OpenCore

Reference Manual (0.7.6.7)

[2021.12.19]

cases, the use of open-source implementations with transparent binary generation (such as OCAT) is encouraged, given that other tools may contain malware. In addition, configurations created for a specific hardware setup should never be used on different hardware setups.

For BIOS booting, a third-party UEFI environment provider is required and `OpenDuetPkg` is one such UEFI environment provider for legacy systems. To run OpenCore on such a legacy system, `OpenDuetPkg` can be installed with a dedicated tool — BootInstall (bundled with OpenCore). Third-party utilities can be used to perform this on systems other than macOS.

For upgrade purposes, refer to the `Differences.pdf` document which provides information about changes to the configuration (as compared to the previous release) as well as to the `Changelog.md` document (which contains a list of modifications across all published updates).

3.3 Contribution

OpenCore can be compiled as a standard EDK II package and requires the EDK II Stable package. The currently supported EDK II release is hosted in acidanthera/audk. Required patches for this package can be found in the `Patches` directory.

When updating the LaTeX documentation (e.g. `Configuration.tex`) please do *not* rebuild the PDF files till merging to master happens. This avoids unnecessary merge conflicts:

- External contributors using the pull-request approach should request the maintainers to handle the PDF rebuild in the pull-request message.
- Internal contributors should rebuild the documentation at merge time in the same or in a separate commit. One can ask another maintainer to rebuild the documentation when lacking the necessary tools in the pull-request message.

The only officially supported toolchain is `XCODE5`. Other toolchains might work but are neither supported nor recommended. Contributions of clean patches are welcome. Please do follow EDK II C Codestyle.

To compile with `XCODE5`, besides Xcode, users should also install NASM and MTOC. The latest Xcode version is recommended for use despite the toolchain name. An example command sequence is as follows:

```
git clone --depth=1 https://github.com/acidanthera/audk UDK
cd UDK
git submodule update --init --recommend-shallow
git clone --depth=1 https://github.com/acidanthera/OpenCorePkg
. ./edksetup.sh
make -C BaseTools
build -a X64 -b RELEASE -t XCODE5 -p OpenCorePkg/OpenCorePkg.dsc
```

Listing 1: Compilation Commands

For IDE usage Xcode projects are available in the root of the repositories. Another approach could be using Language Server Protocols. For example, Sublime Text with LSP for Sublime Text plugin. Add `compile_flags.txt` file with similar content to the UDK root:

```
-I/UefiPackages/MdePkg
-I/UefiPackages/MdePkg/Include
-I/UefiPackages/MdePkg/Include/X64
-I/UefiPackages/MdeModulePkg
-I/UefiPackages/MdeModulePkg/Include
-I/UefiPackages/MdeModulePkg/Include/X64
-I/UefiPackages/OpenCorePkg/Include/AMI
-I/UefiPackages/OpenCorePkg/Include/Acidanthera
-I/UefiPackages/OpenCorePkg/Include/Apple
-I/UefiPackages/OpenCorePkg/Include/Apple/X64
-I/UefiPackages/OpenCorePkg/Include/Duet
-I/UefiPackages/OpenCorePkg/Include/Generic
-I/UefiPackages/OpenCorePkg/Include/Intel
-I/UefiPackages/OpenCorePkg/Include/Microsoft
```

- Enabling XCPM support for an unsupported CPU variant.

Note 1: It may also be the case that the CPU model is supported but there is no power management supported (e.g. virtual machines). In this case, `MinKernel` and `MaxKernel` can be set to restrict CPU virtualisation and dummy power management patches to the particular macOS kernel version.

Note 2: Only the value of `EAX`, which represents the full CPUID, typically needs to be accounted for and remaining bytes should be left as zeroes. The byte order is Little Endian. For example, `C3 06 03 00` stands for CPUID `0x0306C3` (Haswell).

Note 3: For XCPM support it is recommended to use the following combinations. Be warned that one is required to set the correct frequency vectors matching the installed CPU.

- Haswell-E (`0x0306F2`) to Haswell (`0x0306C3`):
  `Cpuid1Data`: `C3 06 03 00 00 00 00 00 00 00 00 00 00 00 00 00`
  `Cpuid1Mask`: `FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00`
- Broadwell-E (`0x0406F1`) to Broadwell (`0x0306D4`):
  `Cpuid1Data`: `D4 06 03 00 00 00 00 00 00 00 00 00 00 00 00 00`
  `Cpuid1Mask`: `FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00`
- Comet Lake U62 (`0x0A0660`) to Comet Lake U42 (`0x0806EC`):
  `Cpuid1Data`: `EC 06 08 00 00 00 00 00 00 00 00 00 00 00 00 00`
  `Cpuid1Mask`: `FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00`
- Rocket Lake (`0x0A0670`) to Comet Lake (~~`0x0906EB`~~`0x0A0655`):
  `Cpuid1Data`: ~~`EB`~~ `55 06` ~~`09`~~ `0A 00 00 00 00 00 00 00 00 00 00 00 00 00 00`
  `Cpuid1Mask`: `FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00`
- ~~Comet Lake U62 (~~Alder Lake (~~`0x0A0660`~~`0x090672`) to Comet Lake ~~U42 (~~(~~`0x0806EC`~~`0x0A0655`):
  `Cpuid1Data`: ~~`EC`~~ `55 06` ~~`08`~~ `0A 00 00 00 00 00 00 00 00 00 00 00 00 00 00`
  `Cpuid1Mask`: `FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00`

Note 4: Be aware that the following configurations are unsupported by XCPM (at least out of the box):

- Consumer Ivy Bridge (`0x0306A9`) as Apple disabled XCPM for Ivy Bridge and recommends legacy power management for these CPUs. `_xcpm_bootstrap` should manually be patched to enforce XCPM on these CPUs instead of this option.
- Low-end CPUs (e.g. Haswell+ Pentium) as they are not supported properly by macOS. Legacy workarounds for older models can be found in the `Special NOTES` section of acidanthera/bugtracker#365.

2. `Cpuid1Mask`
   Type: `plist data`, 16 bytes
   Failsafe: All zero
   Description: Bit mask of active bits in `Cpuid1Data`.

   When each `Cpuid1Mask` bit is set to 0, the original CPU bit is used, otherwise set bits take the value of `Cpuid1Data`.

3. `DummyPowerManagement`
   Type: `plist boolean`
   Failsafe: `false`
   Requirement: 10.4
   Description: Disables `AppleIntelCpuPowerManagement`.

   Note 1: This option is a preferred alternative to `NullCpuPowerManagement.kext` for CPUs without native power management driver in macOS.

   Note 2: While this option is typically needed to disable `AppleIntelCpuPowerManagement` on unsupported platforms, it can also be used to disable this kext in other situations (e.g. with `Cpuid1Data` left blank).

4. `MaxKernel`
   Type: `plist string`
   Failsafe: Empty
   Description: Emulates CPUID and applies `DummyPowerManagement` on specified macOS version or older.

   Note: Refer to the `Add MaxKernel` description for matching logic.

sometimes fails to wake up. For debug kernels `setpowerstate_panic=0` boot argument should be used, which is otherwise equivalent to this quirk.

17. `ProvideCurrentCpuInfo`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.8 (10.14)
    Description: Provides current CPU info to the kernel.

    This quirk ~~currently~~ works differently depending on the CPU:

    - For Microsoft Hyper-V it provides the correct TSC and FSB values to the kernel, as well as disables CPU topology validation ~~.~~ (10.8+).
    - For Intel CPUs it adds support for asymmetrical SMP systems (e.g. Intel Alder Lake) by patching core count to thread count along with the supplemental required changes (10.14+).

    ~~*Note*: These patches currently target Microsoft Hyper-V and may need to be extended for other purposes.~~

18. `SetApfsTrimTimeout`
    Type: `plist integer`
    Failsafe: `-1`
    Requirement: 10.14 (not required for older)
    Description: Set trim timeout in microseconds for APFS filesystems on SSDs.

    The APFS filesystem is designed in a way that the space controlled via the spaceman structure is either used or free. This may be different in other filesystems where the areas can be marked as used, free, and *unmapped*. All free space is trimmed (unmapped/deallocated) at macOS startup. The trimming procedure for NVMe drives happens in LBA ranges due to the nature of the `DSM` command with up to 256 ranges per command. The more fragmented the memory on the drive is, the more commands are necessary to trim all the free space.

    Depending on the SSD controller and the level of drive fragmenation, the trim procedure may take a considerable amount of time, causing noticeable boot slowdown. The APFS driver explicitly ignores previously unmapped areas and repeatedly trims them on boot. To mitigate against such boot slowdowns, the macOS driver introduced a timeout (`9.999999` seconds) that stops the trim operation when not finished in time.

    On several controllers, such as Samsung, where the deallocation process is relatively slow, this timeout can be reached very quickly. Essentially, it means that the level of fragmentation is high, thus macOS will attempt to trim the same lower blocks that have previously been deallocated, but never have enough time to deallocate higher blocks. The outcome is that trimming on such SSDs will be non-functional soon after installation, resulting in additional wear on the flash.

    One way to workaround the problem is to increase the timeout to an extremely high value, which at the cost of slow boot times (extra minutes) will ensure that all the blocks are trimmed. Set this option to a high value, such as `4294967295`, to ensure that all blocks are trimmed. Alternatively, use over-provisioning, if supported, or create a dedicated unmapped partition where the reserve blocks can be found by the controller. Conversely, the trim operation can be disabled by setting a very low timeout value. e.g. `999`. Refer to this article for details.

19. `ThirdPartyDrives`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.6 (not required for older)
    Description: Apply vendor patches to IOAHCIBlockStorage.kext to enable native features for third-party drives, such as TRIM on SSDs or hibernation support on 10.15 and newer.

    Note: This option may be avoided on user preference. NVMe SSDs are compatible without the change. For AHCI SSDs on modern macOS version there is a dedicated built-in utility called `trimforce`. Starting from 10.15 this utility creates `EnableTRIM` variable in `APPLE_BOOT_VARIABLE_GUID` namespace with `01 00 00 00` value.

20. `XhciPortLimit`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.11 (not required for older)

systemd-boot users (probably almost exclusively Arch Linux users) should be aware that OpenLinuxBoot does not support the systemd-boot–specific Boot Loader Interface; therefore `efibootmgr` rather than `bootctl` must be used for any low-level Linux command line interaction with the boot menu.

11.7 Properties

1. `APFS`
   Type: `plist dict`
   Failsafe: None
   Description: Provide APFS support as configured in the APFS Properties section below.

2. `Audio`
   Type: `plist dict`
   Failsafe: None
   Description: Configure audio backend support described in the Audio Properties section below.

   ~~Audio support~~ Unless documented otherwise (e.g. `ResetTrafficClass`) settings in this section are for UEFI audio support only (e.g. OpenCore generated boot chime and audio assist) and are unrelated to any configuration needed for OS audio support (e.g. `AppleALC`).

   UEFI audio support provides a way for upstream protocols to interact with the selected ~~hardware and audio~~ audio hardware and resources. All audio resources should reside in `\EFI\OC\Resources\Audio` directory. Currently the supported audio file formats are MP3 and WAVE PCM. While it is driver-dependent which audio stream format is supported, most common audio cards support 16-bit signed stereo audio at 44100 or 48000 Hz.

   Audio file path is determined by audio type, audio localisation, and audio path. Each filename looks as follows: `[audio type]_[audio localisation]_[audio path].[audio ext]`. For unlocalised files filename does not include the language code and looks as follows: `[audio type]_[audio path].[audio ext]`. Audio extension can either be `mp3` or `wav`.

   - Audio type can be `OCEFIAudio` for OpenCore audio files or `AXEFIAudio` for macOS bootloader audio files.
   - Audio localisation is a two letter language code (e.g. `en`) with an exception for Chinese, Spanish, and Portuguese. Refer to `APPLE_VOICE_OVER_LANGUAGE_CODE` definition for the list of all supported localisations.
   - Audio path is the base filename corresponding to a file identifier. For macOS bootloader audio paths refer to `APPLE_VOICE_OVER_AUDIO_FILE` definition. For OpenCore audio paths refer to `OC_VOICE_OVER_AUDIO_FILE` definition. The only exception is OpenCore boot chime file, which is `OCEFIAudio_VoiceOver_Boot.mp3`.

   Audio localisation is determined separately for macOS bootloader and OpenCore. For macOS bootloader it is set in `preferences.efires` archive in `systemLanguage.utf8` file and is controlled by the operating system. For OpenCore the value of `prev-lang:kbd` variable is used. When native audio localisation of a particular file is missing, English language (`en`) localisation is used. Sample audio files can be found in OcBinaryData repository.

3. `ConnectDrivers`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Perform UEFI controller connection after driver loading.

   This option is useful for loading drivers following UEFI driver model as they may not start by themselves. Examples of such drivers are filesystem or audio drivers. While effective, this option may not be necessary for drivers performing automatic connection, and may slightly slowdown the boot.

   Note: Some types of firmware, particularly those made by Apple, only connect the boot drive to speed up the boot process. Enable this option to be able to see all the boot options when running multiple drives.

4. `Drivers`
   Type: `plist array`
   Failsafe: Empty
   Description: Load selected drivers from `OC/Drivers` directory.

   To be filled with `plist dict` values, describing each driver. Refer to the Drivers Properties section below.

5. `Input`
   Type: `plist dict`
   Failsafe: None

Failsafe: Empty
Description: Device path of the specified audio controller for audio support.

This typically contains builtin analog audio controller (`HDEF`) device path, e.g. `PciRoot(0x0)/Pci(0x1b,0x0)`. The list of recognised audio controllers can be found in the debug log (marked in bold-italic):

```
OCAU: 1/3 PciRoot(0x0)/Pci(0x1,0x0)/Pci(0x0,0x1)/VenMsg(<redacted>,00000000) (4 outputs)
OCAU: 2/3 PciRoot(0x0)/Pci(0x3,0x0)/VenMsg(<redacted>,00000000) (1 outputs)
OCAU: 3/3 PciRoot(0x0)/Pci(0x1B,0x0)/VenMsg(<redacted>,02000000) (7 outputs)
```

~~As an alternative,~~ If using using `AudioDxe`, the required device path is also output as:

`HDA: Connecting controller PciRoot(0x0)/Pci(0x1B,0x0)`

Finally, `gfxutil -f HDEF` command can be used in macOS ~~.~~ to obtain the device path.

Specifying an empty device path will result in the first available audio controller being used, and can be a convenient option to get UEFI audio working if only one audio controller is present.

3. ~~`AudioOut`~~`AudioOutMask`
Type: `plist integer`
Failsafe: `0`
Description: ~~Index of the output port of the specified codec starting from 0.~~ Bit field indicating which output channels to use for UEFI sound.

This ~~typically contains the index of the~~ should typically contain a single bit corresponding to the green out of the builtin analog audio controller (`HDEF`). The number of available output nodes (`N`) for each HDA codec is shown in the debug log (marked in bold-italic):

```
OCAU: 1/3 PciRoot(0x0)/Pci(0x1,0x0)/Pci(0x0,0x1)/VenMsg(<redacted>,00000000) (4 outputs)
OCAU: 2/3 PciRoot(0x0)/Pci(0x3,0x0)/VenMsg(<redacted>,00000000) (1 outputs)
OCAU: 3/3 PciRoot(0x0)/Pci(0x1B,0x0)/VenMsg(<redacted>,02000000) (7 outputs)
```

The ~~quickest way to find the right port is~~ first available output node is bit 0 (value 1), the second node is bit 1 (value 2), etc.

When the debug version of `AudioDxe` is used, then additional information on each output channel of each codec is logged during driver binding. Further information on the available output channels may also be found from a Linux codec dump `cat /proc/asound/card{n}/codec#{m}`.

Using `AudioOutMask`, it is possible to play sound to more than one channel (e.g. main speaker plus bass speaker; headphones plus speakers). For example, if the main speaker is ouput 0 and the bass speaker is output 2, then to play to both set `AudioOutMask` to ~~bruteforce the values from~~ `1 << 0 + 1 << 2` i.e. `5`. This feature is supported when all chosen outputs support the sound file format in use; if any do not then no sound will play and an error will be logged.

Note 1: If all available output channels on the codec support the available sound file format, then a value of ~~`0 to N - 1`~~`-1` may be used to play to all channels simultaneously.

Note 2: Bits in `AudioOutMask` do not represent internal codec node ids as found in detailed codec dumps, but rather the available output nodes as shown e.g. in the `OCAU` log lines above.

4. `AudioSupport`
Type: `plist boolean`
Failsafe: `false`
Description: Activate audio support by connecting to a backend driver.

Enabling this setting routes audio playback from builtin protocols to ~~a dedicated audio port (~~specified (~~`AudioOut`~~`AudioOutMask` dedicated audio ports of the specified codec (`AudioCodec`), located on the specified audio controller (`AudioDevice`).

5. `DisconnectHda`
Type: `plist boolean`
Failsafe: `false`
Description: Disconnect HDA controller before loading drivers.