OpenCore

Reference Manual 参考手册(0.7.5.6)

[2021.11.11]

目录

1 Introduction介绍

This document provides information on the format of the 本文件提供了有关格式的信息OpenCore user configuration file used to set up the correct functioning of the macOS operating system. It is to be read as the official clarification of expected OpenCore behaviour. All deviations, if found in published OpenCore releases, shall be considered to be documentation or implementation issues which should be reported via the 用户用于设置 macOS 正确运行的配置文件操作系统。它应被理解为对预期的官方澄清 OpenCore 行为。所有偏差，如果在已发布的 OpenCore 版本中发现，应被视为文件或实施问题，应通过报告Acidanthera Bugtracker. An errata sheet is available in 勘误表可在OpenCorePkg repository.

This document is structured as a specification and is not meant to provide a step-by-step guide to configuring an end-user Board Support Package 本文档被构造为规范，并不旨在提供逐步配置最终用户板级支持包(BSP) . The intended audience of the document is anticipated to be programmers and engineers with a basic understanding of macOS internals and UEFI functionality. For these reasons, this document is available exclusively in English, and all other sources or translations of this document are unofficial and may contain errors. 的指南。目标受众预计该文档是对 mac 内部结构有基本了解的程序员和工程师和 UEFI 功能。由于这些原因，本文档仅提供英文版本，本文件的所有其他来源或翻译

Third-party articles, utilities, books, and similar, may be more useful for a wider audience as they could provide guide-like material. However, they are subject to their authors' preferences, misinterpretations of this document, and unavoidable obsolescence. In cases of using such sources, such as 第三方文章、实用程序、书籍和类似内容可能对更广泛 然而，它们受制于作者的偏好，对本文档的误解，以及不可避免的过时。在使用此类来源的情况下，例如Dortania's OpenCore Install Guide and 和related material, please refer back to this document on every decision made and re-evaluate potential implications. 请在做出的每项决定中重新参考本文件并重新评估潜在影响。

Please note that regardless of the sources used, users are required to fully understand every OpenCore configuration option, and the principles behind them, before posting issues to the 请注意，无论使用何种来源，用户都必须完全理解每一个 配置选项，以及它们背后的原则，在将问题发布到Acidanthera Bugtracker.

Note: Creating this document would not have been possible without the invaluable contributions from other people如果没有宝贵的意见，就不可能创建此文档其他人的贡献: Andrey1970, Goldfish64, dakanji, PMheart, and several others, with the full list available in 完整列表可在OpenCorePkg history.

1.1 Generic Terms

- `plist` — Subset of ASCII Property List format written in XML, also know as XML plist format version 1. Uniform Type Identifier (UTI): `com.apple.property-list`. Plists consist of `plist objects`, which are combined to form a hierarchical structure. Due to plist format not being well-defined, all the definitions of this document may only be applied after plist is considered valid by running `plutil -lint`. External references: https://www.apple.com/DTDs/PropertyList-1.0.dtd, `man plutil`.
- `plist type` — plist collections (`plist array`, `plist dictionary`, `plist key`) and primitives (`plist string`, `plist data`, `plist date`, `plist boolean`, `plist integer`, `plist real`).
- `plist object` — definite realisation of `plist type`, which may be interpreted as value.

4 ACPI

4.1 ~~Introduction~~介绍

ACPI (Advanced Configuration and Power Interface) is an open standard to discover and configure computer hardware. The ACPI specification defines standard tables (e.g. `DSDT`, `SSDT`, `FACS`, `DMAR`) and various methods (e.g. `_DSM`, `_PRW`) for implementation. Modern hardware needs few changes to maintain ACPI compatibility and some options for such changes are provided as part of OpenCore.

To compile and disassemble ACPI tables, the iASL compiler developed by ACPICA can be used. A GUI front-end to iASL compiler can be downloaded from Acidanthera/MaciASL.

ACPI changes apply globally (to every operating system) with the following effective order:

- `Patch` is processed.
- `Delete` is processed.
- `Add` is processed.
- `Quirks` are processed.

Applying the changes globally resolves the problems of incorrect operating system detection (consistent with the ACPI specification, not possible before the operating system boots), operating system chainloading, and difficult ACPI debugging. Hence, more attention may be required when writing changes to `_OSI`.

Applying the patches early makes it possible to write so called "proxy" patches, where the original method is patched in the original table and is implemented in the patched table.

There are several sources of ACPI tables and workarounds. Commonly used ACPI tables are provided with OpenCore, VirtualSMC, VoodooPS2, and WhateverGreen releases. Besides those, several third-party instructions may be found on the AppleLife Laboratory and DSDT subforums (e.g. Battery register splitting guide). A slightly more user-friendly explanation of some tables included with OpenCore can also be found in Dortania's Getting started with ACPI guide. For more exotic cases, there are several alternatives such as daliansky's ACPI sample collection (English Translation by 5T33Z0 et al). Please note however, that suggested solutions from third parties may be outdated or may contain errors.

4.2 Properties

1. `Add`
   Type: `plist array`
   Failsafe: Empty
   Description: Load selected tables from the `OC/ACPI` directory.

   To be filled with `plist dict` values, describing each add entry. Refer to the Add Properties section below for details.

2. `Delete`
   Type: `plist array`

5 Booter

5.1 ~~Introduction~~介绍

This section allows the application of different types of UEFI modifications to operating system bootloaders, primarily the Apple bootloader (`boot.efi`). The modifications currently provide various patches and environment alterations for different firmware types. Some of these features were originally implemented as part of AptioMemoryFix.efi, which is no longer maintained. Refer to the Tips and Tricks section for instructions on migration.

If this is used for the first time on customised firmware, the following requirements should be met before starting:

- Most up-to-date UEFI firmware (check the motherboard vendor website).
- `Fast Boot` and `Hardware Fast Boot` disabled in firmware settings if present.
- `Above 4G Decoding` or similar enabled in firmware settings if present. Note that on some motherboards, notably the ASUS WS-X299-PRO, this option results in adverse effects and must be disabled. While no other motherboards with the same issue are known, this option should be checked first whenever erratic boot failures are encountered.
- `DisableIoMapper` quirk enabled, or `VT-d` disabled in firmware settings if present, or ACPI DMAR table deleted.
- **No** 'slide' boot argument present in NVRAM or anywhere else. It is not necessary unless the system cannot be booted at all or `No slide values are usable! Use custom slide!` message can be seen in the log.
- `CFG Lock` (MSR `0xE2` write protection) disabled in firmware settings if present. Refer to the ControlMsrE2 notes for details.
- `CSM` (Compatibility Support Module) disabled in firmware settings if present. On NVIDIA 6xx/AMD 2xx or older, GOP ROM may have to be flashed first. Use GopUpdate (see the second post) or AMD UEFI GOP MAKER in case of any potential confusion.
- `EHCI/XHCI Hand-off` enabled in firmware settings `only` if boot stalls unless USB devices are disconnected.
- `VT-x`, `Hyper Threading`, `Execute Disable Bit` enabled in firmware settings if present.
- While it may not be required, sometimes `Thunderbolt support`, `Intel SGX`, and `Intel Platform Trust` may have to be disabled in firmware settings present.

When debugging sleep issues, Power Nap and automatic power off (which appear to sometimes cause wake to black screen or boot loop issues on older platforms) may be temporarily disabled. The specific issues may vary, but ACPI tables should typically be looked at first.

Here is an example of a defect found on some Z68 motherboards. To turn Power Nap and the others off, run the following commands in Terminal:

```
sudo pmset autopoweroff 0
sudo pmset powernap 0
sudo pmset standby 0
```

Note: These settings may be reset by hardware changes and in certain other circumstances. To view their current state, use the `pmset -g` command in Terminal.

resulting memory map exceeds 4 KB.

Note 1: Since several types of firmware come with incorrect memory protection tables, this quirk often comes paired with `SyncRuntimePermissions`.

Note 2: The need for this quirk is determined by early boot failures. This quirk replaces `EnableWriteUnprotector` on firmware supporting Memory Attribute Tables (MAT). This quirk is typically unnecessary when using `OpenDuetPkg` but may be required to boot macOS 10.6, and earlier, for reasons that are as yet unclear.

17. `ResizeAppleGpuBars`
    Type: `plist integer`
    Failsafe: `-1`
    Description: Reduce GPU PCI BAR sizes for compatibility with macOS.

    This quirk reduces GPU PCI BAR sizes for Apple macOS up to the specified value or lower if it is unsupported. The specified value follows PCI Resizable BAR spec. ~~Use `0` for 1 MB, `1` for 2 MB, `2` for 4 MB, and so on up to `19` for 512 GB.~~ While Apple macOS supports a theoretical 1 GB maximum, ~~which is `10`.~~ in practice all non-default values may not work correctly. For this reason the only supported value for this quirk is the minimal supported BAR size, i.e. `0`. Use `-1` to disable this quirk.

    For development purposes one may take risks and try other values. Consider a GPU with 2 BARs:

    - `BAR0` supports sizes from 256 MB to 8 GB. Its value is 4 GB.
    - `BAR1` supports sizes from 2 MB to 256 MB. Its value is 256 MB.

    Example 1: Setting `ResizeAppleGpuBars` to 1 GB will change `BAR0` to 1 GB and leave `BAR1` unchanged.
    Example 2: Setting `ResizeAppleGpuBars` to 1 MB will change `BAR0` to 256 MB and `BAR0` to 2 MB.
    Example 3: Setting `ResizeAppleGpuBars` to 16 GB will make no changes.

    Note~~1~~: See `ResizeGpuBars` quirk for general GPU PCI BAR size configuration and more details about the technology.

    ~~*Note 2*: Certain GPU drivers do not support non-standard BAR sizes, causing sleep wake issues, for this reason for macOS it is recommended to use minimal supported BAR sizes, i.e. specify `0` (1 MB).~~

18. `SetupVirtualMap`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Setup virtual memory at `SetVirtualAddresses`.

    Some types of firmware access memory by virtual addresses after a `SetVirtualAddresses` call, resulting in early boot crashes. This quirk workarounds the problem by performing early boot identity mapping of assigned virtual addresses to physical memory.

    Note: The need for this quirk is determined by early boot failures.

19. `SignalAppleOS`
    Type: `plist boolean`

6 DeviceProperties

6.1 ~~Introduction~~介绍

Device configuration is provided to macOS with a dedicated buffer, called `EfiDevicePathPropertyDatabase`. This buffer is a serialised map of DevicePaths to a map of property names and their values.

Property data can be debugged with gfxutil. To obtain current property data, use the following command in macOS:

```
ioreg -lw0 -p IODeviceTree -n efi -r -x | grep device-properties |
  sed 's/.*<//;s/>.*//' > /tmp/device-properties.hex &&
  gfxutil /tmp/device-properties.hex /tmp/device-properties.plist &&
  cat /tmp/device-properties.plist
```

Device properties are part of the `IODeviceTree` (`gIODT`) plane of the macOS I/O Registry. This plane has several construction stages relevant for the platform initialisation. While the early construction stage is performed by the XNU kernel in the `IODeviceTreeAlloc` method, the majority of the construction is performed by the platform expert, implemented in `AppleACPIPlatformExpert.kext`.

AppleACPIPlatformExpert incorporates two stages of `IODeviceTree` construction implemented by calling `AppleACPIPlatformExpert::mergeDeviceProperties`:

1. During ACPI table initialisation through the recursive ACPI namespace scanning by the calls to `AppleACPIPlatformExpert::createDTNubs`.
2. During IOService registration (`IOServices::registerService`) callbacks implemented as a part of `AppleACPIPlatformExpert::platformAdjustService` function and its private worker method `AppleACPIPlatformExpert::platformAdjustPCIDevice` specific to the PCI devices.

The application of the stages depends on the device presence in ACPI tables. The first stage applies very early but exclusively to the devices present in ACPI tables. The second stage applies to all devices much later after the PCI configuration and may repeat the first stage if the device was not present in ACPI.

For all kernel extensions that may inspect the `IODeviceTree` plane without probing, such as `Lilu` and its plugins (e.g. `WhateverGreen`), it is especially important to ensure device presence in the ACPI tables. A failure to do so may result **in erratic behaviour** caused by ignoring the injected device properties as they were not constructed at the first stage. See `SSDT-IMEI.dsl` and `SSDT-BRG0.dsl` for an example.

6.2 Properties

1. `Add`
   Type: `plist dict`
   Description: Sets device properties from a map (`plist dict`) of device paths to a map (`plist dict`) of variable names and their values in `plist multidata` format.

   Note 1: Device paths must be provided in canonic string format (e.g. `PciRoot(0x0)/Pci(0x1,0x0)/Pci(0x0,0x0)`).

7 Kernel

7.1 ~~Introduction~~介绍

This section allows the application of different kinds of kernelspace modifications on Apple Kernel (XNU). The modifications currently provide driver (kext) injection, kernel and driver patching, and driver blocking.

7.2 Properties

1. `Add`
   Type: `plist array`
   Failsafe: Empty
   Description: Load selected kernel extensions (kexts) from the `OC/Kexts` directory.

   To be filled with `plist dict` values, describing each kext. Refer to the Add Properties section below for details.

   Note 1: The load order is based on the order in which the kexts appear in the array. Hence, dependencies must appear before kexts that depend on them.

   Note 2: To track the dependency order, inspect the `OSBundleLibraries` key in the `Info.plist` file of the kext being added. Any kext included under the key is a dependency that must appear before the kext being added.

   Note 3: Kexts may have inner kexts (`Plugins`) included in the bundle. Such `Plugins` must be added separately and follow the same global ordering rules as other kexts.

2. `Block`
   Type: `plist array`
   Failsafe: Empty
   Description: Remove selected kernel extensions (kexts) from the prelinked kernel.

   To be filled with `plist dictionary` values, describing each blocked kext. Refer to the Block Properties section below for details.

3. `Emulate`
   Type: `plist dict`
   Description: Emulate certain hardware in kernelspace via parameters described in the Emulate Properties section below.

4. `Force`
   Type: `plist array`
   Failsafe: Empty
   Description: Load kernel extensions (kexts) from the system volume if they are not cached.

   To be filled with `plist dict` values, describing each kext. Refer to the Force Properties section below for details. This section resolves the problem of injecting kexts that depend on other kexts, which are not

8 Misc

8.1 ~~Introduction~~介绍

This section contains miscellaneous configuration options affecting OpenCore operating system loading behaviour in addition to other options that do not readily fit into other sections.

OpenCore broadly follows the "bless" model, also known as the "Apple Boot Policy". The primary purpose of the "bless" model is to allow embedding boot options within the file system (and be accessible through a specialised driver) as well as supporting a broader range of predefined boot paths as compared to the removable media list set out in the UEFI specification.

Partitions can only booted by OpenCore when they meet the requirements of a predefined `Scan policy`. This policy sets out which specific file systems a partition must have, and which specific device types a partition must be located on, to be made available by OpenCore as a boot option. Refer to the `ScanPolicy` property for details.

The scan process starts with enumerating all available partitions, filtered based on the `Scan policy`. Each partition may generate multiple primary and alternate options. Primary options represent operating systems installed on the media, while alternate options represent recovery options for the operating systems on the media.

- Alternate options may exist without primary options and vice versa.
- Options may not necessarily represent operating systems on the same partition.
- Each primary and alternate option can be an auxiliary option or not.
  - Refer to the `HideAuxiliary` section for details.

The algorithm to determine boot options behaves as follows:

1. Obtain all available partition handles filtered based on the `Scan policy` (and driver availability).
2. Obtain all available boot options from the `BootOrder` UEFI variable.
3. For each boot option found:
   - Retrieve the device path of the boot option.
   - Perform fixups (e.g. NVMe subtype correction) and expansion (e.g. for Boot Camp) of the device path.
   - On failure, if it is an OpenCore custom entry device path, pre-construct the corresponding custom entry and succeed.
   - Obtain the device handle by locating the device path of the resulting device path (ignore it on failure).
   - Locate the device handle in the list of partition handles (ignore it if missing).
   - For disk device paths (not specifying a bootloader), execute "bless" (may return > 1 entry).
   - For file device paths, check for presence on the file system directly.
   - On the OpenCore boot partition, exclude all OpenCore bootstrap files by file header checks.
   - Mark device handle as *used* in the list of partition handles if any.
   - Register the resulting entries as primary options and determine their types.
     The option will become auxiliary for some types (e.g. Apple HFS recovery).
4. For each partition handle:
   - If the partition handle is marked as *unused*, execute "bless" primary option list retrieval.

9 NVRAM

9.1 ~~Introduction~~介绍

This section allows setting non-volatile UEFI variables commonly described as NVRAM variables. Refer to `man nvram` for details. The macOS operating system extensively uses NVRAM variables for OS — Bootloader — Firmware intercommunication. Hence, the supply of several NVRAM variables is required for the proper functioning of macOS.

Each NVRAM variable consists of its name, value, attributes (refer to UEFI specification), and its GUID, representing which 'section' the NVRAM variable belongs to. The macOS operating system makes use of several GUIDs, including but not limited to:

- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14` (`APPLE_VENDOR_VARIABLE_GUID`)
- `7C436110-AB2A-4BBB-A880-FE41995C9F82` (`APPLE_BOOT_VARIABLE_GUID`)
- `5EDDA193-A070-416A-85EB-2A1181F45B18` (Apple Hardware Configuration Storage for `MacPro7,1`)
- `8BE4DF61-93CA-11D2-AA0D-00E098032B8C` (`EFI_GLOBAL_VARIABLE_GUID`)
- `4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102` (`OC_VENDOR_VARIABLE_GUID`)

Note: Some of the variables may be added by the PlatformNVRAM or Generic subsections of the PlatformInfo section. Please ensure that variables set in this section do not conflict with items in those subsections as the implementation behaviour is undefined otherwise.

The `OC_FIRMWARE_RUNTIME` protocol implementation, currently offered as a part of the `OpenRuntime` driver, is often required for macOS to function properly. While this brings many benefits, there are some limitations that should be considered for certain use cases.

1. Not all tools may be aware of protected namespaces.
   When `RequestBootVarRouting` is used, `Boot`-prefixed variable access is restricted and protected in a separate namespace. To access the original variables, tools must be aware of the `OC_FIRMWARE_RUNTIME` logic.

9.2 Properties

1. `Add`
   Type: `plist dict`
   Description: Sets NVRAM variables from a map (`plist dict`) of GUIDs to a map (`plist dict`) of variable names and their values in `plist multidata` format. GUIDs must be provided in canonic string format in upper or lower case (e.g. `8BE4DF61-93CA-11D2-AA0D-00E098032B8C`).

   The `EFI_VARIABLE_BOOTSERVICE_ACCESS` and `EFI_VARIABLE_RUNTIME_ACCESS` attributes of created variables are set. Variables will only be set if not present or deleted. That is, to overwrite an existing variable value, add the variable name to the `Delete` section. This approach enables the provision of default values until the operating system takes the lead.

   Note: The implementation behaviour is undefined when the `plist key` does not conform to the GUID format.

  - `nvram-log=1` (enables AppleEFINVRAM logs)
  - `nv_disable=1` (disables NVIDIA GPU acceleration)
  - `nvda_drv=1` (legacy way to enable NVIDIA web driver, removed in 10.12)
  - `npci=0x2000` (legacy, disables `kIOPCIConfiguratorPFM64`)
  - `lapic_dont_panic=1` (disable lapic spurious interrupt panic on AP cores)
  - `panic_on_display_hang=1` (trigger panic on display hang)
  - `panic_on_gpu_hang=1` (trigger panic on GPU hang)
  - `slide=VALUE` (manually set KASLR slide)
  - `smcdebug=VALUE` (`AppleSMC` debug mask)
  - `spin_wait_for_gpu=1` (reduces GPU timeout on high load)
  - `-amd_no_dgpu_accel` (alternative to WhateverGreen's `-radvesa` for new GPUs)
  - `-nehalem_error_disable` (disables the AppleTyMCEDriver)
  - `-no_compat_check` (disable model checking on 10.7+)
  - `-s` (single mode)
  - `-v` (verbose mode)
  - `-x` (safe mode)

  There are multiple external places summarising macOS argument lists: example 1, example 2.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:bootercfg`
  Booter arguments, similar to `boot-args` but for `boot.efi`. Accepts a set of arguments, which are hexadecimal 64-bit values with or without `0x`. At different stages `boot.efi` will request different debugging (logging) modes (e.g. after `ExitBootServices` it will only print to serial). Several booter arguments control whether these requests will succeed. The list of known requests is covered below:
  - `0x00` – `INIT`.
  - `0x01` – `VERBOSE` (e.g. `-v`, force console logging).
  - `0x02` – `EXIT`.
  - `0x03` – `RESET:OK`.
  - `0x04` – `RESET:FAIL` (e.g. unknown `board-id`, hibernate mismatch, panic loop, etc.).
  - `0x05` – `RESET:RECOVERY`.
  - `0x06` – `RECOVERY`.
  - `0x07` – `REAN:START`.
  - `0x08` – `REAN:END`.
  - `0x09` – `DT` (can no longer log to DeviceTree).
  - `0x0A` – `EXITBS:START` (forced serial only).
  - `0x0B` – `EXITBS:END` (forced serial only).
  - `0x0C` – `UNKNOWN`.

  In 10.15, debugging support was defective up to the 10.15.4 release due to refactoring issues as well as the ~~introduction~~ 介绍of a new debug protocol. Some of the arguments and their values below may not be valid for versions prior to 10.15.4. The list of known arguments is covered below:
  - `boot-save-log=VALUE` — debug log save mode for normal boot.
    * `0`
    * `1`
    * `2` — (default).

11 UEFI

11.1 ~~Introduction~~介绍

UEFI (Unified Extensible Firmware Interface) is a specification that defines a software interface between an operating system and platform firmware. This section allows loading additional UEFI modules as well as applying tweaks to the onboard firmware. To inspect firmware contents, apply modifications and perform upgrades UEFITool and supplementary utilities can be used.

11.2 Drivers

Depending on the firmware, a different set of drivers may be required. Loading an incompatible driver may lead the system to unbootable state or even cause permanent firmware damage. Some of the known drivers are listed below:

`AudioDxe*`	HDA audio support driver in UEFI firmware for most Intel and some other analog audio controllers. Staging driver, refer to acidanthera/bugtracker#740 for known issues in AudioDxe.
`btrfs_x64`	Open source BTRFS file system driver, required for booting with OpenLinuxBoot from a file system which is now quite commonly used with Linux.
`CrScreenshotDxe*`	Screenshot making driver saving images to the root of OpenCore partition (ESP) or any available writeable filesystem upon pressing `F10`. This is a modified version of `CrScreenshotDxe` driver by Nikolaj Schlej.
`ExFatDxe`	Proprietary ExFAT file system driver for Bootcamp support commonly found in Apple firmware. For Sandy Bridge and earlier CPUs, the `ExFatDxeLegacy` driver should be used due to the lack of `RDRAND` instruction support.
`ext4_x64`	Open source EXT4 file system driver, required for booting with OpenLinuxBoot from the file system most commonly used with Linux.
`HfsPlus`	Recommended. Proprietary HFS file system driver with bless support commonly found in Apple firmware. For Sandy Bridge and earlier CPUs, the `HfsPlusLegacy` driver should be used due to the lack of `RDRAND` instruction support.
`HiiDatabase*`	HII services support driver from `MdeModulePkg`. This driver is included in most types of firmware starting with the Ivy Bridge generation. Some applications with GUI, such as UEFI Shell, may need this driver to work properly.
`EnhancedFatDxe`	FAT filesystem driver from `FatPkg`. This driver is embedded in all UEFI firmware and cannot be used from OpenCore. Several types of firmware have defective FAT support implementation that may lead to corrupted filesystems on write attempts. Embedding this driver within the firmware may be required in case writing to the EFI partition is needed during the boot process.
`NvmExpressDxe*`	NVMe support driver from `MdeModulePkg`. This driver is included in most firmware starting with the Broadwell generation. For Haswell and earlier, embedding it within the firmware may be more favourable in case a NVMe SSD drive is installed.
`OpenCanopy*`	OpenCore plugin implementing graphical interface.

Note: This option only applies to the `System` renderer. On all known affected systems, `ConsoleMode` must be set to an empty string for this option to work.

13. `UIScale`
    Type: `plist integer`, 8 bit
    Failsafe: `-1`
    Description: User interface scaling factor.

    Corresponds to `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:UIScale` variable.

    - `1` — 1x scaling, corresponds to normal displays.
    - `2` — 2x scaling, corresponds to HiDPI displays.
    - `-1` — leaves the current variable unchanged.
    - `0` — automatically chooses scaling based on the current resolution.

    Note 1: Automatic scale factor detection works on the basis of total pixel area and may fail on small HiDPI displays, in which case the value may be manually managed using the NVRAM section.

    Note 2: When switching from manually specified NVRAM variable to this preference an NVRAM reset may be needed.

14. `UgaPassThrough`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Provide UGA protocol instances on top of GOP protocol instances.

    Some types of firmware do not implement the legacy UGA protocol but this may be required for screen output by older EFI applications such as EfiBoot from 10.4.

11.14 ProtocolOverrides Properties

1. `AppleAudio`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Replaces Apple audio protocols with builtin versions.

   Apple audio protocols allow OpenCore and the macOS bootloader to play sounds and signals for screen reading or audible error reporting. Supported protocols are beep generation and VoiceOver. The VoiceOver protocol is specific to Gibraltar machines (T2) and is not supported before macOS High Sierra (10.13). Older macOS versions use the AppleHDA protocol (which is not currently implemented) instead.

   Only one set of audio protocols can be available at a time, so this setting should be enabled in order to enable audio playback in the OpenCore user interface on Mac systems implementing some of these protocols.

   Note: The backend audio driver needs to be configured in `UEFI Audio` section for these protocols to be able to stream audio.

Description: Enable AVX vector acceleration of SHA-512 and SHA-384 hashing algorithms.

3. `DisableSecurityPolicy`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Disable platform security policy.

   Note: This setting disables various security features of the firmware, defeating the purpose of any kind of Secure Boot. Do NOT enable if using UEFI Secure Boot.

4. `ExitBootServicesDelay`
   Type: `plist integer`
   Failsafe: `0`
   Description: Adds delay in microseconds after `EXIT_BOOT_SERVICES` event.

   This is a very rough workaround to circumvent the `Still waiting for root device` message on some APTIO IV firmware (ASUS Z87-Pro) particularly when using FileVault 2. It appears that for some reason, they execute code in parallel to `EXIT_BOOT_SERVICES`, which results in the SATA controller being inaccessible from macOS. A better approach is required and Acidanthera is open to suggestions. Expect 3 to 5 seconds to be adequate when this quirk is needed.

5. `ForceOcWriteFlash`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enables writing to flash memory for all ~~OpenCore~~ OpenCore-managed NVRAM system variables.

   Note: This value should be disabled on most types of firmware but is left configurable to account for firmware that may have issues with volatile variable storage overflows or similar. Boot issues across multiple OSes can be observed on e.g. Lenovo Thinkpad T430 and T530 without this quirk. Apple variables related to Secure Boot and hibernation are exempt from this for security reasons. Furthermore, some OpenCore variables are exempt for different reasons, such as the boot log due to an available user option, and the TSC frequency due to timing issues. When toggling this option, a NVRAM reset may be required to ensure full functionality.

6. `ForgeUefiSupport`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Implement partial UEFI 2.x support on EFI 1.x firmware.

   This setting allows running some software written for UEFI 2.x firmware like NVIDIA GOP Option ROMs on hardware with older EFI 1.x firmware like `MacPro5,1`.

7. `IgnoreInvalidFlexRatio`
   Type: `plist boolean`
   Failsafe: `false`