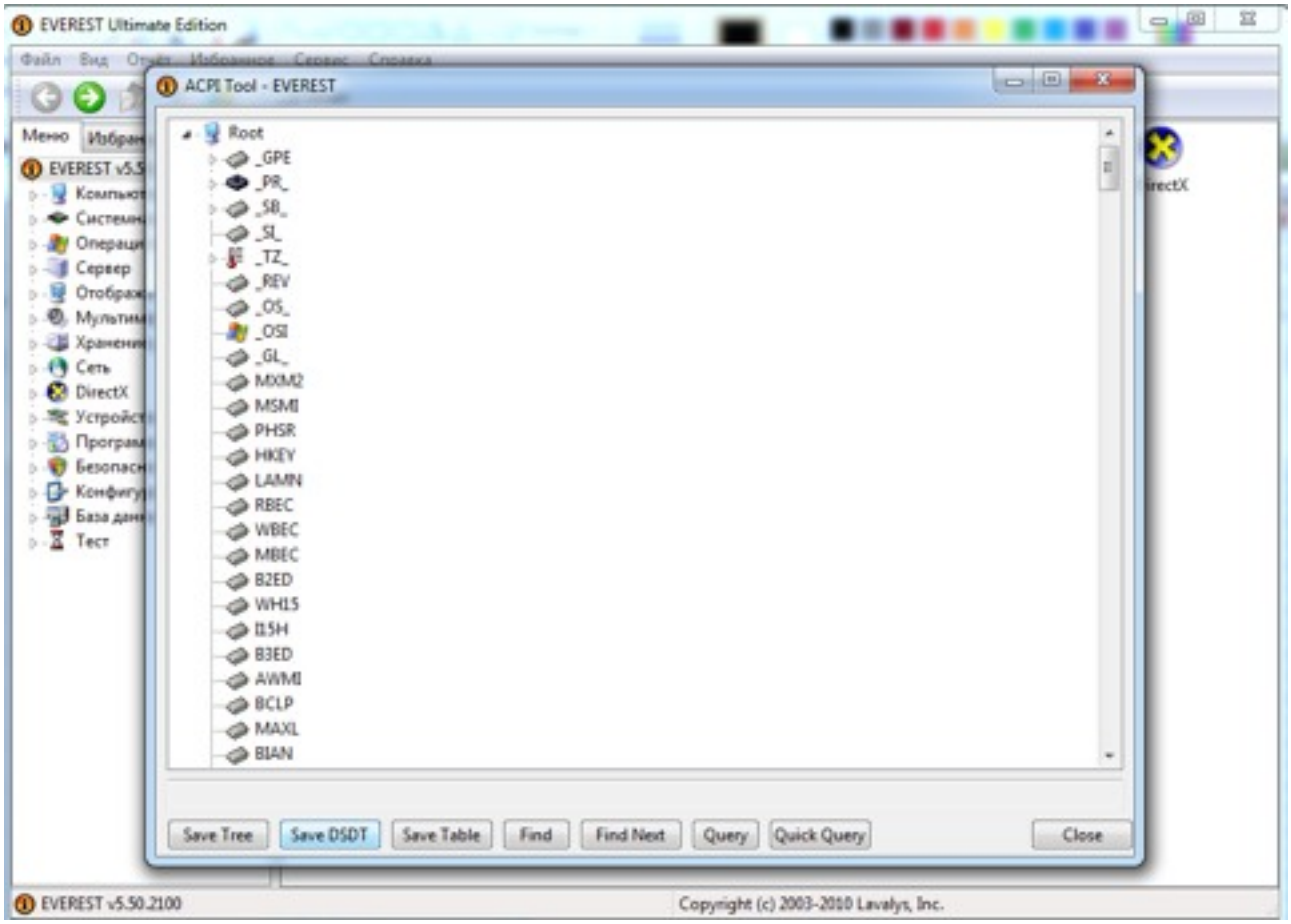
1. Запускаем нашу операционную систему Windows.
2. Устанавливаем Lavalys Everest Ultimate и запускаем его. Перед нами появляется вот такое окно:



3. Необходимо нажать правой кнопкой на строке состояния. Именно внизу, возможно левее от середины экрана. И должно появиться меню как на рисунке и выбрать в меню пункт ACPI Tool.

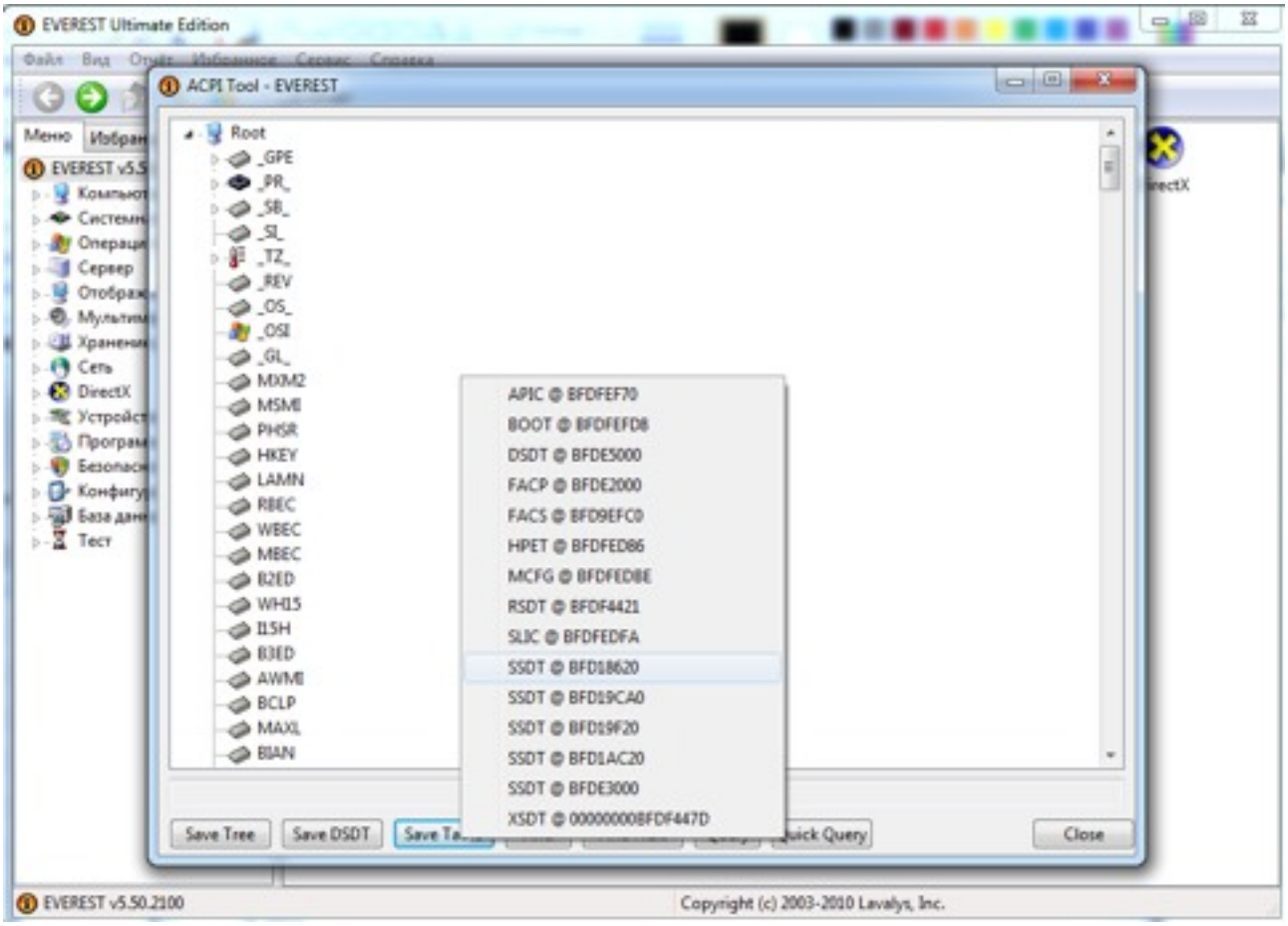


4. Отобразить окно такое как на рисунке.



5. Нажимаем кнопку Save DSDT. И указываем куда сохранять этот файл. Дайте ему имя dsdt. После нажатия на кнопку “Save DSDT” у нас появиться файл dsdt.bin.
6. Переименовываем этот файл в dsdt.aml и вот это наш dsdt.aml, который мы хотели получить ранее. Сохраняем его в удобнодоступном месте или на флешку.

Также можно сохранить все таблицы, которые есть по нажатию на кнопку Save Table.



Также сохраните как и сказано выше с понятными именами и с расширением aml. То есть будут файлы ssdt.aml и так далее.

Получаем DSDT в Mac OS X

Для этого нам необходим DSDTPatcherGUI.

1. Запускаем DSDTPatcherGUI. Появляеться окно представленое ниже:

2. Выбираем Darwin/Mac OS X. И нажимаем Run DSDT Patcher.
3. Вводим пароль и у нас скорее всего высветиться ошибка отображенная на рисунке:

4. Но пишет что лог сохранен, в моем случае, в папке Session/2010-11-30-211928. Это значит что идем в папку, с которой запускали наш DSDTPatcherGUI и там есть папка Session. В ней ищем папку 2010-11-30-211928. В этой папке лежит лог ошибок и папка Debug, в которой лежит наш DSDT в не скомпиленом виде. В принципе он нам такой и нужен для редактирования.

Сохраняем это все на флешку или в удобнодоступное место.

Получили DSDT. Что дальше?

Вот так мы получили DSDT, но это еще не все. Перед редактированием или каким либо изменением нужно исправить в нем все ошибки и скомпилировать. Для этого нам понадобиться программа DSDT SE. Это редактор для редактирования DSDT файлов, а также компилирование их.

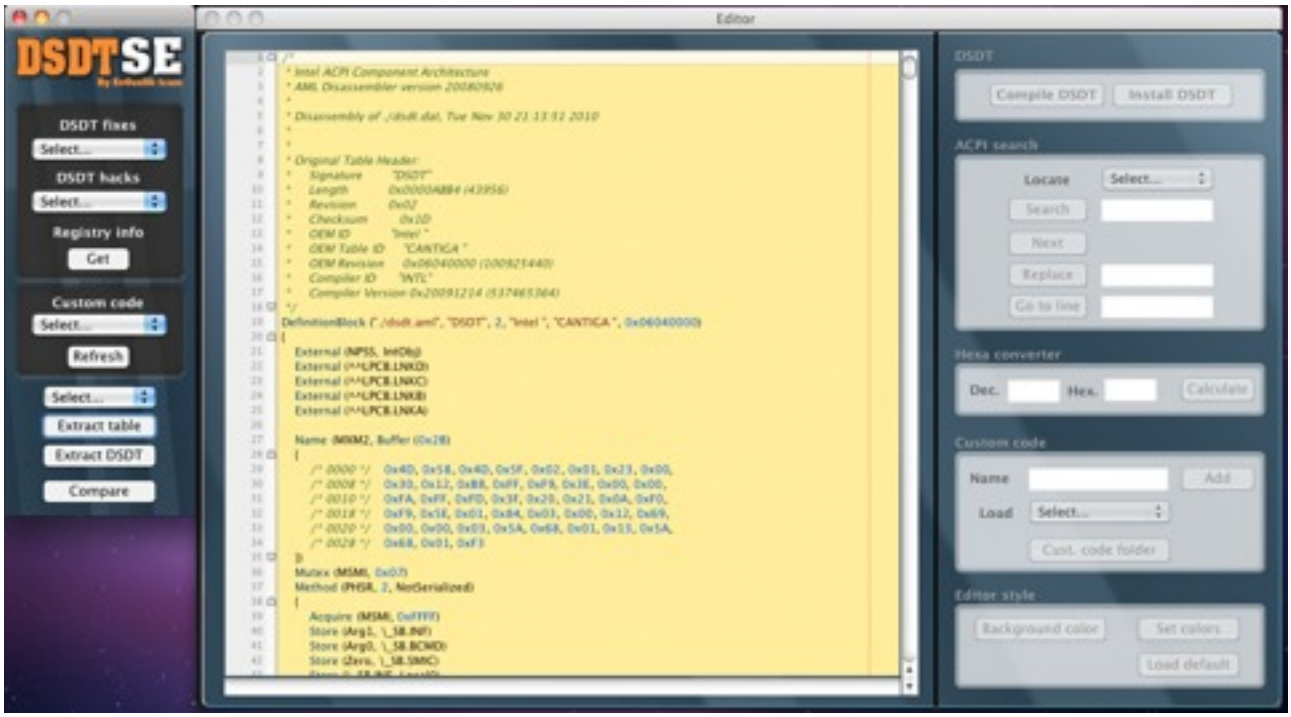
Начнем. Запускаем наш DSDT SE и открываем ранее полученный dsdt файл. будь то файл aml или dsl. Все равно. Этот редактор откроет и тот и другой формат.

Перед нами открывается окно редактора

Чтобы открыть файл, нужно или войти в меню File-> Open .dsl или File-> Open .aml. Или же перетащить наш файл полученый ранее с dsdt на иконку запущенного приложения.

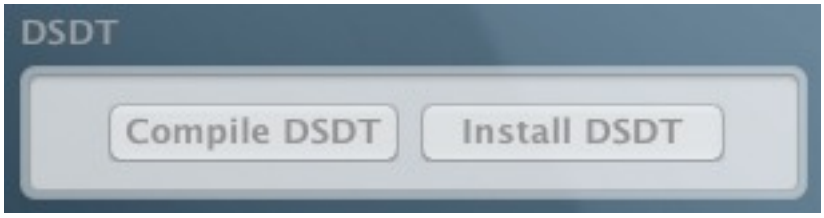
После того как вы открыли файл, сразу не пугайтесь. По началу будет страшно. Особенно если вы никогда не программировали, но

потом будет привычно и все просто. Давайте посмотрим интерфейс приложения.



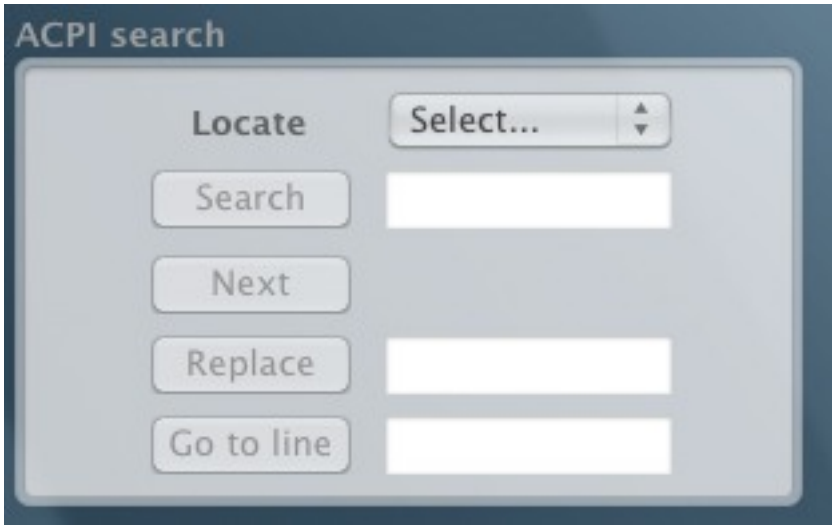
Слева главное окно нашего DSDT SE, которое вы видели ранее. справа окно редактора, в котором открытый нами файл. Вот в нем мы и будем работать основное время. Рассмотрим подробнее.

На рисунке ниже изображены кнопки Compile DSDT и Install DSDT



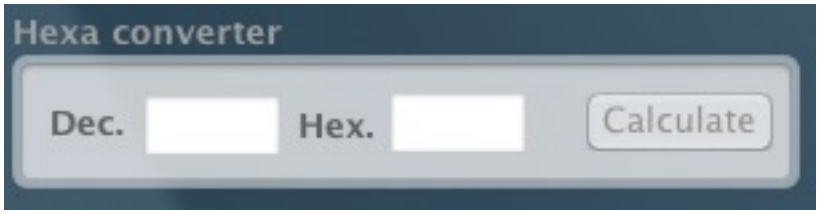
Зачем же они нужны. Дело в том что вот этот текст, который мы редактируем, нужно перевести в такой вид чтобы наша Mac OS поняла его и внесла изменения. Для этого необходимо сделать компиляцию. Естественно это делается с помощью кнопки Compile DSDT. Чтобы потом установить наш полученый файл aml, необходимо нажать Install DSDT, выбрать раздел и поставить галочку на /Extra и нажать Install. Также ввести пароль. Это еще рано делать.

Дальше следущая часть интерфейса служит для поиска нужной строки или слова



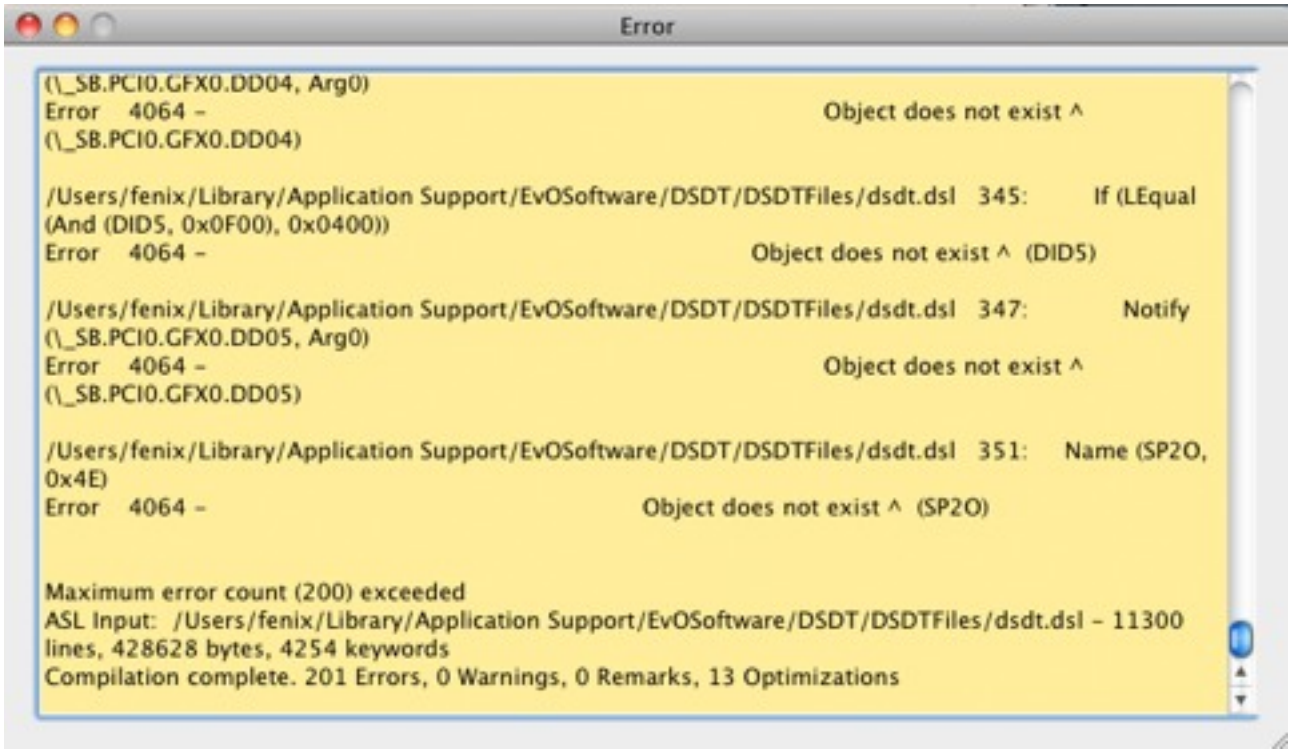
Поле Locate используеться для быстрого перехода в большом файле по секциям. Каждая секция отвечает за свою функциональность и свою часть устройств.
Поле Search необходимо для поиска какого либо слова. Дело в том что если ввести слово и нажать Search, то он найдет только первое его совпадение. Если нажать на Next, то последующие совпадения.
Поле Replace соответственно для замены слова, которое мы ищем на слово, которое введем в поле Replace.
Поле Go to line используеться для быстрого перехода на нужную строку. Очень удобная функция. Думаю оцените по достоинству.

Далее идет конвертор из 10-ричной системы в 16-ричную систему счисления.



Это основной функционал.

Теперь приведем наш dsdt файл в порядок. Нажимаем на кнопку Compile DSDT. У нас отображаеться окно с ошибками в нашем dsdt.



И в самом низу этого списка ошибок показана общая статистика. Сколько ошибок. Сколько предупреждений и так далее. В моем случае 201 ошибка.

Чтобы продолжить работу с dsdt и вносить какие либо изменения по улучшению функциональности устройств необходимо сначала устранить все эти ошибки.

Ищем место где у нас происходит ошибка. Рассмотрим одну из моих ошибок, которые вывело мне это окошко.

Например:

```
/Users/fenix/Library/Application Support/EvOSoftware/DSDT/DSDTFiles/dsdt.dsl 36: Mutex (MSMI,
0x07)
Error 4064 - Object does not exist ^ (MSMI)
```

С виду “что это за куча букв?”, но на самом деле это нам очень может помочь. Смотрим внимательнее и разбираемся. Первое. Вот эта строка:

```
/Users/fenix/Library/Application Support/EvOSoftware/DSDT/DSDTFiles/dsdt.dsl
```

Это путь где лежит наш файл и мы его редактируем.

Второе, очень важное место. У меня это “36” - это строка, в которой у меня ошибка. То есть, вводим это число 36 в поле “Go to line” и нажимаем на кнопку и попадаем на строчку где у нас ошибка.

Третье,

```
Object does not exist ^ (MSMI)
```

Это ошибка. В моем случае просто в начале этого файла необходимо дописать “ External(MSMI) ” без кавычек. И получилось в начале файла вот что.

```
13  *    OEM ID          "Intel "
14  *    OEM Table ID    "CANTIGA "
15  *    OEM Revision    0x06040000 (100925440)
16  *    Compiler ID     "INTL"
17  *    Compiler Version 0x20091214 (537465364)
18  */
19  DefinitionBlock ("./dsdt.aml", "DSDT", 2, "Intel ", "CANTIGA ", 0x06040000)
20  {
21      External(MXM2)
22      External(MSMI)
23      External (NPSS, IntObj)
24      External (^^LPCB.LNKD)
25      External (^^LPCB.LNKC)
26      External (^^LPCB.LNKB)
27      External (^^LPCB.LNKA)
```

Выделенным это то что я добавил в файл.

Исправляем ошибку и компилируем по новой. У нас на одну ошибку меньше. Так нужно исбавить от всех ошибок и желательно от warning'ов.

Дальше идут типичные ошибки и способы их устранения.

Типичные ошибки в dsdt

1 ошибка. ее вы уже видели. Это когда выдаеться сообщение
`Object does not exist ^ (MSMI)`
Для ее устранения необходимо в самом верху написать
`External(MSMI)`

2 ошибка

Если вам выдает вот такую ошибку

```
External (^^LPCB.LNKD)
Error  4014 -                                   From ACPI CA Subsystem ^ (AE_NOT_FOUND
Failure from namespace lookup)
```

То вам необходимо заменить
`External (^^LPCB.LNKD)`
на
`External (\_SB_.PCI0.LPCB.LNKD)`

3 ошибка

Код ошибки:
2617: Name (_T_0, Zero)
Remark 5111 - Use of compiler reserved name ^ (_T_0)
Это значит что вы используетет уже зарезервированое имя. То есть достаточно убрать подчеркивание спереди и все. Пример ниже.
Исходный код:

```
Name (_T_0, Zero)
Store (Local0, _T_0)
```

If (LEqual (_T_0, Zero))

Исправленный код:
Name (T_0, Zero)
Store (Local0, T_0)
If (LEqual (T_0, Zero))

4 ошибка

Код ошибки:

```
8975: Method (VGET, 1, NotSerialized)
Warning 1088 - Not all control paths return a value ^ (VGET)
```

Это значит что ваш метод должен возвращать значение, а не возвращает. Достаточно в конце метода написать Return (0x00) и ошибка исправлена. Пример ниже

Исходный код:

```
Method (VGET, 1, NotSerialized)
{
If (LEqual (Arg0, 0x03))
{
Return (^^SIOR.HWV1 ())
}
}
```

Исправленный код:

```
Method (VGET, 1, NotSerialized)
{
If (LEqual (Arg0, 0x03))
{
Return (^^SIOR.HWV1 ())
}
Return (0x00)
}
```

По окончанию редактирования и исправления ошибок вы получите чистый dsdt.aml файл, который можно будет установить в систему для дальнейшего использования и редактирования. этот файл что у вас получился, сохраните для будущего редактирования и сделайте копию.

Получили файл DSDT. Что же дальше?

И так. Мы получили файл dsdt, дальше нужно сделать основные исправления в файле, такие как исправление секций RTC, TMR, HPET, "Darwin".

Начнем?

Исправление секции RTC

И так. Можно считать первый наш опыт по улучшению и исправлению DSDT. Дело в том что в Mac OS X Snow Leopard после каждой перезагрузки компьютера сбрасывался биос в значение по умолчанию. Естественно это нам не надо. Особенно если вы его настраивали под себя очень долго и нудно. Для этого нам необходимо исправить секцию RTC.

Как же это делается. Открываем наш DSDT SE. В нем открываем файл полученый ранее dsdt.aml или dsdt.dsl. Лучше чтобы у вас был dsdt.dsl и его редактировать всегда.

Как и говорилось ранее в описании работы с DSDT SE справа есть поле Location с выпадающим списком. Там необходимо выбрать RTC. И редактор быстро переместит нас в нужную секцию. В моем случае эта секция выглядит так:

```
Device (RTC)
 {
    Name (_HID, EisaId ("PNP0B00"))
    Name (_CRS, ResourceTemplate ()
    {
       IO (Decode16,
          0x0070,             // Range Minimum
          0x0070,             // Range Maximum
          0x00,               // Alignment
          0x04,               // Length
          )
    })
 }
```

Можно также эту секцию найти введя в поиск(поле Search) фразу RTC. и нажать искать.

Так вот. Тут загвоздка в том что на разных материнских платах это разный блок. Например на материнский платах Acer, ASUS обычно код как у меня:

```
        Device (RTC)
        {
            Name (_HID, EisaId ("PNP0B00"))
            Name (_CRS, ResourceTemplate ()
            {
                IO (Decode16,
                    0x0070,             // Range Minimum
                    0x0070,             // Range Maximum
                    0x00,               // Alignment
                    0x04,               // Length
                    )
            })

            IRQNoFlags ()
                {11}
        }
```

После того как вы нашли этот код, не нужно его приводить в точно такой же вид. Не нужно удалять то, назначение чего вы не знаете. В нашем случае достаточно исправить только поля где указано справа в комментариях Length.

Смотрим что получается:

Мой случай (как в варианте 1):

```
        Device (RTC)
        {
            Name (_HID, EisaId ("PNP0B00"))
            Name (_CRS, ResourceTemplate ()
            {
                IO (Decode16,
                    0x0070,             // Range Minimum
                    0x0070,             // Range Maximum
                    0x00,               // Alignment
                    0x02,               // Length
                    )
            })
        }
```

На других материнских картах бывает вот такой код:

```
        Device (RTC0)
      {
        Name (_HID, EisaId ("PNP0B00"))
        Name (BUF2, ResourceTemplate ()
        {
            IO (Decode16,
                0x0070,             // Range Minimum
                0x0070,             // Range Maximum
                0x00,               // Alignment
                0x02,               // Length
                )
        })
        Name (BUF3, ResourceTemplate ()
        {
            IO (Decode16,
                0x0070,             // Range Minimum
```

```
                0x0070,             // Range Maximum
                0x00,               // Alignment
                0x02,               // Length
                )
            IRQ (Edge, ActiveHigh, Exclusive, )
                {8}
        })
        Method (_CRS, 0, NotSerialized)
        {
            If (And (^^LPC0.HPTF, 0x04))
            {
                Return (BUF2)
            }
            Else
            {
                Return (BUF3)
            }
        }
```

Давайте проанализируем код и посмотрим что тут делается. Есть две переменные или два буффера, просто два контейнера где записаны какие то данные. Пока не стоит задумываться о терминологии. Вот эти два контейнера: `Name (BUF2, ResourceTemplate ()` и `Name (BUF3, ResourceTemplate ()`. И есть метод, который должен выполнить какие то действия `Method (_CRS, 0, NotSerialized)`. Посмотрите на мой более простой пример. У меня также есть этот метод. Правда выглядит немножко иначе. Теперь делаем почти теже действия. Тут нам не нужно прерывания,т.е. строчка `IRQ (Edge, ActiveHigh, Exclusive, ) {8}` не нужна. Берем и удаляем ее. Смотрим метод, который должен выполниться. У него есть какое то условие и если оно выполниться, то нам вернеться BUF2 иначе если не выполниться, то вернеться BUF3. Опять таки BUF2 и BUF3 это наши контейнеры, о которых я говорил выше. А теперь давайте посмотрим и найдем 10 отличий. В принципе их нет. то есть вместо вот этого условия можно смело вписывать один из буферов и этот метод станет такого же вида как и мой метод. И получится или:

```
        Device (RTC0)
    {
        Name (_HID, EisaId ("PNP0B00"))
        Name (BUF2, ResourceTemplate ()
        {
            IO (Decode16,
                0x0070,             // Range Minimum
                0x0070,             // Range Maximum
                0x00,               // Alignment
                0x02,               // Length
                )
        })
        Name (BUF3, ResourceTemplate ()
        {
            IO (Decode16,
                0x0070,             // Range Minimum
                0x0070,             // Range Maximum
                0x00,               // Alignment
```

```
                0x02,               // Length
                )
            })
            Method (_CRS, 0, NotSerialized)
            {
                If (And (^^LPC0.HPTF, 0x04))
                {
                    Return (BUF2)
                }
                Else
                {
                    Return (BUF3)
                }
            }
```

Или же более сокращенный вариант убрав условие.

```
Device (RTC0)
        {
            Name (_HID, EisaId ("PNP0B00"))
            Name (BUF2, ResourceTemplate ()
            {
                IO (Decode16,
                    0x0070,             // Range Minimum
                    0x0070,             // Range Maximum
                    0x00,               // Alignment
                    0x02,               // Length
                    )
            })
            Method (_CRS, 0, NotSerialized)
            {
                    Return (BUF2)
            }
```

То есть, все что изменили это парамметр Length в данной секции изменили с 0x04 на 0x02.

Также, если у вас есть такие строки `IRQNoFlags () {11}` то их необходимо удалить. Также смотрите мой пример выше. Я его удалил. Вместо 11 может стоять любое число. Все равно удаляем их.

И в качестве косметики, а может и не только, это уже спорный вопрос, переименовываем ваш RTC0 в RTC, благодаря информации Aleox. Дело в том что желательно все абривиатуры устройств использовать такие какие используются в DSDT оригинальных “Маках”.

Компилируем. Записываем этот файл в папку Extra с каким нибудь именем не dsdt.aml. А с каким то другим. И запоминаем его. Перезагружаемся и в биосе че нибудь ставим не такое как по умолчанию. Начинаем загружаться и в загрузчике нужно ввести флаг загрузки

dsdt=/Extra/file_name.aml

вместо file_name необходимо написать свое имя файла и жмем Enter.

После загрузки системы снова перезагружаемся. Заходим в биос и все настройки должны остаться в том же состоянии как вы их оставили в прошлый раз.

Причем если у вас в системе стоял кекст для изменения этого явления (сброса биоса), то его уже можно удалить. У меня это был кекст ElliottForceLegacyRTC.kext.

Если операционная система не загрузилась, то загружайтесь так как обычно. без флага dsdt если вы его не вводили ранее.

Исправление секции TMR

Как вы убедились выше, нет ничего сложного, особенно если следовать инструкциям. Это может делать почти каждый и приносить плоды и улучшать работу своего чудесного и любимого PC. Но вот пришло время сделать еще одну интересность, которая улучшит работу.

Открываем наш dsdt.dsl отредактированый ранее и ищем секцию TMR. Выбираем в Location справа эту секцию или же если не нашло ее, то вводим в поле Search TMR и нажимаем Search.

Но тут также может быть подвох. Опять такие эту секцию может не найти. Не опечаливаемся ищем еще раз, но теперь со словами TIMR. На разных материнский платах по разному звучит эта секция.

И вот она долгожданная и нарядная к нам пришла. Берем и снимаем с нее все лишнее. Как это делается? Вот моя секция:

```
Device (TIMR)
            {
                Name (_HID, EisaId ("PNP0100"))
                Name (_CRS, ResourceTemplate ()
                {
                    IO (Decode16,
                        0x0040,             // Range Minimum
                        0x0040,             // Range Maximum
                        0x01,               // Alignment
                        0x04,               // Length
                        )
                    IO (Decode16,
                        0x0050,             // Range Minimum
                        0x0050,             // Range Maximum
                        0x10,               // Alignment
                        0x04,               // Length
                        )
                        IRQNoFlags ()
                        {2}
                })
            }
        }
```

Как видите она у меня называется TIMR, а не TMR. Но опять таки это одно и тоже. Смотрим в нее внимательно. В этой секции прерываний не должно быть. То есть вот эти строчки `IRQNoFlags () {2}` просто напросто нужно удалить. Больше ничего делать не надо. И получаем вот такую секцию:

```
Device (TIMR)
        {
            Name (_HID, EisaId ("PNP0100"))
            Name (_CRS, ResourceTemplate ()
            {
                IO (Decode16,
                    0x0040,             // Range Minimum
                    0x0040,             // Range Maximum
                    0x01,               // Alignment
                    0x04,               // Length
                    )
                IO (Decode16,
                    0x0050,             // Range Minimum
                    0x0050,             // Range Maximum
                    0x10,               // Alignment
                    0x04,               // Length
                    )
            })
        }
    }
```

Так же, благодаря информации Aleox, необходимо переименовать TMR, в TIMR. Иногда это имеет значение. Правда также спорный вопрос.

Вот мы и сняли все лишнее и получилось чистенькая секция. Сохраняем, компилируем и устанавливаем в папку Extra. Пробуем загрузиться. Опять таки, не давайте сразу имя dsdt.aml, иначе могут быть проблемы. Дайте другое имя и потом когда будете загружаться в загрузчике нужно ввести DSDT=/Extra/file_name.aml.

Исправление секции PIC

И снова совершенствуем и чистим от не нужного наш DSDT файл. Открываем сохраненный ранее dsdt.dsl файл и начинаем поиски в дебрях непонятного кода. Со мной вы его начнете немного понимать, а может и хорошо понимать. Кто знает?

И так открыли, ищем секцию PIC. Опять таки или выбираем в выпадающем списке Location секцию PIC. Или вводим в поиск (Search) фразу PIC и ищем.

Эта секция может называться также IPIC. Это одно и тоже.

Приведу часть своей секции и объясню что нужно искать и что там лишнее. Вот моя секция:

```
Device (IPIC)
            {
               Name (_HID, EisaId ("PNP0000"))
               Name (_CRS, ResourceTemplate ()
               {
                  IO (Decode16,
                     0x0020,             // Range Minimum
                     0x0020,             // Range Maximum
                     0x01,               // Alignment
                     0x02,               // Length
                     )
                  IO (Decode16,
                     0x0024,             // Range Minimum
                     0x0024,             // Range Maximum
                     0x01,               // Alignment
                     0x02,               // Length
                     )
.....................................
            IO (Decode16,
                     0x04D0,             // Range Minimum
                     0x04D0,             // Range Maximum
                     0x01,               // Alignment
                     0x02,               // Length
                     )
                  IRQNoFlags ()
                     {2}
               })
            }
```

Как видите, у меня эта ссекция называется IPIC. Как говорилось выше, это одно и тоже с PIC. И так, что же тут могло спрятаться и мешать нашей работе? Так так так, а вот оно. Нужно найти все прерывания что находяться тут и убрать их. То есть необходимо взять и убрать все строчки похожие на эту `IRQNoFlags () {2}`. Вместо цифры 2 может стоять абсолютно любое число. Но прерывание 2 необходимо оставить. То есть я не удаляю свои строчки. А если у вас кроме прерывания два стоят какие то еще, то мы их удаляем.

Сохраняем, компилируем и устанавливаем в папку Extra. Пробуем загрузиться. Опять таки, не давайте сразу имя dsdt.aml, иначе могут быть проблемы. Дайте другое имя и потом когда будете загружаться в загрузчике нужно ввести DSDT=/Extra/file_name.aml.

Исправление секции HPET

Ну что? Вроде не слишком уж и сложно все по началу. Давайте сделаем еще одно улучшение. В большинстве случаев это исправление

убирает проблемы с USB и с прочими прерываниями. Исправим секцию HPET. Для этого как мы уже знаем или выбираем эту секцию в поле Location в DSDT SE или же вводим слово HPET в поиск и ищем эту секцию. Она может выглядить по разному. Например моя секция:

```
Device (HPET)
            {
               Name (_HID, EisaId ("PNP0103"))
               Name (_CID, EisaId ("PNP0C01"))
               Name (_CRS, ResourceTemplate ()
               {
                  IRQNoFlags ()
                     {0}
                  IRQNoFlags ()
                     {8}
                  Memory32Fixed (ReadOnly,
                     0xFED00000,         // Address Base
                     0x00000400,         // Address Length
                     )
               })
            }
```

Что же в ней можно исправить? Дело в том что в большинстве случаев на PC необходимы прерывания 0, 8, 11, 15. В моем случае метод IRQNoFlags() резервирует прерывание, то есть у меня зарезервированы прерывания 0 и 8. Необходимо добавить 11 и 15. В конечном итоге у меня получается вот такой код.

```
Device (HPET)
            {
               Name (_HID, EisaId ("PNP0103"))
               Name (_CID, EisaId ("PNP0C01"))
               Name (_CRS, ResourceTemplate ()
               {
                  IRQNoFlags ()
                     {0}
                  IRQNoFlags ()
                     {8}
                  IRQNoFlags ()
                     {11}
                  IRQNoFlags ()
                     {15}
                  Memory32Fixed (ReadOnly,
                     0xFED00000,         // Address Base
                     0x00000400,         // Address Length
                     )
               })
            }
```

Проходим по всему файлу dsdt и ищем где встречаються прерывания с 0, 8, 11, 15, т.е. строчки вида `IRQNoFlags () {11}` и удаляем их. Так как могут быть конфликты.

Бывают и такие случаи что этой секции просто нет. Тогда вот этот код что у меня выше связанный с этой секцией, перенабираем в свой dsdt. Но его необходимо вставить в нужную часть файл, а именно.

Само проще ищем наши секции RTC или TMR(TIMR) и пишем рядом с ними вот эти строчки как у меня выше и должно у вас все получится. Я верю в это.

Опять таки компилируем. Если нет ошибок, значит все отлично. Записываем файл в папку Extra с именем не dsdt.aml и загружаемся с параметром dsdt=/Extra/file_name.aml. Возможно даже сразу вы увидете в логе меньше ошибок чем обычно и у вас появиться USB в системе.

Небольшой вывод о проделаной работе. Зачем же это мы делаем? Немного углублюсь в архитектуру компьютера. Дело в том что в нашем компьютере есть определенные таймеры. И по этим таймерам срабатывают прерывания устройств. Вот в DSDT они прописываются как RTC, TMR и HPET. Как сказал Slice: “HPET - более современный, скоростной и точный, поэтому, все ТАЙМЕРНЫЕ прерывания отнимаем у старых RTC, TMR, и добавляем в HPET.” То есть ваше оборудование уже будет работать благодаря более “умному” таймеру.

Пропись Darwin

Вот тут уже чуть сложнее. Пока не будем углубляться далеко. Дело в том что в DSDT прописаны такие вещи как поведение определенных устройств для разной операционной системы. То есть в windows XP могут работать одни устройства, в Windows 7 работают другие, а в Linux вообще не работают, хотя врядли. Ну так вот. Если не углубляться сильно далеко, то мы берем и ищем слово Windows в нашем DSDT. Если не нашли, то попробуйте Linux и Darwin. Вот пример моей секции где написано определение операционной системы:

```
Scope (_SB.PCI0)
  {
    Method (_INI, 0, NotSerialized)
    {
      Store (0x07D0, OSYS)
      If (CondRefOf (\_OSI, Local0))
      {
        If (_OSI ("Linux"))
        {
          Store (0x03E8, OSYS)
        }
        If (_OSI ("Windows 2001"))
        {
          Store (0x07D1, OSYS)
        }
        If (_OSI ("Windows 2001 SP1"))
        {
          Store (0x07D1, OSYS)
        }
```

```
            If (_OSI ("Windows 2001 SP2"))
            {
               Store (0x07D2, OSYS)
            }
            If (_OSI ("Windows 2006"))
            {
               Store (0x07D6, OSYS)
            }
            If (LAnd (MPEN, LEqual (OSYS, 0x07D1)))
            {
               TRAP (TRTP, \ESCS)
            }
            TRAP (TRTI, SOOT)
         }
      }
   }
```

Давайте рассмотрим код. Вот эти строчки определяют Если Linux (`If (_OSI ("Linux"))`), то в OSYS записывается число `0x03E8`. В дальнейшем по этому числу будет сверяться какие устройства будут работать. Например у меня где то в DSDT существуют вот таки строчки `If (LEqual (OSYS, 0x07D6))` , т.е. OSYS проверяется со значением `0x07D6`. Если посмотреть выше в мой пример кода, то видно что число `0x07D6` соответствует Windows 2006. То есть только в Windows 2006 будет работать, который будет в этом условие. Windows 2006 - это Windows 7. Вот у меня какая то железяка будет работать только в ней. Вот вы можете просмотреть все условия в своем DSDT и принять решение, под какую операционную систему маскироваться. Лично я просмотрел все условия и максимальное значение устройств было для Windows 2006. У вас соответственно вообще может не быть Windows 2006, а могут быть другие операционные системы. Как правило всегда есть Windows 2001. Это Windows XP. Можете замаскироваться под нее. Но я думаю что если в вашем DSDT были предусмотрены разные операционные системы, то железо делалось под самую современную операционную систему и скорее всего это будет Windows. Вот он обычно самый последний в списке операционных систем и думаю вам нужно маскироваться под самую современную что есть у вас в списке. Хотя не факт. Нужно пробовать методом проб и ошибок маскироваться под ту или иную операционную систему. Обычно это или Windows 2006 или Windows 2001.

Я маскируюсь под Windows 2006 и все что мне нужно было сделать это поменять Windows 2006 на Darwin и получилось вот что:

```
Scope (_SB.PCI0)
   {
      Method (_INI, 0, NotSerialized)
      {
         Store (0x07D0, OSYS)
         If (CondRefOf (\_OSI, Local0))
         {
            If (_OSI ("Linux"))
```

```
            {
                Store (0x03E8, OSYS)
            }
            If (_OSI ("Windows 2001"))
            {
                Store (0x07D1, OSYS)
            }
            If (_OSI ("Windows 2001 SP1"))
            {
                Store (0x07D1, OSYS)
            }
            If (_OSI ("Windows 2001 SP2"))
            {
                Store (0x07D2, OSYS)
            }
            If (_OSI ("Darwin"))
            {
                Store (0x07D6, OSYS)
            }
            If (LAnd (MPEN, LEqual (OSYS, 0x07D1)))
            {
                TRAP (TRTP, \ESCS)
            }
            TRAP (TRTI, SOOT)
        }
    }
}
```

Сохраняемся. Компилируем наш DSDT, исправляем все ошибки и компилируем вновь. Сохраняем полученную dsdt.aml в папку Extra с каким нибудь названием чтобы не удалить ранее созданый стабильный dsdt и в случае неудачной загрузки чтобы загрузиться с нормального dsdt. Перезагружаемся и при загрузке в строке пишем DSDT=/Extra/file_name.aml и загружаемся. Если нормально загрузиться и все будет работать как и раньше, а то и лучше, то у вас все получилось. Хотя не факт.

Часть 2

Пропись USB

Ну что? Первая часть была вводной. Если у вас все получилось и вы разобрались что от вас хотят, то можно продолжить редактировать DSDT более углубленно. Правда дальше будет слегка посложнее. Но думаю с моей помощью вы сможете все сделать правильно. Если же вам было очень сложна первая часть и часть вы не поняли, то советую вам прочитать ее еще раз и попробовать все таки выполнить и только тогда переходить ко второй части.

В этой главе я расскажу вам как прописывать USB в DSDT, так чтобы можно было работать с оригинальным (“ванильным”) IOUSBFamily.kext’ом.

Ну что ж. Не сильно люблю большие вступления, давайте сразу перейдем к практике.

На многих современных материнских платах довольно таки часто не работают USB. Чтобы оно появилось используют или в файле com.apple.Boot.plist прописывали строчку USBBusFix = Yes или же использовали VoodooUSBEHC.kext. То есть чтобы проверить правильно мы все прописали или нет необходимо будет удалить строчку в com.apple.Boot.plist и этот кекст от Voodoo.

И так что же нам нужно сделать чтобы избавиться от разных кастылей в нашей операционной системе и в случае обновления достаточно было положить только наш dsdt.aml в папку Extra и все. И все будет работать с родными кекстами. В данном случае USB. И так. открываем наш dsdt.dsl в DSDTSE и ищем секцию где прописаны наши USB. Для этого необходимо или в поле Location выбрать USB/UHCI. Или же в поле Search вписываем USB и ищем секцию напоминающую вот эту:

```
Device (USB0)
        {
            Name (_ADR, 0x001D0000)
            OperationRegion (U1CS, PCI_Config, 0xC4, 0x04)
            Field (U1CS, DWordAcc, NoLock, Preserve)
            {
                U1EN,   2
            }
.....................................
}
```

Также есть блок похож на этот:

```
Device (EHC1)
```

```
        {
            Name (_ADR, 0x001D0007)
            Name (_PRW, Package (0x02)
            {
                0x0D,
                0x03
            })
            Method (_DSM, 4, NotSerialized)
            {
.....................
}
```

USB1, USB2, USB3, EHC1, EHC2 и так далее, т.е. все что связано с USB и EHC нам нужно. Это все блоки где нужно исправлять и добавлять методы. Приведу пример одного USB:

```
Device (USB1)
        {
            Name (_ADR, 0x001D0000)
            OperationRegion (U1CS, PCI_Config, 0xC4, 0x04)
            Field (U1CS, DWordAcc, NoLock, Preserve)
            {
                U1EN,   2
            }

            Method (_PSW, 1, NotSerialized)
            {
                If (Arg0)
                {
                    Store (0x03, U1EN)
                }
                Else
                {
                    Store (Zero, U1EN)
                }
            }

            Method (_S3D, 0, NotSerialized)
            {
                Return (0x02)
            }

            Method (_S4D, 0, NotSerialized)
            {
                Return (0x02)
            }

            Name (_PRW, Package (0x02)
            {
                0x03,
                0x03
            })
            Method (_DSM, 4, NotSerialized)
            {
                Store (Package (0x0F)
                    {
                        "device-id",
                        Buffer (0x04)
                        {
                            0x34, 0x3A, 0x00, 0x00
                        },
```

```
                "AAPL,clock-id",
                Buffer (One)
                {
                    0x01
                },

                "PCI",
                Buffer (One)
                {
                    0x00
                },

                "device_type",
                Buffer (0x05)
                {
                    "UHCI"
                },

                "AAPL,current-available",
                0x04B0,
                "AAPL,current-extra",
                0x02BC,
                "AAPL,current-in-sleep",
                0x03E8,
                Buffer (One)
                {
                    0x00
                }
            }, Local0)
            DTGP (Arg0, Arg1, Arg2, Arg3, RefOf (Local0))
            Return (Local0)
        }
    }
```

Это весь код моего USB. У вас он может существенно отличаться. Но главное что здесь необходимо это метод `Method (_DSM, 4, NotSerialized)` . Давайте рассмотрим его подробнее и что необходимо изменять.

Этого метода у вас может не быть. Тогда просто перенаберите его в тоже место где у меня.

Теперь рассматриваем построчно.

Вот здесь необходимо поставить значения своего USB. Значения ниже.

```
                "device-id",
                Buffer (0x04)
                {
                    0x34, 0x3A, 0x00, 0x00
                },
```

Для каждого из устройств нужно прописать значения из следущих представленных (информация Slice):

Intel чипсет

USB1 – 0x2830 или 0x3a34

USB2 – 0x2831 или 0x3a35

USB3 – 0x2832 или 0x3a36

USB4 – 0x2834 или 0x3a37
USB5 – 0x2835 или 0x3a38
USB6 – 0x3a39
EHC1 – 0x2836 или 0x3a3a
EHC2 – 0x283a или 0x3a3c

nForce чипсет
USB1 – 0x0aa5
USB2 – 0x0aa7
EHC1 – 0x0aa6
EHC2 – 0x0aa9

И так давайте смотреть. Если у вас в Device (USB1) USB1, то в device-id нужно подставить значения 0x3a34. причем смотрите как вписаны значения у меня 0x34, 0x3A, 0x00, 0x00 сначала идет 34, а потом 3A. То есть наоборот попарно. И это для Intel чипсета. Точно также необходимо прописать и остальные устройства. Для USB2 это будет 0x35, 0x3A, 0x00, 0x00 и так далее.

Идем дальше

```
"AAPL,clock-id",
Buffer (One)
{
   0x01
},
```

Тут необходимо поставить уникальный номер USB. Чтобы долго не мучаться, просто ставьте 1,2,3 и так далее. у меня например USB 1 это 1, в USB2 стоит 2. То есть имеется вот такой вид:

```
"AAPL,clock-id",
Buffer (One)
{
   0x02
},
```

И так далее для каждого устройства.
Смотрим дальше.

```
"device_type",
Buffer (0x05)
{
   "UHCI"
},
```

Тут необходимо прописать так. если в Device (USB1) стоит USB то тут вписуется UHCI. То есть как в примере, Если там стоит Device (EHC1) EHC, то нужно прописывать EHCI, вот так:

```
"device_type",
Buffer (0x05)
{
   "EHCI"
},
```

Остальное остается без изменений. То есть приведу еще один пример прописи, чтобы вы сравнили с предыдущим моим примером.

```
Device (EHC1)
        {
            Name (_ADR, 0x001D0007)
            Name (_PRW, Package (0x02)
            {
                0x0D,
                0x03
            })
            Method (_DSM, 4, NotSerialized)
            {
                Store (Package (0x0F)
                    {
                        "device-id",
                        Buffer (0x04)
                        {
                            0x3A, 0x3A, 0x00, 0x00
                        },

                        "AAPL,clock-id",
                        Buffer (One)
                        {
                            0x04
                        },

                        "built-in",
                        Buffer (One)
                        {
                            0x00
                        },

                        "device_type",
                        Buffer (0x05)
                        {
                            "EHCI"
                        },

                        "AAPL,current-available",
                        0x04B0,
                        "AAPL,current-extra",
                        0x02BC,
                        "AAPL,current-in-sleep",
                        0x03E8,
                        Buffer (One)
                        {
                            0x00
                        }
                    }, Local0)
                DTGP (Arg0, Arg1, Arg2, Arg3, RefOf (Local0))
                Return (Local0)
            }
        }
```

Вот отличия от первого примера. Во первых тут прописываем для EHC1 вместо USB1, дальше device-id тут взят из таблицы выше и он равен 0x3A3A. Берем попарно и меняем их местами как в первом примере. Тут правда разницы особо не заметите. Дальше device-type получается EHCI. и clock-id равен 0x04. Так как у меня это по списку

шло 4 устройство. у вас может быть абсолютно любое число. главное чтобы не повторялось с другими USB.

Теперь случай из практики. У меня было устройство USB0, в таблице такого значения не было. Но я не отчаялся и просто как и рассказывалось в clock-id прописал уникальный номер. в device-id подставил значение из первой колонки, а не из второй и у меня все заработало. Так что вы тоже можете пробовать если у вас что то не получается, но описаного выше должно хватить чтобы сделать USB.

Сохраняемся. Компилируем. Ошибок не должно быть. И как обычно. Ложим файл в папку Extra с именем не dsdt.aml и при загрузке указываем что нужно грузиться с другим DSDT, то есть пишем DSDT=/Extra/file_name.aml. И проверяем работоспособность USB. Не забудьте удалить кекст и инструкцию для хамелеона USBFix.

Если USB работает, то необходимо проверить все USB. это удобно делается программой IORegistryExplorer. Показано что куда вы воткнули и что вытыкнули. и смотреть все ли отключаеться, все ли включается из USB. Так же если вы проверяете в IORegistryExplorer, то у ваших USB и EHCI есть такое значение как Errata, так вот, оно не должно равняться 0x00. Должно быть какое то значение. Если вы все правильно сделали, то все должно работать нормально. Лично у меня после этого появилась функция Sleep.

Так же, как рекомендация, после проделанной работы необходимо переименовать ваши USB и EHC согласно следущим значениям(информация Aleox). Если у вас были названы устройства как USB1, USB2 и т.д., то вам необходимо переименовать их в UHC1, UHC2 и т.д. Причем начать с 1, а не с 0. Допустим у меня в DSDT было USB0, я дал ему имя UHC1 и от него продолжил прибавлять единицу, т.е. UHC2, UHC3 и т.д. Если же встречаються EUSB или EHCI, то переименовываем его в EHC1, EHC2 и так далее. Но учтите одну вещь, наши USB и EHCI могут учавствовать еще в чем то в нашем DSDT, то есть не спешите сразу переименовывать все и сразу. Тем более если вы это делаете первый раз. Переименуйте один USB1 в UHC1 и попробуйте скомпилировать. Уверен что у вас будут ошибки. Просто поищите все записи где встречается USB1 и везде переименуйте на UHC1. Искать с помощью поиска. Вводите в поле Search слово USB1 и нажимаете на Next и смотрите. Уверен что найдете еще совпадения, переименовывайте и так исправяться все ошибки. По окончанию исправления повторите тоже самое с остальными устройствами.