MAME Documentation

Release 0.194

MAMEdev Team

January 23, 2018

Note: This documentation is a work in progress. You can track the status of these topics through MAME's issue tracker. Learn how you can contribute on GitHub.

CHAPTER

ONE

WHAT IS MAME

MAME is a multi-purpose emulation framework.

MAME's purpose is to preserve decades of software history. As electronic technology continues to rush forward, MAME prevents this important "vintage" software from being lost and forgotten. This is achieved by documenting the hardware and how it functions. The source code to MAME serves as this documentation. The fact that the software is usable serves primarily to validate the accuracy of the documentation (how else can you prove that you have recreated the hardware faithfully?). Over time, MAME (originally stood for Multiple Arcade Machine Emulator) absorbed the sister-project MESS (Multi Emulator Super System), so MAME now documents a wide variety of (mostly vintage) computers, video game consoles and calculators, in addition to the arcade video games that were its initial focus.

I. Purpose

MAME's main purpose is to be a reference to the inner workings of the emulated machines. This is done both for educational purposes and for preservation purposes, in order to prevent historical software from disappearing forever once the hardware it runs on stops working. Of course, in order to preserve the software and demonstrate that the emulated behavior matches the original, one must also be able to actually use the software. This is considered a nice side effect, and is not MAME's primary focus.

It is not our intention to infringe on any copyrights or patents on the original games. All of MAME's source code is either our own or freely available. To operate, the emulator requires images of the original ROMs, CDs, hard disks or other media from the machines, which must be provided by the user. No portions of the original game code are included in the executable.

II. Cost

MAME is free. Its source code is free. The project as whole is distributed under the GNU General Public License, version 2 or later (GPL-2.0+), but most of code (including core functionality) is also available under the 3-clause BSD license (BSD-3-clause).

III. Software Image Files

ROM, CD, hard disk and other media images are all copyrighted material. They cannot be distributed without the explicit permission of the copyright holder(s). They are not "abandonware", nor has any of the software supported by MAME passed out of copyright.

MAME is not intended to be used as a tool for mass copyright infringement. Therefore, it is strongly against the authors' wishes to sell, advertise, or link to resources that provide illegal copies of ROM, CD, hard disk or other media images.

IV. Derivative Works

Because the name MAME is trademarked, you must abide by the rules set out for trademark usage if you wish to use "MAME" as part of the name your derivative work. In general, this means you must request permission, which requires that you follow the guidelines above.

The version number of any derivative work should reflect the version number of the MAME release from which it was was derived.

V. Official Contact Information

For questions regarding the MAME license, trademark, or other usage, go to http://www.mamedev.org/

CHAPTER

TWO

GETTING MAME PREPARED

This section covers initial preparation work needed to use MAME, including downloading and compiling MAME.

An Introduction to MAME

MAME, formerly was an acronym which stood for Multi Arcade Machine Emulator, documents and reproduces through emulation the inner components of arcade machines, computers, consoles, chess computers, calculators, and many other types of electronic amusement machines. As a nice side-effect, MAME allows to use on a modern PC those programs and games which were originally developed for the emulated machines.

At one point there were actually two separate projects, MAME and MESS. MAME covered arcade machines, while MESS covered everything else. They are now merged into the one MAME.

MAME is mostly programmed in C with some core components in C++. MAME can currently emulate over 32000 individual systems from the last 5 decades.

Purpose of MAME

The primary purpose of MAME is to preserve decades of arcade, computer, and console history. As technology continues to rush forward, MAME prevents these important "vintage" systems from being lost and forgotten.

Systems Emulated by MAME

ProjectMESS contains a complete list of the systems currently emulated. As you will notice, being supported does not always mean that the status of the emulation is perfect. You may want

1. to check the status of the emulation in the wiki pages of each system, accessible from the drivers page (e.g. for Apple Macintosh, from the page for the mac.c driver you can reach the pages for both **macplus** and **macse**),
2. to read the corresponding **sysinfo.dat** entry in order to better understand which issues you may encounter while running a system in MAME (again, for Apple Macintosh Plus you have to check this entry).

Alternatively, you can simply see the status by yourself, launching the system emulation and taking a look to the red or yellow warning screen which appears before the emulation starts, if any. Notice that if you have information which can help to improve the emulation of a supported system, or if you can directly contribute fixes and/or addition to the current source, you can follow the instructions at the contact page or post to the MAME Forums at http://forum.mamedev.org/

Supported OS

The current source code can be directly compiled under all the main OSes: Microsoft Windows (both with DirectX/BGFX native support or with SDL support), Linux, FreeBSD, and Mac OS X. Also, both 32-bit and 64-bit are supported, but be aware 64-bit often shows significant speed increases.

System Requirements

MAME is written in fairly generic C/C++, and has been ported to numerous platforms. Over time, as computer hardware has evolved, the MAME code has evolved as well to take advantage of the greater processing power and hardware capabilities offered.

The official MAME binaries are compiled and designed to run on a standard Windows-based system. The minimum requirements are:

- Intel Core series CPU or equivalent, at least 2.0 GHz
- 32-bit OS (Vista SP1 or later on Windows, 10.9 or later on Mac)
- 4 GB RAM
- DirectX 9.0c for Windows
- A Direct3D, or OpenGL capable graphics card
- Any DirectSound capable sound card/onboard audio

Of course, the minimum requirements are just that: minimal. You may not get optimal performance from such a system, but MAME should run. Modern versions of MAME require more power than older versions, so if you have a less-capable PC, you may find that using an older version of MAME may get you better performance, at the cost of greatly lowered accuracy and fewer supported systems.

MAME will take advantage of 3D hardware for compositing artwork and scaling the games to full screen. To make use of this, you should have a modern Direct3D 8-capable video card with at least 16MB of video RAM.

HLSL or GLSL special effects such as CRT simulation will put a very heavy load on your video card, especially at higher resolutions. You will need a fairly powerful modern video card, and the load on your video card goes up exponentially as your resolution increases. If HLSL or GLSL are too intensive, try dropping your output resolution.

Keep in mind that even on the fastest computers available, MAME is still incapable of playing some systems at full speed. The goal of the project isn't to make all system run speedy on your system; the goal is to document the hardware and reproduce the behavior of the hardware as faithfully as possible.

BIOS Dumps and Software

Most of the systems emulated by MAME requires a dump of the internal chips of the original system. These can be obtained by extracting the data from an original unit, or finding them (at your own risk) in the WorldWideWeb. Being copyrighted material, MAME does not come with any of these.

Also, you may want to find some software to be run on the emulated machine. Again, Google and other search engines are your best friends. MAME does not provide any software to be run on the emulated machines because it is very often (almost always, in the case of console software) copyrighted material.

Installing MAME

Microsoft Windows

You simply have to download the latest binary archive available from http://www.mamedev.org and to extract its content to a folder. You will end up with many files (below you will find explanations about some of these), and in particular **MAME.EXE**. This is a command line program. The installation procedure ends here. Easy, isn't it?

Other Operating Systems

In this case, you can either look for pre-compiled (SDL)MAME binaries (e.g. in the repositories of your favorite Linux distro) which should simply extract all the needed files in a folder you choose, or compile the source code by yourself. In the latter case, see our section on *All Platforms*.

Compiling MAME

All Platforms

- Whenever you are changing build parameters, (such as switching between a SDL-based build and a native Windows renderer one, or adding tools to the compile list) you need to run a **make REGENIE=1** to allow the settings to be regenerated. Failure to do this will cause you very difficult to troubleshoot problems.
- If you want to add various additional tools to the compile, such as *CHDMAN*, add a **TOOLS=1** to your make statement, like **make REGENIE=1 TOOLS=1**
- You can do driver specific builds by using *SOURCES=<driver>* in your make statement. For instance, building Pac-Man by itself would be **make SOURCES=src/mame/drivers/pacman.cpp REGENIE=1** including the necessary *REGENIE* for rebuilding the settings.
- Speeding up the compilation can be done by using more cores from your CPU. This is done with the **-j** parameter. *Note: the maximum number you should use is the number of cores your CPU has, plus one. No higher than that will speed up the compilation, and may in fact slow it down.* For instance, **make -j5** on a quad-core CPU will provide optimal speed.
- Debugging information can be added to a compile using *SYMBOLS=1* though most users will not want or need to use this.

Putting all of these together, we get a couple of examples:

Rebuilding MAME for just the Pac-Man driver, with tools, on a quad-core (e.g. i5 or i7) machine:

make SOURCES=src/mame/drivers/pacman.cpp TOOLS=1 REGENIE=1 -j5

Rebuilding MAME on a dual-core (e.g. i3 or laptop i5) machine:

make -j3

Microsoft Windows

Here are specific notes about compiling MAME for Microsoft Windows.

- Refer to the MAME tools site for the latest toolkit for getting MAME compiled on Windows.
- You will need to download the toolset from that link to begin. Periodically, these tools are updated and newer versions of MAME from that point on will **require** updated tools to compile.
- You can do compilation on Visual Studio 2015 (if installed on your PC) by using **make vs2015**. This will always regenerate the settings, so **REGENIE=1** is *not* needed.
- Make sure you get SDL 2 2.0.3 or 2.0.4 as earlier versions are buggy.

Fedora Linux

You'll need a few prerequisites from your distro. Make sure you get SDL2 2.0.3 or 2.0.4 as earlier versions are buggy.

sudo dnf install gcc gcc-c++ SDL2-devel SDL2_ttf-devel libXinerama-devel qt5-qtbase-devel qt5-qttools expat-devel fontconfig-devel alsa-lib-devel

Compilation is exactly as described above in All Platforms.

Debian and Ubuntu (including Raspberry Pi and ODROID devices)

You'll need a few prerequisites from your distro. Make sure you get SDL2 2.0.3 or 2.0.4 as earlier versions are buggy.

sudo apt-get install git build-essential libsdl2-dev libsdl2-ttf-dev libfontconfig-dev qt5-default

Compilation is exactly as described above in All Platforms.

Arch Linux

You'll need a few prerequisites from your distro.

sudo pacman -S base-devel git sdl2 gconf sdl2_ttf gcc qt5

Compilation is exactly as described above in All Platforms.

Apple Mac OS X

You'll need a few prerequisites to get started. Make sure you're on OS X 10.9 Mavericks or later. You will NEED SDL2 2.0.4 for OS X.

- Install **Xcode** from the Mac App Store
- Launch **Xcode**. It will download a few additional prerequisites. Let this run through before proceeding.
- Once that's done, quit **Xcode** and open a **Terminal** window
- Type **xcode-select –install** to install additional tools necessary for MAME

Next you'll need to get SDL2 installed.

- Go to this site and download the *Mac OS X* .dmg file
- If the .dmg doesn't auto-open, open it

- Click 'Macintosh HD' (or whatever your Mac's hard disk is named) in the left pane of a **Finder** window, then open the **Library** folder and drag the **SDL2.framework** folder from the SDL disk image into the **Frameworks** folder

Lastly to begin compiling, use Terminal to navigate to where you have the MAME source tree (*cd* command) and follow the normal compilation instructions from above in All Platforms.

It's possible to get MAME working from 10.6, but a bit more complicated:

- You'll need to install clang-3.7, ld64, libcxx and python27 from MacPorts
- Then add these options to your make command or useroptions.mak:

OVERRIDE_CC=/opt/local/bin/clang-mp-3.7
OVERRIDE_CXX=/opt/local/bin/clang++-mp-3.7
PYTHON_EXECUTABLE=/opt/local/bin/python2.7
ARCHOPTS=-stdlib=libc++

Emscripten Javascript and HTML

First, download and install Emscripten by following the instructions at the official site

Once Emscripten has been installed, it should be possible to compile MAME out-of-the-box using Emscripten's '**emmake**' tool. Because a full MAME compile is too large to load into a web browser at once, you will want to use the SOURCES parameter to compile only a subset of the project, e.g. (in the mame directory):

emmake make SUBTARGET=pacmantest SOURCES=src/mame/drivers/pacman.cpp

The SOURCES parameter should have the path to at least one driver .cpp file. The make process will attempt to locate and include all dependencies necessary to produce a complete build including the specified driver(s). However, sometimes it is necessary to manually specify additional files (using commas) if this process misses something. E.g.:

emmake make SUBTARGET=apple2e SOURCES=src/mame/drivers/apple2e.cpp,src/mame/machine/applefdc.cpp

The value of the SUBTARGET parameter serves only to differentiate multiple builds and need not be set to any specific value.

Other make parameters can also be used, e.g. *-j* for multithreaded compilation as described earlier.

When the compilation reaches the emcc phase, you may see a number of *"unresolved symbol"* warnings. At the moment, this is expected for OpenGL-related functions such as glPointSize. Any others may indicate that an additional dependency file needs to be specified in the SOURCES list. Unfortunately this process is not automated and you will need to search the source tree to locate the files supplying the missing symbols. You may also be able to get away with ignoring the warnings if the code path referencing them is not used at run-time.

If all goes well, a .js file will be output to the current directory. This file cannot be run by itself, but requires an HTML loader to provide it with a canvas to output to and pass in command-line parameters. The Emularity project provides such a loader.

There are example .html files in that repository which can be edited to point to your newly compiled MAME js filename and pass in whatever parameters you desire. You will then need to place all of the following on a web server:

- The compiled MAME .js file
- The .js files from the Emularity package (loader.js, browserfs.js, etc.)
- A .zip file with the ROMs for the MAME driver you would like to run (if any)

- Any software files you would like to run with the MAME driver
- An Emularity loader .html modified to point to all of the above

You need to use a web server instead of opening the local files directly due to security restrictions in modern web browsers.

If the result fails to run, you can open the Web Console in your browser to see any error output which may have been produced (e.g. missing or incorrect ROM files). A "ReferenceError: foo is not defined" error most likely indicates that a needed source file was omitted from the SOURCES list.

CHAPTER

THREE

BASIC MAME USAGE AND CONFIGURATION

This section describes general usage information about MAME. It is intended to cover aspects of using and configuring MAME that are common across all operating systems. For additional OS-specific options, please see the separate documentation for your platform of choice.

Using MAME

If you want to dive right in and skip the command line, there's a nice graphical way to use MAME without the need to download and set up a front end. Simply start MAME with no parameters, by doubleclicking the **mame.exe** file or running it directly from the command line. If you're looking to harness the full power of MAME, keep reading further.

On Macintosh OS X and *nix-based platforms, please be sure to set your font up to match your locale before starting, otherwise you may not be able to read the text due to missing glyphs.

If you are a new MAME user, you could find this emulator a bit complex at first. Let's take a moment to talk about softlists, as they can simplify matters quite a bit. If the content you are trying to play is a documented entry on one of the MAME softlists, starting the content is as easy as

mame.exe <*system*> <*software*>

For instance:

mame.exe nes metroidu

will load the USA version of Metroid for the Nintendo Entertainment System.

Alternatively, you could start MAME with

mame.exe nes

and choose the *software list* from the cartridge slot. From there, you could pick any softlist-compatible software you have in your roms folders. Please note that many older dumps of cartridges and discs may either be bad or require renaming to match up to the softlist in order to work in this way.

If you are loading an arcade board or other non-softlist content, things are only a little more complicated:

The basic usage, from command line, is

mame.exe <*system*> <*media*> <*software*> <*options*>

where

- <*system*> is the shortname of the system you want to emulate (e.g. nes, c64, etc.)
- <*media*> is the switch for the media you want to load (if it's a cartridge, try **-cart** or **-cart1**; if it's a floppy disk, try **-flop** or **-flop1**; if it's a CD-ROM, try **-cdrom**)

- <*software*> is the program / game you want to load (and it can be given either as the fullpath to the file to load, or as the shortname of the file in our software lists)
- <*options*> is any additional command line option for controllers, video, sound, etc.

Remember that if you type a <system> name which does not correspond to any emulated system, MAME will suggest you some possible choices which are close to what you typed; and if you don't know which <media> switch are available, you can always launch

mame.exe <*system*> **-listmedia**

If you don't know what <*options*> are available, there are a few things you can do. First of all, you can check the command line options section of this manual. You can also try one of the many *Frontends* available for MAME.

Alternatively, you should keep in mind the following command line options, which might be very useful on occasion:

mame.exe -help

tells what MAME is the basic structure of MAME launching options, i.e. as explained above.

mame.exe -showusage

gives you the (quite long) list of available command line options for MAME. The main options are described, in the *Universal Commandline Options* section of this manual.

mame.exe -showconfig

gives you a (quite long) list of available configuration options for MAME. These configuration can always be modified at command line, or by editing them in mame.ini which is the main configuration file for MAME. You can find a description of some configuration options in the *Universal Commandline Options* section of the manual (in most cases, each configuration option has a corresponding command line option to configure and modify it).

mame.exe -createconfig

creates a brand new **mame.ini** file, with default configuration settings. Notice that mame.ini is basically a plain text file, hence you can open it with any text editor (e.g. Notepad, Emacs or TextEdit) and configure every option you need. However, no particular tweaks are needed to start, so you can basically leave most of the options unaltered.

If you execute **mame64 -createconfig** when you already have an existing mame.ini from a previous MAME version, MAME automatically updates the pre-existing mame.ini by copying changed options into it.

Once you are more confident with MAME options, you may want to configure a bit more your setup. In this case, keep in mind the order in which options are read; see *Order of Config Loading* for details.

Default Keys

All the keys below are fully configurable in the user interface. This list shows the standard keyboard configuration.

Key	Action
Tab	Toggles the configuration menu.

Continued on next page

Table 3.1 – continued from previous page

~	Toggles the On Screen Display. When the on-screen display is visible, you can use the following keys to control it: * **Up** - select previous parameter to modify * **Down** - select next parameter to modify * **Left** - decrease the value of the selected parameter * **Right** - increase the value of the selected parameter * **Enter** - reset parameter value to its default * **Control+Left** - decrease the value by 10x * **Shift+Left** - decrease the value by 0.1x * **Alt+Left** - decrease the value by the smallest amount * **Control+Right** - increase the value by 10x * **Shift+Right** - increase the value by 0.1x * **Alt+Right** - increase the value by the smallest amount If you are running with -debug, this key sends a 'break' in emulation.
P	Pauses the game.
Shift+P	While paused, advances to next frame. If rewind is enabled, a new rewind save state is also captured.
Shift+~	While paused, loads the most recent rewind save state.
F2	Service Mode for games that support it.
F3	Resets the game.
Shift+F3	Performs a "hard reset", which tears everything down and re-creates it from scratch. This is a more thorough and complete reset than the reset you get from hitting F3.
LCtrl+F3	[SDL ONLY] - Toggle uneven stretch.

Continued on next page

Table 3.1 – continued from previous page

F4	Shows the game palette, decoded GFX, and any tilemaps. Use the Enter key to switch between the three modes (palette, graphics, and tilemaps). Press F4 again to turn off the display. The key controls in each mode vary slightly: Palette/colortable mode: * **[ ]** - switch between palette and colortable modes * **Up/Down** - scroll up/down one line at a time * **Page Up/Page Down** - scroll up/down one page at a time * **Home/End** - move to top/bottom of list * **-/+** - increase/decrease the number of colors per row * **Enter** - switch to graphics viewer Graphics mode: * **[ ]** - switch between different graphics sets * **Up/Down** - scroll up/down one line at a time * **Page Up/Page Down** - scroll up/down one page at a time * **Home/End** - move to top/bottom of list * **Left/Right** - change color displayed * **R** - rotate tiles 90 degrees clockwise * **-/+** - increase/decrease the number of tiles per row * **Enter** - switch to tilemap viewer Tilemap mode: * **[ ]** - switch between different tilemaps * **Up/Down/Left/Right** - scroll 8 pixels at a time * **Shift+Up/Down/Left/Right** - scroll 1 pixel at a time * **Control+Up/Down/Left/Right** - scroll 64 pixels at a time * **R** - rotate tilemap view 90 degrees clockwise * **-/+** - increase/decrease the zoom factor * **Enter** - switch to palette/colortable mode Note: Not all games have decoded graphics and/or tilemaps.

Continued on next page

Table 3.1 – continued from previous page

LCtrl+F4	[*SDL ONLY*] - Toggles keeping aspect ratio.
LCtrl+F5	[*SDL ONLY*] - Toggle Filter.
Alt+Ctrl+F5	[*NON SDL MS WINDOWS ONLY*] - Toggle HLSL Post-Processing.
F6	Toggle cheat mode (if started with "-cheat").
LCtrl+F6	Decrease Prescaling.
F7	Load a save state. You will be requested to press a key to determine which save state you wish to load. *Note that the save state feature is not supported for a large number of* *drivers. If support is not enabled for a given driver, you will receive* *a warning when attempting to save or load.*
LCtrl+F7	Increase Prescaling.
Shift+F7	Create a save state. Requires an additional keypress to identify the state, similar to the load option above.
F8	Decrease frame skip on the fly.
F9	Increase frame skip on the fly.
F10	Toggle speed throttling.
F11	Toggles speed display.
Shift+F11	Toggles internal profiler display (if compiled in).

Continued on next page

Table 3.1 – continued from previous page

Alt+F11	Record HLSL Rendered Video.
F12	Saves a screen snapshot.
Alt+F12	Take HLSL Rendered Snapshot.
Insert	[*WINDOW ONLY, NON-SDL*] Fast forward. While held, runs game with throttling disabled and with the maximum frameskip.
Page DN	[*SDL ONLY*] Fast forward. While held, runs the game with throttling disabled and with the maximum frameskip.
Alt+ENTER	Toggles between full-screen and windowed mode.
Scroll Lock	Default mapping for the **uimodekey**. This key allows users to disable and enable the emulated keyboard in machines that require it. All emulations which require emulated keyboards will start in that mode and you can only access the internal UI (hitting TAB) by first hitting this key. You can change the initial status of the emulated keyboard as presented upon start by using **-ui_active** as detailed below.
Escape	Exits emulator.

MAME Menus

If you started MAME without any command line parameters, you'll be shown the game selection menu immediately. While the keys listed above will let you navigate the menus, you can also use a mouse.

[todo: This needs SERIOUS expansion. Waiting on answer to a few questions..]

Frontends

There are a number of third party tools for MAME to make system and software selection simpler. These tools are called "Frontends", and there are far too many to list conclusively here. Some are free, some are commercial– caveat emptor. Some older frontends predate the merging of MAME and MESS and do not support the additional console, handheld, etc functionality that MAME inherited from MESS.

This following list is not an endorsement of any of these frontends by the MAME team, but simply showing a number of commonly used free frontends as a good starting point to begin from.

QMC2 (multiple platforms)
Download: http://qmc2.batcom-it.net/

IV/Play (Microsoft Windows)
Download: http://www.mameui.info/

EmuLoader (Microsoft Windows)
Download: http://emuloader.mameworld.info/

The MAME team will not provide support for issues with frontends. For support, we suggest contacting the frontend author or trying any of the popular MAME-friendly forums on the internet.

About ROMs and Sets

Handling and updating of ROMs and Sets used in MAME is probably the biggest area of confusion and frustration that MAME users will run into. This section aims to clear up a lot of the most common questions and cover simple details you'll need to know to use MAME effectively.

Let's start with a simple definition of what a ROM is.

What is a ROM image?

For arcade games, a ROM image or file is a copy of all of the data inside a given chip on the arcade motherboard. For most consoles and handhelds, the individual chips are frequently (but not always) merged into a single file. As arcade machines are much more complicated in their design, you'll typically need the data from a number of different chips on the board. Grouping all of the files from Puckman together will get you a **ROM set** of Puckman.

An example ROM image would be the file **pm1_prg1.6e** stored in the **Puckman** ROM set.

Why ROM and not some other name?

ROM stands for Read-Only Memory. The chips used to store the game data were not rewritable and were permanent (as long as the chip wasn't damaged or aged to death!)

As such, a copy of the data necessary to reconstitute and replace a dead data chip on a board became known as a "ROM image" or ROMs for short.

Parents, Clones, Splitting, and Merging

As the MAME developers received their third or fourth revision of Pac-Man, with bugfixes and other code changes, they quickly discovered that nearly all of the board and chips were identical to the previously dumped version. In order to save space, MAME was adjusted to use a parent set system.

A given set, usually (but not necessarily) the most recent bugfixed World revision of a game, will be designated as the parent. All sets that use mostly the same chips (e.g. Japanese Puckman and USA/World Pac-Man) will be clones that contain only the changed data compared to the parent set.

This typically comes up as an error message to the user when trying to run a Clone set without having the Parent set handy. Using the above example, trying to play the USA version of Pac-Man without having the **PUCKMAN.ZIP** parent set will result in an error message that there are missing files.

Now we add the final pieces of the puzzle: non-merged, split, and merged sets.

MAME is extremely versatile about where ROM data is located and is quite intelligent about looking for what it needs. This allows us to do some magic with how we store these ROM sets to save further space.

A **non-merged set** is one that contains absolutely everything necessary for a given game to run in one ZIP file. This is ordinarily very space-inefficient, but is a good way to go if you want to have very few sets and want everything self-contained and easy to work with. We do not recommend this for most users.

A **split set** is one where the parent set contains all of the normal data it should, and the clone sets contain *only* what has changed as compared to the parent set. This saves some space, but isn't quite as efficient as

A **merged set** takes the parent set and one or more clone sets and puts them all inside the parent set's storage. To use the existing Pac-Man example, combining the Puckman, Midway Pac-Man (USA) sets, along with various bootleg versions– and combining it all into ((PUCKMAN.ZIP**, would be making a merged set. A complete merged set with the parent and all clones is the most common format MAME sets are stored in as it saves the most space.

With those basic principles, there are two other kinds of set that will come up in MAME use from time to time.

First, the **BIOS set**: Some arcade machines shared a common hardware platform, such as the Neo-Geo arcade hardware. As the main board had data necessary to start up and self-test the hardware before passing it off to the game cartridge, it's not really appropriate to store that data as part of the game ROM sets. Instead, it is stored as a BIOS image for the system itself (e.g. **NEOGEO.ZIP** for Neo-Geo games)

Secondly, the **device set**. Frequently the arcade manufacturers would reuse pieces of their designs multiple times in order to save on costs and time. Some of these smaller circuits would reappear in later boards that had minimal common ground with the previous boards that used the circuit, so you couldn't just have them share the circuit/ROM data through a normal parent/clone relationship. Instead, these re-used designs and ROM data are categorized as a *Device*, with the data stored as a *Device set*. For instance, Namco used the *Namco 51xx* custom I/O chip to handle the joystick and DIP switches for Galaga and other games, and as such you'll also need the **NAMCO51.ZIP** device set as well as any needed for the game.

Troubleshooting your ROM sets and the history of ROMs

A lot of the frustration users feel towards MAME can be directly tied to what may feel like pointless ROM changes that seem to only serve to make life more difficult for end-users. Understanding the source of these changes and why they are necessary will help you to avoid being blindsided by change and to know what you need to do to keep your sets running.

A large chunk of arcade ROMs and sets existed before emulation did. These early sets were created by arcade owners and used to repair broken boards by replacing damaged chips. Unfortunately, these sets eventually proved to be missing critical information. Many of the early dumps missed a new type of chip that contained, for instance, color palette information for the screen. The earliest emulators approximated colors until the authors discovered the existence of these missing chips. This resulted in a need to go back and get the missing data and update the sets to add the new dumps as needed.

It wouldn't be much longer before it would be discovered that many of the existing sets had bad data for one or more chips. These, too, would need to be re-dumped, and many sets would need complete overhauls.

Occasionally games would be discovered to be completely wrongly documented. Some games thought to be legitimate ended up being bootleg copies from pirate manufacturers. Some games thought to be bootlegs ended up being legit. Some games were completely mistaken as to which region the board was actually from (e.g. World as compared to Japan) and this too would require adjustments and renaming.

Even now, occasional miracle finds occur that change our understanding of these games. As accurate documentation is critical to detailing the history of the arcades, MAME will change sets as needed to keep things as accurate as possible within what the team knows at the time of each release.

This results in very spotty compatibility for ROM sets designated for older versions of MAME. Some games may not have changed much within 20-30 revisions of MAME, and others may have drastically changed multiple times.

If you hit problems with a set not working, there are several things to check– are you trying to run a set meant for an older version of MAME? Do you have any necessary BIOS or Device ROMs? Is this a Clone set that would need to have the Parent as well? MAME will tell you what files are missing as well as where it looked for these files. Use that to determine which set(s) may be missing files.

ROMs and CHDs

ROM chip data tends to be relatively small and gets loaded to system memory outright. Some games also used additional storage mediums such as hard drives, CD-ROMs, DVDs, and Laserdiscs. Those storage mediums are, for multiple technical reasons, not well-suited to being stored the same way as ROM data and won't fit completely in memory in some cases.

Thus, a new format was created for these in the CHD file. **Compressed Hunks of Data**, or CHD for short, are designed very specifically around the needs of mass storage media. Some arcade games, consoles, and PCs will require a CHD to run. As CHDs are already compressed, they should **NOT** be stored in a ZIP or 7Z file as you would for ROM images.

Common Issues and Questions (FAQ)

Disclaimer: The following information is not legal advice and was not written by a lawyer.

Why does my game show an error screen if I insert coins rapidly?

This is not a bug in MAME. On original arcade hardware, you simply could not insert coins as fast as you can mashing the button. The only ways you could get credit feeding at that kind of pace was if the coin mech hardware was defective or if you were physically trying to cheat the coin mech.

In either case, the game would display an error for the operator to look into the situation to prevent cheating them out of their hard-earned cash. Keep a slow, coin-insert-ish pace and you'll not trigger this.

Why is my non-official MAME package (e.g. EmuCR build) broken?

In many cases, updates to various subsystems such as HLSL, BGFX, or Lua plugins come as updates to the external shader files as well as to the core MAME code. Unfortunately, builds that come from third parties may come as just a main MAME executable or with outdated external files, which can break the coupling between these external files and MAME core code. Despite repeated attempts at contacting some of these third parties to warn them, they persist in distributing broken MAME updates.

As we have no control over how third parties distribute these, all we really can do is disclaim the use of sites like EmuCR and say that we cannot provide support for packages we didn't build. Compile your own MAME or use one of the official packages provided by us.

Why does MAME support console games and dumb terminals? Wouldn't it be faster if MAME had just the arcade games? Wouldn't it take less RAM? Wouldn't MAME be faster if you just X?

This is a common misconception. The actual size of the MAME file doesn't affect the speed of it; only the parts that are actively being used are in memory at any given time.

In truth, the additional supported devices are a good thing for MAME as they allow us to stress test sections of the various CPU cores and other parts of the emulation that don't normally see heavy utilization. While a computer and an arcade machine may use the exact same CPU, how they use that CPU can differ pretty dramatically.

No part of MAME is a second-class citizen to any other part. Video poker machines are just as important to document and preserve as arcade games.

There's still room for improvements in MAME's speed, but chances are that if you're not already a skilled programmer any ideas you have will have already been covered. Don't let that discourage you– MAME is open source, and improvements are always welcome.

Why do my Neo Geo ROMs no longer work? How do I get the Humble Bundle Neo Geo sets working?

Recently the Neo Geo BIOS was updated to add a new version of the Universal BIOS. This was done between 0.171 and 0.172, and results in an error trying to load Neo Geo games with an un-updated **neogeo.zip** set.

This also affects the Humble Bundle set: the games themselves are correct and up to date as of MAME 0.173 (and most likely will remain so) though you'll have to pull the ROM set .ZIP files out of the package somehow yourself. However, the Neo Geo BIOS set (**neogeo.zip**) included in the Humble Bundle set is incomplete as of the 0.172 release of MAME.

We suggest you contact the provider of your sets (Humble Bundle and DotEmu) and ask them to update their content to the newest revision. If enough people ask nicely, maybe they'll update the package.

How can I use the Sega Genesis & Mega Drive Classics collection from Steam with MAME?

As of the April 2016 update to the program, the ROM images included in the set are now 100% compatible with MAME and other Genesis/Mega Drive emulators. The ROMs are contained in the **steamapps\Sega Classics\uncompressed ROMs** folder as a series of *.68K* and *.SGD* images that can be loaded directly into MAME. PDF manuals for the games can be found in **steamapps\Sega Classics\manuals** as well.

Why does MAME report "missing files" even if I have the ROMs?

There can be several reasons for this:

- It is not unusual for the ROMs to change for a game between releases of MAME. Why would this happen? Oftentimes, better or more complete ROM dumps are made, or errors are found in the way the ROMs were previously defined. Early versions of MAME were not as meticulous about this issue, but more recent MAME builds are. Additionally, there can be more features of a game emulated in a later release of MAME than an earlier release, requiring more ROM code to run.
- You may find that some games require CHD files. A CHD file is a compressed representation of a game's hard disk, CD-ROM, or laserdisc, and is generally not included as part of a game's ROMs. However, in most cases, these files are required to run the game, and MAME will complain if they cannot be found.
- Some games such as Neo-Geo, Playchoice-10, Convertible Video System, Deco Cassette, MegaTech, MegaPlay, ST-V Titan, and others need their BIOS ROMs in addition to the game ROMs. The BIOS ROMs often contain ROM code that is used for booting the machine, menu processor code on multi-game systems, and code common to all games on a system. BIOS ROMS must be named correctly and left zipped inside your ROMs folder.
- Older versions of MAME needed decryption tables in order for MAME to emulate Capcom Play System 2 (a.k.a. CPS2) games. These are created by team CPS2Shock.
- Some games in MAME are considered "Clones" of another game. This is often the case when the game in question is simply an alternate version of the same game. Common alternate versions of games include versions with text in other languages, versions with different copyright dates, later versions or updates, bootlegs, etc. "Cloned" games often overlap some of the ROM code as the original or "parent" version of the game. To see if you have any "clones" type "**MAME -listclones**". To run a "cloned game" you simply need to place its parent ROM file in your ROMs folder (leave it zipped).

How can I be sure I have the right ROMs?

MAME checks to be sure you have the right ROMs before emulation begins. If you see any error messages, your ROMs are not those tested to work properly with MAME. You will need to obtain a correct set of ROMs through legal

methods.

If you have several games and you wish to verify that they are compatible with the current version of MAME, you can use the *-verifyroms* parameter. For example:

mame -verifyroms robby ...checks your ROMs for the game *Robby Roto* and displays the results on the screen.

mame -verifyroms * >verify.txt ...checks the validity of ALL the ROMs in your ROMS directory, and writes the results to a textfile called *verify.txt*.

Why is it that some games have the US version as the main set, some have Japanese, and some are the World?

While this rule isn't always true, there is typically a method to how sets are arranged. The usual priority is to go with the **World** set if it's available, **US** if no World English set exists, and **Japanese** or other origin region if no World or US English set.

Exceptions arise where the US or World sets have significant censorship/changes from the original version. For instance, Gals Panic (set **galsnew**) uses the US version as parent because it has additional features compared to the world export version (set **galsnewa**). These features are optional censorship, an additional control layout option (stick with no button use), and English-language voice clips.

Another exception comes for games where it was licensed to a third party for export release. Pac-Man, for instance, was published by Midway in the US though it was created by Namco. As a result, the parent set is the Japanese **puckman** set, which retains the Namco copyright.

Lastly, a developer adding a new set can choose to use whatever naming and parent scheme they wish and are not restricted to the above rules. Most follow these guidelines, however.

How do I legally obtain ROMs or disk images to run on MAME?

You have several options:

- You can obtain a license to them by purchasing one via a distributor or vendor who has proper authority to do so.
- You can download one of the ROM sets that have been released for free to the public for non-commerical use.
- You can purchase an actual arcade PCB, read the ROMs or disks yourself, and let MAME use that data.

Beyond these options, you are on your own.

Isn't copying ROMs a legal gray area?

No, it's not. You are not permitted to make copies of software without the copyright owner's permission. This is a black & white issue.

Can't game ROMs be considered abandonware?

No. Even the companies that went under had their assets purchased by somebody, and that person is the copyright owner.

I had ROMs that worked with an old version of MAME and now they don't. What happened?

As time passes, MAME is perfecting the emulation of older games, even when the results aren't immediately obvious to the user. Often times the better emulation requires more data from the original game to operate. Sometimes the data was overlooked, sometimes it simply wasn't feasible to get at it (for instance, chip "decapping" is a technique that only became affordable very recently for people not working in high-end laboratories). In other cases it's much simpler: more sets of a game were dumped and it was decided to change which sets were which version.

What about those arcade cabinets on eBay that come with all the ROMs?

If the seller does not have a proper license to include the ROMs with his system, he is not allowed to legally include any ROMs with his system. If he has purchased a license to the ROMs in your name from a distributor or vendor with legitimate licenses, then he is okay to include them with the cabinet. After signing an agreement, cabinet owners that include legitimate licensed ROMs may be permitted to include a version of MAME that runs those ROMs and nothing more.

What about those guys who burn DVDs of ROMs for the price of the media?

What they are doing is just as illegal as selling the ROMs outright. As long as somebody owns the copyright, making illegal copies is illegal, period. If someone went on the internet and started a business of selling cheap copies of the latest U2 album for the price of media, do you think they would get away with it?

Even worse, a lot of these folks like to claim that they are helping the project. In fact, they only create more problems for the MAME team. We are not associated with these people in any way regardless of how "official" they may attempt to appear. You are only helping criminals make a profit through selling software they have no right to sell. **Anybody using the MAME name and/or logo to sell such products is also in violation of the MAME trademark.**

But isn't there a special DMCA exemption that makes ROM copying legal?

No, you have misread the exemptions. The exemption allows people to reverse engineer the copy protection or encryption in computer programs that are obsolete. The exemption simply means that figuring out how these obsolete programs worked is not illegal according to the DMCA. It does not have any effect on the legality of violating the copyright on computer programs, which is what you are doing if you make copies of ROMs.

But isn't it OK to download and "try" ROMs for 24 hours?

This is an urban legend that was made up by people who put ROMs up for download on their sites, in order to justify the fact that they were breaking the law. There is nothing like this in any copyright law.

If I buy a cabinet with legitimate ROMs, can I set it up in a public place to make money?

Probably not. ROMs are typically only licensed for personal, non-commercial purposes.

But I've seen Ultracade and Global VR Classics cabinets out in public places? Why can they do it?

Ultracade had two separate products. The Ultracade product is a commercial machine with commercial licenses to the games. These machines were designed to be put on location and make money, like traditional arcade machines. Their other product is the Arcade Legends series. These are home machines with non- commercial licenses for the games, and can only be legally operated in a private environment. Since their buyout by Global VR they only offer the Global VR Classics cabinet, which is equivalent to the earlier Ultracade product.

HELP! I'm getting a black screen or an error message in regards to DirectX on Windows!

You probably have missing or damaged DirectX runtimes. You can download the latest DirectX setup tool from Microsoft at https://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=35

Additional troubleshooting information can be found on Microsoft's website at https://support.microsoft.com/en-us/kb/179113

I have a controller that doesn't want to work with the standard Microsoft Windows version of MAME, what can I do?

By default, MAME on Microsoft Windows tries to do raw reads of the joystick(s), mouse/mice, and keyboard(s). This works with most devices and provides the most stable results. However, some devices need special drivers to translate their output and these drivers may not work with raw input.

One thing you can try is setting the keyboardprovider, mouseprovider, or joystickprovider setting (depending on which kind of device your input device acts as) from rawinput to one of the other options such as dinput or win32. See *OSD related options* for details on supported providers.

CHAPTER

FOUR

MAME COMMANDLINE USAGE AND OS-SPECIFIC CONFIGURATION

Universal Commandline Options

This section contains configuration options that are applicable to *all* MAME sub-builds (both SDL and Windows native).

Core commands

-help / -h / -?

Displays current MAME version and copyright notice.

-validate / -valid

Performs internal validation on every driver in the system. Run this before submitting changes to ensure that you haven't violated any of the core system rules.

Configuration commands

-createconfig / -cc

Creates the default mame.ini file. All the configuration options (not commands) described below can be permanently changed by editing this configuration file.

-showconfig / -sc

Displays the current configuration settings. If you route this to a file, you can use it as an INI file. For example, the command:

mame -showconfig >mame.ini

is equivalent to **-createconfig**.

-showusage / -su

Displays a summary of all the command line options. For options that are not mentioned here, the short summary given by "mame -showusage" is usually sufficient.

Frontend commands

Note: By default, all the '**-list**' commands below write info to the screen. If you wish to write the info to a textfile instead, add this to the end of your command:

> filename

...where 'filename' is the textfile's path and name (e.g., c:\mame\list.txt).

-listxml / -lx [<*gamename|wildcard*>]

List comprehensive details for all of the supported games. The output is quite long, so it is usually better to redirect this into a file. The output is in XML format. By default all games are listed; however, you can limit this list by specifying a driver name or wildcard after the -listxml command.

-listfull / -ll [<*gamename|wildcard*>]

Displays a list of game driver names and descriptions. By default all games are listed; however, you can limit this list by specifying a driver name or wildcard after the **-listfull** command.

-listsource / -ls [<*gamename|wildcard*>]

Displays a list of drivers and the names of the source files their game drivers live in. Useful for finding which driver a game runs on in order to fix bugs. By default all games are listed; however, you can limit this list by specifying a driver name or wildcard after the **-listsource** command.

-listclones / -lc [<*gamename|wildcard*>]

Displays a list of clones. By default all clones are listed; however, you can limit this list by specifying a driver name or wildcard after the **-listsource** command.

-listbrothers / -lb [<*gamename|wildcard*>]

Displays a list of '*brothers*', or rather, other sets which are located in the same sourcefile as the gamename searched for.

-listcrc [<*gamename|wildcard*>]

Displays a full list of CRCs of all ROM images referenced by all drivers within MAME.

-listroms [<*gamename|wildcard*>]

Displays a list of ROM images referenced by the specified game.

-listsamples [<*gamename|wildcard*>]

Displays a list of samples referenced by the specified game.

-verifyroms [<*gamename|wildcard*>]

Checks for invalid or missing ROM images. By default all drivers that have valid ZIP files or directories in the rompath are verified; however, you can limit this list by specifying a driver name or wildcard after the **-verifyroms** command.

-verifysamples [<*gamename|wildcard*>]

Checks for invalid or missing samples. By default all drivers that have valid ZIP files or directories in the samplepath are verified; however, you can limit this list by specifying a driver name or wildcard after the **-verifyroms** command.

-romident [*path\to\romstocheck.zip*]

Attempts to identify ROM files, if they are known to MAME, in the specified .zip file or directory. This command can be used to try and identify ROM sets taken from unknown boards. On exit, the errorlevel is returned as one of the following:

- 0: means all files were identified
- 7: means all files were identified except for 1 or more "non-ROM" files
- 8: means some files were identified
- 9: means no files were identified

-listdevices / -ld [<*gamename|wildcard*>]

Displays a list of all devices known to be hooked up to a game. The ":" is considered the game itself with the devices list being attached to give the user a better understanding of what the emulation is using.

-listslots [<*gamename|wildcard*>]

Show available slots and options for each slot (if available). Primarily used for MAME to allow control over internal plug-in cards, much like PCs needing video, sound and other expansion cards.

The slot name (e.g. **ctrl1**) can be used from the command line (**-ctrl1** in this case)

-listmedia / -lm [<*gamename|wildcard*>]

List available media that the chosen game or system allows to be used. This includes media types (cartridge, cassette, diskette and more) as well as common file extentions which are supported.

-listsoftware [<*gamename|wildcard*>]

Posts to screen all software lists which can be used by the entered gamename or system. Note that this is simply a copy/paste of the .XML file which reside in the HASH folder which are allowed to be used.

-verifysoftware [<*gamename|wildcard*>]

Checks for invalid or missing ROM images in your software lists. By default all drivers that have valid ZIP files or directories in the rompath are verified; however, you can limit this list by specifying a specific driver name or wildcard after the -verifysoftware command.

-getsoftlist [<*gamename|wildcard*>]

Posts to screen a specific software list which matches with the gamename provided.

-verifysoftlist [softwarelistname]

Checks a specified software list for missing ROM images if files exist for issued softwarelistname. By default, all drivers that have valid ZIP files or directories in the rompath are verified; however, you can limit this list by specifying a specific softwarelistname (without .XML) after the -verifysoftlist command.

OSD related options

-uimodekey [*keystring*]

Key used to toggle emulated keyboard on and off. Default setting is *SCRLOCK*.

-uifontprovider

Chooses provider for UI font: win, none or auto. Default setting is *AUTO*.

-menu

Enables menu bar at the top of the MAME window, if available by UI implementation. Default is *OFF*

-keyboardprovider

Chooses how MAME will get keyboard input.

On Windows, you can choose from: auto, rawinput, dinput, win32, or none On SDL, you can choose from: auto, sdl, none

The default is *auto*. On Windows, auto will try rawinput with fallback to dinput. On SDL, auto will default to sdl.

-mouseprovider

Chooses how MAME will get mouse input.

On Windows, you can choose from: auto, rawinput, dinput, win32, or none On SDL, you can choose from: auto, sdl, none

The default is *auto*. On Windows, auto will try rawinput with fallback to dinput. On SDL, auto will default to sdl.

-lightgunprovider

Chooses how MAME will get light gun input.

On Windows, you can choose from: auto, rawinput, win32, or none On SDL, you can choose from: auto, x11 or none

The default is *auto*. On Windows, auto will try rawinput with fallback to win32, or none if it doesn't find any. On SDL/Linux, auto will default to x11, or none if it doesn't find any. On other SDL, auto will default to none.

-joystickprovider

Chooses how MAME will get joystick input.

On Windows, you can choose from: auto, winhybrid, dinput, xinput, or none On SDL, you can choose from: auto, sdl, none

The default is *auto*. On Windows, auto will default to dinput.

Note that Microsoft X-Box 360 and X-Box One controllers will be happiest with *winhybrid* or *xinput*. The *winhybrid* option supports a mix of DirectInput and XInput controllers at the same time. On SDL, auto will default to sdl.

OSD CLI options

-listmidi

Create a list of available MIDI I/O devices for use with emulation.

-listnetwork

Create a list of available Network Adapters for use with emulation.

OSD output options

-output

Chooses how MAME will handle processing of output notifiers.

you can choose from: auto, none, console or network

Note that network port is fixed at 8000.

Configuration options

-[no]readconfig / -[no]rc

Enables or disables the reading of the config files. When enabled (which is the default), MAME reads the following config files in order:

- mame.ini

- <mymame>.ini (i.e. if MAME was renamed mame060.exe, MAME parses mame060.ini here)
- debug.ini (if the debugger is enabled)
- <driver>.ini (based on the source filename of the driver)
- vertical.ini (for games with vertical monitor orientation)
- horizont.ini (for games with horizontal monitor orientation)
- arcade.ini (for games in source added with GAME() macro)
- console.ini (for games in source added with CONS() macro)
- computer.ini (for games in source added with COMP() macro)
- othersys.ini (for games in source added with SYST() macro)
- vector.ini (for vector games only)
- <parent>.ini (for clones only, may be called recursively)
- <gamename>.ini

(See *Order of Config Loading* for further details)

The settings in the later INIs override those in the earlier INIs. So, for example, if you wanted to disable overlay effects in the vector games, you can create a vector.ini with the "effect none" line in it, and it will override whatever effect value you have in your mame.ini. The default is ON (*-readconfig*).

Core search path options

-rompath / -rp <*path*>

Specifies a list of paths within which to find ROM or hard disk images. Multiple paths can be specified by separating them with semicolons. The default is 'roms' (that is, a directory "roms" in the same directory as the MAME executable).

-hashpath <*path*>

Specifies a list of paths within which to find Software List HASH files. Multiple paths can be specified by separating them with semicolons. The default is 'hash' (that is, a directory "roms" in the same directory as the MAME executable).

-samplepath / -sp <*path*>

Specifies a list of paths within which to find sample files. Multiple paths can be specified by separating them with semicolons. The default is 'samples' (that is, a directory "samples" in the same directory as the MAME executable).

-artpath <*path*> **/ -artwork_directory** <*path*>

Specifies a list of paths within which to find artwork files. Multiple paths can be specified by separating them with semicolons. The default is 'artwork' (that is, a directory "artwork" in the same directory as the MAME executable).

-ctrlrpath / -ctrlr_directory <*path*>

Specifies a list of paths within which to find controller-specific configuration files. Multiple paths can be specified by separating them with semicolons. The default is 'ctrlr' (that is, a directory "ctrlr" in the same directory as the MAME executable).

-inipath <*path*>

Specifies a list of paths within which to find .INI files. Multiple paths can be specified by separating them with semicolons. The default is '.;ini' (that is, search in the current directory first, and then in the directory "ini" in the same directory as the MAME executable).

-fontpath <*path*>

Specifies a list of paths within which to find .BDF font files. Multiple paths can be specified by separating them with semicolons. The default is '.' (that is, search in the same directory as the MAME executable).

-cheatpath <*path*>

Specifies a list of paths within which to find .XML cheat files. Multiple paths can be specified by separating them with semicolons. The default is 'cheat' (that is, a folder called 'cheat' located in the same directory as the as the MAME executable).

-crosshairpath <*path*>

Specifies a list of paths within which to find crosshair files. Multiple paths can be specified by separating them with semicolons. The default is 'crsshair' (that is, a directory "crsshair" in the same directory as the MAME executable). If the Crosshair is set to default in the menu, MAME will look for gamename\cross#.png and then cross#.png in the specified crsshairpath, where # is the player number. Failing that, MAME will use built-in default crosshairs.

-pluginspath <*path*>

Specifies a list of paths within which to find Lua plugins for MAME.

-languagepath <*path*>

Specifies a list of paths within which to find language files for localized UI text.

Core Output Directory Options

-cfg_directory <*path*>

Specifies a single directory where configuration files are stored. Configuration files store user configurable settings that are read at startup and written when MAME exits. The default is 'cfg' (that is, a directory "cfg" in the same directory as the MAME executable). If this directory does not exist, it will be automatically created.

-nvram_directory <*path*>

Specifies a single directory where NVRAM files are stored. NVRAM files store the contents of EEPROM and non-volatile RAM (NVRAM) for games which used this type of hardware. This data is read at startup and written when MAME exits. The default is 'nvram' (that is, a directory "nvram" in the same directory as the MAME executable). If this directory does not exist, it will be automatically created.

-input_directory <*path*>

Specifies a single directory where input recording files are stored. Input recordings are created via the -record option and played back via the -playback option. The default is 'inp' (that is, a directory "inp" in the same directory as the MAME executable). If this directory does not exist, it will be automatically created.

-state_directory <*path*>

Specifies a single directory where save state files are stored. Save state files are read and written either upon user request, or when using the -autosave option. The default is 'sta' (that is, a directory "sta" in the same directory as the MAME executable). If this directory does not exist, it will be automatically created.

-snapshot_directory <*path*>

Specifies a single directory where screen snapshots are stored, when requested by the user. The default is 'snap' (that is, a directory "snap" in the same directory as the MAME executable). If this directory does not exist, it will be automatically created.

-diff_directory <*path*>

Specifies a single directory where hard drive differencing files are stored. Hard drive differencing files store any data that is written back to a hard disk image, in order to preserve the original image. The differencing files are created at startup when a game with a hard disk image. The default is 'diff' (that is, a directory "diff" in the same directory as the MAME executable). If this directory does not exist, it will be automatically created.

-comment_directory <*path*>

Specifies a single directory where debugger comment files are stored. Debugger comment files are written by the debugger when comments are added to the disassembly for a game. The default is 'comments' (that is, a directory "comments" in the same directory as the MAME executable). If this directory does not exist, it will be automatically created.

Core state/playback options

-[no]rewind

When enabled and emulation is paused, automatically creates a save state in memory every time a frame is advanced. Rewind save states can then be loaded consecutively by pressing the rewind single step shortcut key (*Left Shift + Tilde by default*). The default rewind value is OFF (-norewind).

If debugger is in a 'break' state, a save state is instead created every time step in, step over, or step out occurs. In that mode, rewind save states can be loaded by executing the debugger 'rewind'(or 'rw') command.

-rewind_capacity <*value*>

Sets the rewind capacity value, in megabytes. It is the total amount of memory rewind savestates can occupy. When capacity is hit, old savestates get erased as new ones are captured. Setting capacity lower than the current savestate size disables rewind. Values below 0 are automatically clamped to 0.

-state <*slot*>

Immediately after starting the specified game, will cause the save state in the specified <slot> to be loaded.

-[no]autosave

When enabled, automatically creates a save state file when exiting MAME and automatically attempts to reload it when later starting MAME with the same game. This only works for games that have explicitly enabled save state support in their driver. The default is OFF (-noautosave).

-playback / -pb <*filename*>

Specifies a file from which to play back a series of game inputs. Thisfeature does not work reliably for all games, but can be used to watch a previously recorded game session from start to finish. In order to make things consistent, you should only record and playback with all configuration (.cfg), NVRAM (.nv), and memory card files deleted. The default is NULL (no playback).

-exit_after_playback

Tells MAME to exit after finishing playback of the input file.

-record / -rec <*filename*>

Specifies a file to record all input from a game session. This can be used to record a game session for later playback. This feature does not work reliably for all games, but can be used to watch a previously

recorded game session from start to finish. In order to make things consistent, you should only record and playback with all configuration (.cfg), NVRAM (.nv), and memory card files deleted. The default is NULL (no recording).

-record_timecode

Tells MAME to create a timecode file. It contains a line with elapsed times on each press of timecode shortcut key (*default is F12*). This option works only when recording mode is enabled (**-record** option). The file is saved in the *inp* folder. By default, no timecode file is saved.

-mngwrite <*filename*>

Writes each video frame to the given <filename> in MNG format, producing an animation of the game session. Note that -mngwrite only writes video frames; it does not save any audio data. Use -wavwrite for that, and reassemble the audio/video using offline tools. The default is NULL (no recording).

-aviwrite <*filename*>

Stream video and sound data to the given <filename> in AVI format, producing an animation of the game session complete with sound. The default is NULL (no recording).

-wavwrite <*filename*>

Writes the final mixer output to the given <filename> in WAV format, producing an audio recording of the game session. The default is NULL (no recording).

-snapname <*name*>

Describes how MAME should name files for snapshots. <name> is a string that provides a template that is used to generate a filename.

Three simple substitutions are provided: the / character represents the path separator on any target platform (even Windows); the string %g represents the driver name of the current game; and the string %i represents an incrementing index. If %i is omitted, then each snapshot taken will overwrite the previous one; otherwise, MAME will find the next empty value for %i and use that for a filename.

The default is %g/%i, which creates a separate folder for each game, and names the snapshots under it starting with 0000 and increasing from there.

In addition to the above, for drivers using different media, like carts or floppy disks, you can also use the %d_[media] indicator. Replace [media] with the media switch you want to use.

A few examples: if you use 'mame robby -snapname foo/%g%i' snapshots will be saved as 'snaps\foo\robby0000.png' , 'snaps\foo\robby0001.png' and so on; if you use 'mame nes -cart robby -snapname %g/%d_cart' snapshots will be saved as 'snaps\nes\robby.png' ; if you use 'mame c64 -flop1 robby -snapname %g/%d_flop1/%i' snapshots will be saved as 'snaps\c64\robby\0000.png'.

-snapsize <*width*>*x*<*height*>

Hard-codes the size for snapshots and movie recording. By default, MAME will create snapshots at the game's current resolution in raw pixels, and will create movies at the game's starting resolution in raw pixels. If you specify this option, then MAME will create both snapshots and movies at the size specified, and will bilinear filter the result. Note that this size does not automatically rotate if the game is vertically oriented. The default is '*auto*'.

-snapview <*viewname*>

Specifies the view to use when rendering snapshots and movies. By default, both use a special 'internal' view, which renders a separate snapshot per screen or renders movies only of the first screen. By specifying this option, you can override this default behavior and select a single view that will apply to all snapshots and movies. Note that <viewname> does not need to be a perfect match; rather, it will select the first view whose name matches all the characters specified by <viewname>.

For example, **-snapview native** will match the "Native (15:14)" view even though it is not a perfect match. <viewname> can also be 'auto', which selects the first view with all screens present. The default value is '*internal*'.

-[no]snapbilinear

Specify if the snapshot or movie should have bilinear filtering applied. Shutting this off can make a difference in some performance while recording video to a file. The default is ON (*-snapbilinear*).

-statename <*name*>

Describes how MAME should store save state files, relative to the state_directory path. <name> is a string that provides a template that is used to generate a relative path.

Two simple substitutions are provided: the / character represents the path separator on any target platform (even Windows); the string %g represents the driver name of the current game.

The default is %g, which creates a separate folder for each game.

In addition to the above, for drivers using different media, like carts or floppy disks, you can also use the %d_[media] indicator. Replace [media] with the media switch you want to use.

A few examples: if you use 'mame robby -statename foo/%g' save states will be stored inside 'sta\foo\robby\' ; if you use 'mame nes -cart robby -statename %g/%d_cart' save states will be stored inside 'sta\nes\robby\' ; if you use 'mame c64 -flop1 robby -statename %g/%d_flop1' save states will be stored inside 'sta\c64\robby\'.

-[no]burnin

Tracks brightness of the screen during play and at the end of emulation generates a PNG that can be used to simulate burn-in effects on other games. The resulting PNG is created such that the least used-areas of the screen are fully white (since burned-in areas are darker, all other areas of the screen must be lightened a touch).

The intention is that this PNG can be loaded via an artwork file with a low alpha (e.g, 0.1-0.2 seems to work well) and blended over the entire screen. The PNG files are saved in the snap directory under the gamename/burnin-<screen.name>.png. The default is OFF (*-noburnin*).

Core performance options

-[no]autoframeskip / -[no]afs

Automatically determines the frameskip level while you're playing the game, adjusting it constantly in a frantic attempt to keep the game running at full speed. Turning this on overrides the value you have set for -frameskip below. The default is OFF (*-noautoframeskip*).

-frameskip / -fs <*level*>

Specifies the frameskip value. This is the number of frames out of every 12 to drop when running. For example, if you say -frameskip 2, then MAME will display 10 out of every 12 frames. By skipping those frames, you may be able to get full speed in a game that requires more horsepower than your computer has. The default value is **-frameskip 0**, which skips no frames.

-seconds_to_run / -str <*seconds*>

This option can be used for benchmarking and automated testing. It tells MAME to stop execution after a fixed number of seconds. By combining this with a fixed set of other command line options, you can set up a consistent environment for benchmarking MAME performance. In addition, upon exit, the **-str** option will write a screenshot called *final.png* to the game's snapshot directory.

-[no]throttle

Configures the default thottling setting. When throttling is on, MAME attempts to keep the game running at the game's intended speed. When throttling is off, MAME runs the game as fast as it can. Note that the fastest speed is more often than not limited by your graphics card, especially for older games. The default is ON (*-throttle*).

-[no]sleep

Allows MAME to give time back to the system when running with -throttle. This allows other programs to have some CPU time, assuming that the game isn't taxing 100% of your CPU resources. This option can potentially cause hiccups in performance if other demanding programs are running. The default is ON (*-sleep*).

-speed <*factor*>

Changes the way MAME throttles gameplay such that the game runs at some multiplier of the original speed. A <factor> of 1.0 means to run the game at its normal speed. A <factor> of 0.5 means run at half speed, and a <factor> of 2.0 means run at 2x speed. Note that changing this value affects sound playback as well, which will scale in pitch accordingly. The internal resolution of the fraction is two decimalplaces, so a value of 1.002 is the same as 1.0. The default is 1.0.

-[no]refreshspeed / -[no]rs

Allows MAME to dynamically adjust the gameplay speed such that it does not exceed the slowest refresh rate for any targeted monitors in your system. Thus, if you have a 60Hz monitor and run a game that is actually designed to run at 60.6Hz, MAME will dynamically change the speed down to 99% in order to prevent sound hiccups or other undesirable side effects of running at a slower refresh rate. The default is OFF (*-norefreshspeed*).

Core rotation options

-[no]rotate

Rotate the game to match its normal state (horizontal/vertical). This ensures that both vertically and horizontally oriented games show up correctly without the need to rotate your monitor. If you want to keep the game displaying 'raw' on the screen the way it would have in the arcade, turn this option OFF. The default is ON (*-rotate*).

-[no]ror
-[no]rol

Rotate the game screen to the right (clockwise) or left (counter-clockwise) relative to either its normal state (if **-rotate** is specified) or its native state (if **-norotate** is specified). The default for both of these options is OFF (*-noror -norol*).

-[no]autoror
-[no]autorol

These options are designed for use with pivoting screens that only pivot in a single direction. If your screen only pivots clockwise, use -autorol to ensure that the game will fill the screen either horizontally or vertically in one of the directions you can handle. If your screen only pivots counter-clockwise, use **-autoror**.

-[no]flipx
-[no]flipy

Flip (mirror) the game screen either horizontally (-flipx) or vertically (-flipy). The flips are applied after the -rotate and -ror/-rol options are applied. The default for both of these options is OFF (*-noflipx -noflipy*).

Core artwork options

-[no]artwork_crop / -[no]artcrop

Enable cropping of artwork to the game screen area only. This works best with -video gdi or -video d3d, and means that vertically oriented games running full screen can display their artwork to the left and right sides of the screen. This option can also be controlled via the Video Options menu in the user interface. The default is OFF (*-noartwork_crop*).

-[no]use_backdrops / -[no]backdrop

Enables/disables the display of backdrops. The default is ON (*-use_backdrops*).

-[no]use_overlays / -[no]overlay

Enables/disables the display of overlays. The default is ON (*-use_overlays*).

-[no]use_bezels / -[no]bezels

Enables/disables the display of bezels. The default is ON (*-use_bezels*).

-[no]use_cpanels / -[no]cpanels

Enables/disables the display of control panels. The default is ON (*-use_cpanels*).

-[no]use_marquees / -[no]marquees

Enables/disables the display of marquees. The default is ON (*-use_marquees*).

Core screen options

-brightness <*value*>

Controls the default brightness, or black level, of the game screens. This option does not affect the artwork or other parts of the display. Using the MAME UI, you can individually set the brightness for each game screen; this option controls the initial value for all visible game screens. The standard value is 1.0. Selecting lower values (down to 0.1) will produce a darkened display, while selecting higher values (up to 2.0) will give a brighter display. The default is *1.0*.

-contrast <*value*>

Controls the contrast, or white level, of the game screens. This option does not affect the artwork or other parts of the display. Using the MAME UI, you can individually set the contrast for each game screen; this option controls the initial value for all visible game screens. The standard value is 1.0. Selecting lower values (down to 0.1) will produce a dimmer display, while selecting higher values (up to 2.0) will give a more saturated display. The default is *1.0*.

-gamma <*value*>

Controls the gamma, which produces a potentially nonlinear black to white ramp, for the game screens. This option does not affect the artwork or other parts of the display. Using the MAME UI, you can individually set the gamma for each game screen; this option controls the initial value for all visible game screens. The standard value is 1.0, which gives a linear ramp from black to white. Selecting lower values (down to 0.1) will increase the nonlinearity toward black, while selecting higher values (up to 3.0) will push the nonlinearity toward white. The default is *1.0*.

-pause_brightness <*value*>

This controls the brightness level when MAME is paused. The default value is *0.65*.

-effect <*filename*>

Specifies a single PNG file that is used as an overlay over any game screens in the video display. This PNG file is assumed to live in the root of one of the artpath directories. The pattern in the PNG file is repeated both horizontally and vertically to cover the entire game screen areas (but not any external artwork), and is rendered at the target resolution of the game image. For -video gdi and -video d3d modes, this means that one pixel in the PNG will map to one pixel on your output display. The RGB values of each pixel in the PNG are multiplied against the RGB values of the target screen. The default is '*none*', meaning no effect.

Core vector options

-[no]antialias / -[no]aa

Enables antialiased line rendering for vector games. The default is ON (*-antialias*).

-beam <*width*>

Sets the width of the vectors. This is a scaling factor against the standard vector width. A value of 1.0 will keep the default vector line width. Smaller values will reduce the width, and larger values will increase the width. The default is *1.0*.

-flicker <*value*>

Simulates a vector "flicker" effect, similar to a vector monitor that needs adjustment. This option requires a float argument in the range of 0.00 - 100.00 (0=none, 100=maximum). The default is *0*.

Core sound options

-samplerate <*value*> / **-sr** <*value*>

Sets the audio sample rate. Smaller values (e.g. 11025) cause lower audio quality but faster emulation speed. Higher values (e.g. 48000) cause higher audio quality but slower emulation speed. The default is *48000*.

-[no]samples

Use samples if available. The default is ON (*-samples*).

-volume / -vol <*value*>

Sets the startup volume. It can later be changed with the user interface (see Keys section). The volume is an attenuation in dB: e.g., "**-volume -12**" will start with -12dB attenuation. The default is *0*.

Core input options

-[no]coin_lockout / -[no]coinlock

Enables simulation of the "coin lockout" feature that is implemented on a number of game PCBs. It was up to the operator whether or not the coin lockout outputs were actually connected to the coin mechanisms. If this feature is enabled, then attempts to enter a coin while the lockout is active will fail and will display a popup message in the user interface (In debug mode). If this feature is disabled, the coin lockout signal will be ignored. The default is ON (*-coin_lockout*).

-ctrlr <*controller*>

Enables support for special controllers. Configuration files are loaded from the ctrlrpath. They are in the same format as the .cfg files that are saved, but only control configuration data is read from the file. The default is NULL (no controller file).

-[no]mouse

Controls whether or not MAME makes use of mouse controllers. When this is enabled, you will likely be unable to use your mouse for other purposes until you exit or pause the game. The default is OFF (*-nomouse*).

-[no]joystick / -[no]joy

Controls whether or not MAME makes use of joystick/gamepad controllers. When this is enabled, MAME will ask DirectInput about which controllers are connected. The default is OFF (*-nojoystick*).

-[no]lightgun / -[no]gun

Controls whether or not MAME makes use of lightgun controllers. Note that most lightguns map to the mouse, so using -lightgun and -mouse together may produce strange results. The default is OFF (*-nolightgun*).

-[no]multikeyboard / -[no]multikey

Determines whether MAME differentiates between multiple keyboards. Some systems may report more than one keyboard; by default, the data from all of these keyboards is combined so that it looks like a single keyboard. Turning this option on will enable MAME to report keypresses on different keyboards independently. The default is OFF (*-nomultikeyboard*).

-[no]multimouse

Determines whether MAME differentiates between multiple mice. Some systems may report more than one mouse device; by default, the data from all of these mice is combined so that it looks like a single mouse. Turning this option on will enable MAME to report mouse movement and button presses on different mice independently. The default is OFF (*-nomultimouse*).

-[no]steadykey / -[no]steady

Some games require two or more buttons to be pressed at exactly the same time to make special moves. Due to limitations in the keyboard hardware, it can be difficult or even impossible to accomplish that using the standard keyboard handling. This option selects a different handling that makes it easier to register simultaneous button presses, but has the disadvantage of making controls less responsive. The default is OFF (*-nosteadykey*)

-[no]ui_active

Enable user interface on top of emulated keyboard (if present). The default is OFF (*-noui_active*)

-[no]offscreen_reload / -[no]reload

Controls whether or not MAME treats a second button input from a lightgun as a reload signal. In this case, MAME will report the gun's position as (0,MAX) with the trigger held, which is equivalent to an offscreen reload. This is only needed for games that required you to shoot offscreen to reload, and then only if your gun does not support off screen reloads. The default is OFF (*-nooffscreen_reload*).

-joystick_map <*map*> / **-joymap** <*map*>

Controls how joystick values map to digital joystick controls. MAME accepts all joystick input from the system as analog data. For true analog joysticks, this needs to be mapped down to the usual 4-way or 8-way digital joystick values. To do this, MAME divides the analog range into a 9x9 grid. It then takes the joystick axis position (for X and Y axes only), maps it to this grid, and then looks up a translation from a joystick map. This parameter allows you to specify the map. The default is 'auto', which means that a standard 8-way, 4-way, or 4-way diagonal map is selected automatically based on the input port configuration of the current game.

Maps are defined as a string of numbers and characters. Since the grid is 9x9, there are a total of 81 characters necessary to define a complete map. Below is an example map for an 8-way joystick:

777888999 777888999 777888999 444555666 444555666 444555666 111222333 111222333 111222333	Note that the numeric digits correspond to the keys on a numeric keypad. So '7' maps to up+left, '4' maps to left, '5' maps to neutral, etc. In addition to the numeric values, you can specify the character 's', which means "sticky". In this case, the value of the map is the same as it was the last time a non-sticky value was read.

To specify the map for this parameter, you can specify a string of rows separated by a '.' (which indicates the end of a row), like so:

```
777888999.777888999.777888999.444555666.444555666.444555666.111222333.111222333.111222333
```

However, this can be reduced using several shorthands supported by the <map> parameter. If information about a row is missing, then it is assumed that any missing data in columns 5-9 are left/right symmetric with data in columns 0-4; and any missing data in colums 0-4 is assumed to be copies of the previous data. The same logic applies to missing rows, except that up/down symmetry is assumed.

By using these shorthands, the 81 character map can be simply specified by this 11 character string: 7778...4445

Looking at the first row, 7778 is only 4 characters long. The 5th entry can't use symmetry, so it is assumed to be equal to the previous character '8'. The 6th character is left/right symmetric with the 4th character, giving an '8'. The 7th character is left/right symmetric with the 3rd character, giving a '9' (which is '7' with left/right flipped). Eventually this gives the full 777888999 string of the row.

The second and third rows are missing, so they are assumed to be identical to the first row. The

fourth row decodes similarly to the first row, producing 444555666. The fifth row is missing so it is assumed to be the same as the fourth.

The remaining three rows are also missing, so they are assumed to be the up/down mirrors of the first three rows, giving three final rows of 111222333.

-joystick_deadzone <*value*> / **-joy_deadzone** <*value*> / **-jdz** <*value*>

If you play with an analog joystick, the center can drift a little. joystick_deadzone tells how far along an axis you must move before the axis starts to change. This option expects a float in the range of 0.0 to 1.0. Where 0 is the center of the joystick and 1 is the outer limit. The default is *0.3*.

-joystick_saturation <*value*> / **joy_saturation** <*value*> / **-jsat** <*value*>

If you play with an analog joystick, the ends can drift a little, and may not match in the +/- directions. joystick_saturation tells how far along an axis movement change will be accepted before it reaches the maximum range. This option expects a float in the range of 0.0 to 1.0, where 0 is the center of the joystick and 1 is the outer limit. The default is *0.85*.

-natural

Allows user to specify whether or not to use a natural keyboard or not. This allows you to start your game or system in a 'native' mode, depending on your region, allowing compatability for non-"QWERTY" style keyboards. The default is OFF (*-nonatural*)

-joystick_contradictory

Enable contradictory direction digital joystick input at the same time such as **Left and Right** or **Up and Down** at the same time. The default is OFF (*-nojoystick_contradictory*)

-coin_impulse *[n]*

Set coin impulse time based on n (n<0 disable impulse, n==0 obey driver, 0<n set time n). Default is *0*.

Core input automatic enable options

-paddle_device enable (none|keyboard|mouse|lightgun|joystick) if a paddle control is present

-adstick_device enable (none|keyboard|mouse|lightgun|joystick) if an analog joystick control is present

-pedal_device enable (none|keyboard|mouse|lightgun|joystick) if a pedal control is present

-dial_device enable (none|keyboard|mouse|lightgun|joystick) if a dial control is present

-trackball_device enable (none|keyboard|mouse|lightgun|joystick) if a trackball control is present

-lightgun_device enable (none|keyboard|mouse|lightgun|joystick) if a lightgun control is present

-positional_device enable (none|keyboard|mouse|lightgun|joystick) if a positional control is present

-mouse_device enable (none|keyboard|mouse|lightgun|joystick) if a mouse control is present

Each of these options controls autoenabling the mouse, joystick, or lightgun depending on the presence of a particular class of analog control for a particular game. For example, if you specify the option -paddle mouse, then any game that has a paddle control will automatically enable mouse controls just as if you had explicitly specified -mouse. Note that these controls override the values of -[no]mouse, -[no]joystick, etc.

Debugging options

-[no]verbose / -[no]v

Displays internal diagnostic information. This information is very useful for debugging problems with your configuration. IMPORTANT: when reporting bugs, please run with **mame -verbose** and include the resulting information. The default is OFF (*-noverbose*).

-[no]oslog

Output error.log data to the system debugger. The default is OFF (*-nooslog*).

-[no]log

Creates a file called error.log which contains all of the internal log messages generated by the MAME core and game drivers. The default is OFF (*-nolog*).

-[no]debug

Activates the integrated debugger. By default, the debugger is entered by pressing the tilde (~) key during emulation. It is also entered immediately at startup. The default is OFF (*-nodebug*).

-debugscript <*filename*>

Specifies a file that contains a list of debugger commands to execute immediately upon startup. The default is NULL (*no commands*).

-[no]update_in_pause

Enables updating of the main screen bitmap while the game is paused. This means that the VIDEO_UPDATE callback will be called repeatedly during pause, which can be useful for debugging. The default is OFF (*-noupdate_in_pause*).

Core communication options

-comm_localhost <*string*>

Local address to bind to. This can be a traditional xxx.xxx.xxx.xxx address or a string containing a resolvable hostname. The default is value is "*0.0.0.0*"

-comm_localport <*string*>

Local port to bind to. This can be any traditional communications port as an unsigned 16-bit integer (0-65535). The default value is "*15122*".

-comm_remotehost <*string*>

Remote address to connect to. This can be a traditional xxx.xxx.xxx.xxx address or a string containing a resolvable hostname. The default is value is "*0.0.0.0*"

-comm_remoteport <*string*>

Remote port to connect to. This can be any traditional communications port as an unsigned 16-bit integer (0-65535). The default value is "*15122*".

Core misc options

-[no]drc Enable DRC cpu core if available. The default is ON (*-drc*).

-drc_use_c

Force DRC use the C code backend. The default is OFF (*-nodrc_use_c*).

-drc_log_uml

Write DRC UML disassembly log. The default is OFF (*-nodrc_log_uml*).

-drc_log_native

write DRC native disassembly log. The default is OFF (*-nodrc_log_native*).

-bios <*biosname*>

Specifies the specific BIOS to use with the current game, for game systems that make use of a BIOS. The **-listxml** output will list all of the possible BIOS names for a game. The default is '*default*'.

-[no]cheat / -[no]c

Activates the cheat menu with autofire options and other tricks from the cheat database, if present. The default is OFF (*-nocheat*).

-[no]skip_gameinfo

Forces MAME to skip displaying the game info screen. The default is OFF (*-noskip_gameinfo*).

-uifont <*fontname*>

Specifies the name of a font file to use for the UI font. If this font cannot be found or cannot be loaded, the system will fall back to its built-in UI font. On some platforms 'fontname' can be a system font name (TTF) instead of a (BDF) font file. The default is '*default*' (use the OSD-determined default font).

-ramsize *[n]*

Allows you to change the default RAM size (if supported by driver).

-confirm_quit

Display a Confirm Quit dialong to screen on exit, requiring one extra step to exit MAME. The default is OFF (*-noconfirm_quit*).

-ui_mouse

Displays a mouse cursor when using the built-in UI for MAME. The default is (*-noui_mouse*).

-autoboot_command *"<command>"*

Command string to execute after machine boot (in quotes " "). To issue a quote to the emulation, use """ in the string. Using **\n** will issue a create a new line, issuing what was typed prior as a command.

Example: -autoboot_command "load """"$""",8,1\n"

-autoboot_delay *[n]*

Timer delay (in seconds) to trigger command execution on autoboot.

-autoboot_script / -script *[filename.lua]*

File containing scripting to execute after machine boot.

-language <*language*>

Specify a localization language found in the *languagepath* tree.

Windows-Specific Commandline Options

This section contains configuration options that are specific to the native (non-SDL) Windows version of MAME.

Debugging options

-[no]oslog

Outputs the error.log data to the Windows debugger. This can be used at the same time as -log to output the log data to both targets as well. Default is OFF (*-nooslog*).

-watchdog <*duration*> / **-wdog** <*duration*>

Enables an internal watchdog timer that will automatically kill the MAME process if more than <*duration*> seconds passes without a frame update. Keep in mind that some games sit for a while during load time without updating the screen, so <*duration*> should be long enough to cover that. 10-30 seconds on a modern system should be plenty in general. By default there is no watchdog.

-debugger_font <*fontname*> / **-dfont** <*fontname*>

Specifies the name of the font to use for debugger windows. By default, the font is Lucida Console.

-debugger_font_size <*points*> / **-dfontsize** <*points*>

Specifies the size of the font to use for debugger windows, in points. By default, this is set to 9pt.

Performance options

-priority <*priority*>

Sets the thread priority for the MAME threads. By default the priority is left alone to guarantee proper cooperation with other applications. The valid range is -15 to 1, with 1 being the highest priority. The default is *0* (*NORMAL priority*).

-numprocessors <*auto|value*> / **-np** <*auto|value*>

Specify the number of processors to use for work queues. Specifying "*auto*" will use the value reported by the system or environment variable **OSDPROCESSORS**. To avoid abuse, this value is internally limited to 4 times the number of processors reported by the system. The default is "*auto*".

-profile *[n]*

Enables profiling, specifying the stack depth of *[n]* to track.

-bench *[n]*

Benchmark for *[n]* number of emulated seconds; implies the command string:

-str [n] -video none -sound none -nothrottle

Video options

-video <*bgfx|gdi|d3d|none*>

Specifies which video subsystem to use for drawing. Using '*bgfx*' specifies the new hardware accelerated renderer. By specifying '*gdi*' here, you tell MAME to render video using older standard Windows graphics drawing calls. This is the slowest but most compatible option on older versions of Windows. Specifying '*d3d*' tells MAME to use Direct3D for rendering. This produces the highest quality output and enables all rendering options. It is recommended if you have a semi-recent (2002+) video card or onboard Intel video of the HD3000 line or better. The final option 'none' displays no windows and does no drawing. This is primarily present for doing CPU benchmarks without the overhead of the video system. The default is *d3d*.

-numscreens <*count*>

Tells MAME how many output windows to create. For most games, a single output window is all you need, but some games originally used multiple screens (*e.g. Darius, PlayChoice-10*). Each screen (up to 4) has its own independent settings for physical monitor, aspect ratio, resolution, and view, which can be set using the options below. The default is *1*.

-[no]window / -[no]w

Run MAME in either a window or full screen. The default is OFF (*-nowindow*).

-[no]maximize / -[no]max

Controls initial window size in windowed mode. If it is set on, the window will initially be set to the maximum supported size when you start MAME. If it is turned off, the window will start out at the smallest supported size. This option only has an effect when the -window option is used. The default is ON (*-maximize*).

-[no]keepaspect / -[no]ka

Enables aspect ratio enforcement. When this option is on, the game's proper aspect ratio (generally 4:3 or 3:4) is enforced, so you get the game looking like it should. When running in a window with this option on, you can only resize the window to the proper aspect ratio, unless you are holding down the CONTROL key. By turning the option off, the aspect ratio is allowed to float. In full screen mode, this means that all games will stretch to the full screen size (even vertical games). In window mode, it means that you can freely resize the window without any constraints. The default is ON (*-keepaspect*).

The MAME team heavily suggests you leave this at default. Stretching games beyond their original aspect ratio will mangle the appearance of the game in ways that no filtering or HLSL can repair.

-prescale <*amount*>

Controls the size of the screen images when they are passed off to the graphics system for scaling. At the minimum setting of 1, the screen is rendered at its original resolution before being scaled. At higher settings, the screen is expanded by a factor of <*amount*> before being scaled. With **-video d3d**, this produces a less blurry image at the expense of some speed. The default is *1*.

-[no]waitvsync

Waits for the refresh period on your computer's monitor to finish before starting to draw video to your screen. If this option is off, MAME will just draw to the screen at any old time, even in the middle of a refresh cycle. This can cause "tearing" artifacts, where the top portion of the screen is out of sync with the bottom portion. Tearing is not noticeable on all games, and some people hate it more than others. However, if you turn this option on, you will waste more of your CPU cycles waiting for the proper time to draw, so you will see a performance hit. You should only need to turn this on in windowed mode. In full screen mode, it is only needed if **-triplebuffer** does not remove the tearing, in which case you should use **-notriplebuffer -waitvsync**. Note that this option does not work with **-video gdi** mode. The default is OFF (*-nowaitvsync*).

-[no]syncrefresh

Enables speed throttling only to the refresh of your monitor. This means that the game's actual refresh rate is ignored; however, the sound code still attempts to keep up with the game's original refresh rate, so you may encounter sound problems. This option is intended mainly for those who have tweaked their video card's settings to provide carefully matched refresh rate options. Note that this option does not work with -video gdi mode. The default is OFF (*-nosyncrefresh*).

Direct3D-specific options

-[no]filter / -[no]d3dfilter / -[no]flt

Enable bilinear filtering on the game screen graphics. When disabled, point filtering is applied, which is crisper but leads to scaling artifacts. If you don't like the filtered look, you are probably better off increasing the *-prescale* value rather than turning off filtering altogether. The default is ON (*-filter*).

Per-window options

-screen <*display*>
-screen0 <*display*>
-screen1 <*display*>
-screen2 <*display*>
-screen3 <*display*>

Specifies which physical monitor on your system you wish to have each window use by default. In order to use multiple windows, you must have increased the value of the **-numscreens** option. The name of each display in your system can be determined by running MAME with the -verbose option. The display names are typically in the format of: \\.*DISPLAYn*, where 'n' is a number from 1 to the number of connected monitors. The default value for these options is '*auto*', which means that the first window is placed on the first display, the second window on the second display, etc.

The **-screen0**, **-screen1**, **-screen2**, **-screen3** parameters apply to the specific window. The **-screen** parameter applies to all windows. The window-specific options override values from the all window option.

-aspect <*width:height*> / **-screen_aspect** <*num:den*>
-aspect0 <*width:height*>
-aspect1 <*width:height*>
-aspect2 <*width:height*>
-aspect3 <*width:height*>

Specifies the physical aspect ratio of the physical monitor for each window. In order to use multiple windows, you must have increased the value of the **-numscreens** option. The physical aspect ratio can be determined by measuring the width and height of the visible screen image and specifying them separated by a colon. The default value for these options is '*auto*', which means that MAME assumes the aspect ratio is proportional to the number of pixels in the desktop video mode for each monitor.

The **-aspect0**, **-aspect1**, **-aspect2**, **-aspect3** parameters apply to the specific window. The **-aspect** parameter applies to all windows. The window-specific options override values from the all window option.

-resolution <*widthxheight[@refresh]*> / **-r** <*widthxheight[@refresh]*>
-resolution0 <*widthxheight[@refresh]*> / **-r0** <*widthxheight[@refresh]*>
-resolution1 <*widthxheight[@refresh]*> / **-r1** <*widthxheight[@refresh]*>
-resolution2 <*widthxheight[@refresh]*> / **-r2** <*widthxheight[@refresh]*>
-resolution3 <*widthxheight[@refresh]*> / **-r3** <*widthxheight[@refresh]*>

Specifies an exact resolution to run in. In full screen mode, MAME will try to use the specific resolution you request. The width and height are required; the refresh rate is optional. If omitted or set to 0, MAME will determine the mode automatically. For example, **-resolution 640x480** will force 640x480 resolution, but MAME is free to choose the refresh rate. Similarly, **-resolution 0x0@60** will force a 60Hz refresh rate, but allows MAME to choose the resolution. The string "*auto*" is also supported, and is equivalent to *0x0@0*. In window mode, this resolution is used as a maximum size for the window. This option requires the **-switchres** option as well in order to actually enable resolution switching with **-video d3d**. The default value for these options is '*auto*'.

The **-resolution0**, **-resolution1**, **-resolution2**, **-resolution3** parameters apply to the specific window. The -resolution parameter applies to all windows. The window-specific options override values from the all window option.

-view <*viewname*>
-view0 <*viewname*>
-view1 <*viewname*>
-view2 <*viewname*>
-view3 <*viewname*>

Specifies the initial view setting for each window. The <*viewname*> does not need to be a perfect match; rather, it will select the first view whose name matches all the characters specified by <*viewname*>. For example, **-view native** will match the "*Native (15:14)*" view even though it is not a perfect match. The value '*auto*' is also supported, and requests that MAME perform a default selection. The default value for these options is '*auto*'.

The **-view0**, **-view1**, **-view2**, **-view3** parameters apply to the specific window. The **-view** parameter applies to all windows. The window-specific options override values from the all window option.

Full screen options

-[no]triplebuffer / -[no]tb

Enables or disables "triple buffering". Normally, MAME just draws directly to the screen, without any fancy buffering. But with this option enabled, MAME creates three buffers to draw to, and cycles between them in order. It attempts to keep things flowing such that one buffer is currently displayed, the second buffer is waiting to be displayed, and the third buffer is being drawn to. **-triplebuffer** will override **-waitvsync**, if the buffer is successfully created. This option does not work with **-video gdi**. The default is OFF (*-notriplebuffer*).

-[no]switchres

Enables resolution switching. This option is required for the **-resolution*** options to switch resolutions in full screen mode. On modern video cards, there is little reason to switch resolutions unless you are trying to achieve the "exact" pixel resolutions of the original games, which requires significant tweaking. This option is also useful on LCD displays, since they run with a fixed resolution and switching resolutions on them is just silly. This option does not work with **-video gdi**. The default is OFF (*-noswitchres*).

-full_screen_brightness <*value*> / **-fsb** <*value*>

Controls the brightness, or black level, of the entire display. The standard value is 1.0. Selecting lower values (down to 0.1) will produce a darkened display, while selecting higher values (up to 2.0) will give a brighter display. Note that not all video cards have hardware to support this option. This option does not work with **-video gdi**. The default is *1.0*.

-full_screen_contrast <*value*> / **-fsc** <*value*>

Controls the contrast, or white level, of the entire display. The standard value is 1.0. Selecting lower values (down to 0.1) will produce a dimmer display, while selecting higher values (up to 2.0) will give a more saturated display. Note that not all video cards have hardware to support this option. This option does not work with **-video gdi**. The default is *1.0*.

-full_screen_gamma <*value*> / **-fsg** <*value*>

Controls the gamma, which produces a potentially nonlinear black to white ramp, for the entire display. The standard value is 1.0, which gives a linear ramp from black to white. Selecting lower values (down to 0.1) will increase the nonlinearity toward black, while selecting higher values (up to 3.0) will push the nonlinearity toward white. Note that not all video cards have hardware to support this option. This option does not work with **-video gdi**. The default is *1.0*.

Sound options

-sound <*dsound|sdl|none*>

Specifies which sound subsystem to use. '*none*' disables sound altogether. The default is *dsound*.

-audio_latency <*value*>

This controls the amount of latency built into the audio streaming. By default MAME tries to keep the DirectSound audio buffer between 1/5 and 2/5 full. On some systems, this is pushing it too close to the edge, and you get poor sound sometimes. The latency parameter controls the lower threshold. The default is *1* (meaning lower=1/5 and upper=2/5). Set it to 2 (**-audio_latency 2**) to keep the sound buffer between 2/5 and 3/5 full. If you crank it up to 4, you can *definitely* notice audio lag.

Input device options

-[no]dual_lightgun / -[no]dual

Controls whether or not MAME attempts to track two lightguns connected at the same time. This option requires -lightgun. This option is a hack for supporting certain older dual lightgun setups. If you have multiple lightguns connected, you will probably just need to enable -mouse and configure each lightgun independently. The default is *OFF* (*-nodual_lightgun*).

SDL-Specific Commandline Options

This section contains configuration options that are specific to any build supported by SDL (including Windows where compiled as SDL instead of native).

In addition to the keys described in config.txt, the following additional keys are defined for SDL-specific versions of MAME:

Debugging options

-[no]oslog

Outputs the error.log data to the stderr TTY channel (usually the command line window MAME was started in). This can be used at the same time as *-log* to output the log data to both targets as well. Default is OFF (*-nooslog*).

-watchdog <*duration*> / **-wdog** <*duration*>

Enables an internal watchdog timer that will automatically kill the MAME process if more than <*duration*> seconds passes without a frame update. Keep in mind that some games sit for a while during load time without updating the screen, so <*duration*> should be long enough to cover that. 10-30 seconds on a modern system should be plenty in general. By default there is no watchdog.

Performance options

-numprocessors <*auto|value*> / **-np** <*auto|value*>

Specify the number of processors to use for work queues. Specifying "*auto*" will use the value reported by the system or environment variable **OSDPROCESSORS**. To avoid abuse, this value is internally limited to 4 times the number of processors reported by the system. The default is "*auto*".

-sdlvideofps

Enable output of benchmark data on the SDL video subsystem, including your system's video driver, X server (if applicable), and OpenGL stack in **-video opengl** mode.

-bench *[n]*

Benchmark for [n] number of emulated seconds; implies the command string: **-str [n] -video none -sound none -nothrottle**. Default is OFF (*-nobench*)

Video options

-video <*soft|opengl|none*>

Specifies which video subsystem to use for drawing. The default for Mac OS X is '*opengl*' because OS X is guaranteed to have a compliant OpenGL stack. The default on all other systems is '*soft*'.

-numscreens <*count*>

Tells MAME how many output windows to create. For most games, a single output window is all you need, but some games originally used multiple screens. Each screen (up to 4) has its own independent settings for physical monitor, aspect ratio, resolution, and view, which can be set using the options below. The default is 1.

-[no]window / -[no]w

Run MAME in either a window or full screen. The default is OFF (*-nowindow*).

-[no]maximize / -[no]max

Controls initial window size in windowed mode. If it is set on, the window will initially be set to the maximum supported size when you start MAME. If it is turned off, the window will start out at the smallest supported size. This option only has an effect when the **-window** option is used. The default is ON (*-maximize*).

-[no]keepaspect / -[no]ka

Enables aspect ratio enforcement. When this option is on, the game's proper aspect ratio (generally 4:3 or 3:4) is enforced, so you get the game looking like it should. When running in a window with this option on, you can only resize the window to the proper aspect ratio, unless you are holding down the CONTROL key. By turning the option off, the aspect ratio is allowed to float. In full screen mode, this means that all games will stretch to the full screen size (even vertical games). In window mode, it means that you can freely resize the window without any constraints. The default is ON (*-keepaspect*).

-[no]unevenstretch

Allow non-integer stretch factors allowing for great window sizing flexability. The default is ON. (*-unevenstretch*)

-[no]centerh

Center horizontally within the view area. Default is ON (*-centerh*).

-[no]centerv

Center vertically within the view area. Default is ON (*-centerv*).

-[no]waitvsync

Waits for the refresh period on your computer's monitor to finish before starting to draw video to your screen. If this option is off, MAME will just draw to the screen at any old time, even in the middle of a refresh cycle. This can cause "tearing" artifacts, where the top portion of the screen is out of sync with the bottom portion. Tearing is not noticeable on all games, and some people hate it more than others. However, if you turn this option on, you will waste more of your CPU cycles waiting for the proper time to draw, so you will see a performance hit. You should only need to turn this on in windowed mode. In full screen mode, it is only needed if **-triplebuffer** does not remove the tearing, in which case you should use **-notriplebuffer -waitvsync.** Note that support for this option depends entirely on your operating system and video drivers; in general it will not work in windowed mode so **-video opengl** and fullscreen give the greatest chance of success. The default is OFF (*-nowaitvsync*).

-[no]syncrefresh

Enables speed throttling only to the refresh of your monitor. This means that the game's actual refresh rate is ignored; however, the sound code still attempts to keep up with the game's original refresh rate, so you may encounter sound problems. This option is intended mainly for those who have tweaked their video card's settings to provide carefully matched refresh rate options. The default is OFF (*-nosyncrefresh*).

Video soft-specific options

-scalemode

Scale mode: none, async, yv12, yuy2, yv12x2, yuy2x2 (**-video soft** only). Default is '*none*'.

Video OpenGL-specific options

-[no]filter / -[no]flt

Enable bilinear filtering on the game screen graphics. When disabled, point filtering is applied, which is crisper but leads to scaling artifacts. If you don't like the filtered look, you are probably better off increasing the **-prescale** value rather than turning off filtering altogether. The default is ON (*-filter*).

-prescale <*amount*>

Controls the size of the screen images when they are passed off to the graphics system for scaling. At the minimum setting of 1, the screen is rendered at its original resolution before being scaled. At higher settings, the screen is expanded by a factor of <*amount*> before being scaled. This produces a less blurry

image at the expense of some speed and also increases the effective resolution of non-screen elements such as artwork and fonts. The default is *1*.

Video OpenGL debugging options

These 4 options are for compatibility in **-video opengl**. If you report rendering artifacts you may be asked to try messing with them by the devs, but normally they should be left at their defaults which results in the best possible video performance.

-[no]gl_forcepow2texture

Always use only power-of-2 sized textures (default *off*)

-[no]gl_notexturerect

Don't use OpenGL GL_ARB_texture_rectangle (default *on*)

-[no]gl_vbo

Enable OpenGL VBO, if available (default *on*)

-[no]gl_pbo

Enable OpenGL PBO, if available (default *on*)

Video OpenGL GLSL options

-gl_glsl

Enable OpenGL GLSL, if available (default *off*)

-gl_glsl_filter

Enable OpenGL GLSL filtering instead of FF filtering – *0-plain, 1-bilinear* (default is *1*)

-glsl_shader_mame0

Custom OpenGL GLSL shader set MAME bitmap 0 [todo: better details on usage at some point. See http://forums.bannister.org/ubbthreads.php?ubb=showflat&Number=100988#Post100988]

-glsl_shader_mame1

Custom OpenGL GLSL shader set MAME bitmap 1

-glsl_shader_mame2

Custom OpenGL GLSL shader set MAME bitmap 2

-glsl_shader_mame3

Custom OpenGL GLSL shader set MAME bitmap 3

-glsl_shader_mame4

Custom OpenGL GLSL shader set MAME bitmap 4

-glsl_shader_mame5

Custom OpenGL GLSL shader set MAME bitmap 5

-glsl_shader_mame6

Custom OpenGL GLSL shader set MAME bitmap 6

-glsl_shader_mame7

Custom OpenGL GLSL shader set MAME bitmap 7

-glsl_shader_mame8

Custom OpenGL GLSL shader set MAME bitmap 8

-glsl_shader_mame9

Custom OpenGL GLSL shader set MAME bitmap 9

-glsl_shader_screen0

Custom OpenGL GLSL shader screen bitmap 0

-glsl_shader_screen1

Custom OpenGL GLSL shader screen bitmap 1

-glsl_shader_screen2

Custom OpenGL GLSL shader screen bitmap 2

-glsl_shader_screen3

Custom OpenGL GLSL shader screen bitmap 3

-glsl_shader_screen4

Custom OpenGL GLSL shader screen bitmap 4

-glsl_shader_screen5

Custom OpenGL GLSL shader screen bitmap 5

-glsl_shader_screen6

Custom OpenGL GLSL shader screen bitmap 6

-glsl_shader_screen7

Custom OpenGL GLSL shader screen bitmap 7

-glsl_shader_screen8

Custom OpenGL GLSL shader screen bitmap 8

-glsl_shader_screen9

Custom OpenGL GLSL shader screen bitmap 9

-gl_glsl_vid_attr

Enable OpenGL GLSL handling of brightness and contrast. Better RGB game performance. Default is *on*.

Per-window options

NOTE: **Multiple Screens may fail to work correctly on some Macintosh machines as of right now.**

-screen <*display*>
-screen0 <*display*>
-screen1 <*display*>
-screen2 <*display*>
-screen3 <*display*>

Specifies which physical monitor on your system you wish to have each window use by default. In order to use multiple windows, you must have increased the value of the **-numscreens** option. The name of each display in your system can be determined by running MAME with the -verbose option. The display names are typically in the format of a number from 1 to the number of connected monitors. The default value for these options is '*auto*', which means that the first window is placed on the first display, the second window on the second display, etc.

The **-screen0**, **-screen1**, **-screen2**, **-screen3** parameters apply to the specific window. The **-screen** parameter applies to all windows. The window-specific options override values from the all window option.

-aspect <*width:height*> / **-screen_aspect** <*num:den*>
-aspect0 <*width:height*>
-aspect1 <*width:height*>
-aspect2 <*width:height*>
-aspect3 <*width:height*>

Specifies the physical aspect ratio of the physical monitor for each window. In order to use multiple windows, you must have increased the value of the **-numscreens** option. The physical aspect ratio can be determined by measuring the width and height of the visible screen image and specifying them separated by a colon. The default value for these options is '*auto*', which means that MAME assumes the aspect ratio is proportional to the number of pixels in the desktop video mode for each monitor.

The **-aspect0**, **-aspect1**, **-aspect2**, **-aspect3** parameters apply to the specific window. The **-aspect** parameter applies to all windows. The window-specific options override values from the all window option.

-resolution <*widthxheight[@refresh]*> / **-r** <*widthxheight[@refresh]*>
-resolution0 <*widthxheight[@refresh]*> / **-r0** <*widthxheight[@refresh]*>
-resolution1 <*widthxheight[@refresh]*> / **-r1** <*widthxheight[@refresh]*>
-resolution2 <*widthxheight[@refresh]*> / **-r2** <*widthxheight[@refresh]*>
-resolution3 <*widthxheight[@refresh]*> / **-r3** <*widthxheight[@refresh]*>

Specifies an exact resolution to run in. In full screen mode, MAME will try to use the specific resolution you request. The width and height are required; the refresh rate is optional. If omitted or set to 0, MAME will determine the mode automatically. For example, **-resolution 640x480** will force 640x480 resolution, but MAME is free to choose the refresh rate. Similarly, **-resolution 0x0@60** will force a 60Hz refresh rate, but allows MAME to choose the resolution. The string "*auto*" is also supported, and is equivalent to *0x0@0*. In window mode, this resolution is used as a maximum size for the window. This option requires the **-switchres** option. The default value for these options is '*auto*'.

The **-resolution0**, **-resolution1**, **-resolution2**, **-resolution3** parameters apply to the specific window. The -resolution parameter applies to all windows. The window-specific options override values from the all window option.

-view <*viewname*>
-view0 <*viewname*>
-view1 <*viewname*>
-view2 <*viewname*>
-view3 <*viewname*>

Specifies the initial view setting for each window. The <*viewname*> does not need to be a perfect match; rather, it will select the first view whose name matches all the characters specified by <*viewname*>. For example, **-view native** will match the "*Native (15:14)*" view even though it is not a perfect match. The value '*auto*' is also supported, and requests that MAME perform a default selection. The default value for these options is '*auto*'.

The **-view0**, **-view1**, **-view2**, **-view3** parameters apply to the specific window. The **-view** parameter applies to all windows. The window-specific options override values from the all window option.

Full screen options

-[no]switchres

Enables resolution switching. This option is required for the **-resolution*** options to switch resolutions in full screen mode. On modern video cards, there is little reason to switch resolutions unless you are trying to achieve the "exact" pixel resolutions of the original games, which requires significant tweaking. This option is also useful on LCD displays, since they run with a fixed resolution and switching resolutions on them is just silly. The default is OFF (*-noswitchres*).

Sound options

-sound <*sdl|none*>

Specifies which sound subsystem to use. '*none*' disables sound altogether. The default is *sdl*.

-audio_latency <*value*>

This controls the amount of latency built into the audio streaming. By default MAME tries to keep the audio buffer between 1/5 and 2/5 full. On some systems, this is pushing it too close to the edge, and you get poor sound sometimes. The latency parameter controls the lower threshold. The default is *1* (meaning lower=1/5 and upper=2/5). Set it to 2 (**-audio_latency 2**) to keep the sound buffer between 2/5 and 3/5 full. If you crank it up to 4, you can *definitely* notice audio lag.

SDL Keyboard Mapping

-keymap

Enable keymap. Default is OFF (*-nokeymap*)

-keymap_file <*file*>

Keymap Filename. Default is '*keymap.dat*'.

-uimodekey <*key*>

Key to toggle keyboard mode. Default is '*SCRLOCK*'

SDL Joystick Mapping

-joy_idx1 <*name*>

Name of joystick mapped to joystick #1, default is *auto*.

-joy_idx2 <*name*>

Name of joystick mapped to joystick #2, default is *auto*.

-joy_idx3 <*name*>

Name of joystick mapped to joystick #3, default is *auto*.

-joy_idx4 <*name*>

Name of joystick mapped to joystick #4, default is *auto*.

-joy_idx5 <*name*>

Name of joystick mapped to joystick #5, default is *auto*.

-joy_idx6 <*name*>

Name of joystick mapped to joystick #6, default is *auto*.

-joy_idx7 <*name*>

Name of joystick mapped to joystick #7, default is *auto*.

-joy_idx8 <*name*>

Name of joystick mapped to joystick #8, default is *auto*.

-sixaxis

Use special handling for PS3 Sixaxis controllers. Default is OFF (*-nosixaxis*)

SDL Lowlevel driver options

-videodriver <*driver*>

SDL video driver to use ('x11', 'directfb', ... or '*auto*' for SDL default)

-audiodriver <*driver*>

SDL audio driver to use ('alsa', 'arts', ... or '*auto*' for SDL default)

-gl_lib <*driver*>

Alternative **libGL.so** to use; '*auto*' for system default

CHAPTER

FIVE

ADVANCED CONFIGURATION

Multiple Configuration Files

MAME has a very powerful configuration file system that can allow you to tweak settings on a per-game, per-system, or even per-monitor type basis, but requires careful thought about how you arrange your configs.

Order of Config Loading

1. The command line is parsed first, and any settings passed that way *will take priority over anything in an INI file*.
2. **MAME.INI (or other platform INI; e.g. MESS.INI) is parsed twice.** The first pass may change various path settings, so the second pass is done to see if there is a valid config file at that new location (and if so, change settings using that file)
3. **DEBUG.INI if in debug mode.** This is an advanced config file, most people won't need to use it or be concerned by it.
4. **System-specific INI files where appropriate (e.g. NEOGEO_NOSLOT.INI or CPS2.INI)** As an example, Street Fighter Alpha is a CPS2 game, and so **CPS2.INI** would be loaded here.
5. **Monitor orientation INI file (either HORIZONT.INI or VERTICAL.INI)** Pac-Man, for one example, is a vertical monitor setup, so it would load **VERTICAL.INI**. Street Fighter Alpha is a horizontal game, so it loads **HORIZONT.INI**.
6. **System-type INI files (ARCADE.INI, CONSOLE.INI, COMPUTER.INI, or OTHERSYS.INI)** Both Pac-Man and Street Fighter Alpha are arcade games, so **ARCADE.INI** would be loaded here. Atari 2600 would load **CONSOLE.INI**.
7. **Screen-type INI file (VECTOR.INI for vector games, RASTER.INI for raster games, LCD.INI for LCD games)** Pac-Man and Street Fighter Alpha are raster, so **RASTER.INI** gets loaded here. Tempest is a vector monitor game, and **VECTOR.INI** would be loaded here.
8. **Source INI files.** This is an advanced config file, most people won't need to use it and it can be safely ignored. MAME will attempt to load **SOURCE/SOURCEFILE.INI** and **SOURCEFILE.INI**, where sourcefile is the actual filename of the source code file. *mame -listsource <game>* will show the source file for a given game.

   For instance, Banpresto's Sailor Moon, Atlus's Dodonpachi, and Nihon System's Dangun Feveron all share a large amount of hardware and are grouped into the CAVE.C file, meaning they all parse **source/cave.ini**
9. **Parent INI file.** For example, if running Pac-Man, which is a clone of Puck-Man, it'd be **PUCKMAN.INI**
10. Driver INI file. Using our previous example of Pac-Man, this would be **PACMAN.INI**.

Examples of Config Loading Order

1. **Alcon, which is the US clone of Slap Fight. (*mame alcon*)** Command line, MAME.INI, VERTICAL.INI, ARCADE.INI, RASTER.INI, SLAPFGHT.INI, and lastly ALCON.INI (*remember command line parameters take precedence over all else!*)
2. **Super Street Fighter 2 Turbo (*mame ssf2t*)** Command line, MAME.INI, HORIZONT.INI, ARCADE.INI, RASTER.INI, CPS2.INI, and lastly SSF2T.INI (*remember command line parameters take precedence over all else!*)

Tricks to Make Life Easier

Some users may have a wall-mounted or otherwise rotatable monitor, and may wish to actually play vertical games with the rotated display. The easiest way to accomplish this is to put your rotation modifiers into **VERTICAL.INI**, where they will only affect vertical games.

[todo: more practical examples]

MAME Path Handling

MAME has a specific order it uses when checking for user files such as ROM sets and cheat files.

Order of Path Loading

Let's use an example of the cheat file for AfterBurner 2 for Sega Genesis/MegaDrive (aburner2 in the megadrive softlist), and your cheatpath is set to "cheat" (as per the default) – this is how MAME will search for that cheat file:

1. cheat/megadriv/aburner2.xml
2. cheat/megadriv.zip -> aburner2.xml Notice that it checks for a .ZIP file first before a .7Z file.
3. cheat/megadriv.zip -> <arbitrary path>/aburner2.xml It will look for the first (if any) aburner2.xml file it can find inside that zip, no matter what the path is.
4. cheat.zip -> megadriv/aburner2.xml Now it is specifically looking for the file and folder combination, but inside the cheat.zip file.
5. cheat.zip -> <arbitrary path>/megadriv/aburner2.xml Like before, except looking for the first (if any) aburner2.xml inside a megadriv folder inside the zip.
6. cheat/megadriv.7z -> aburner2.xml Now we start checking 7ZIP files.
7. cheat/megadriv.7z -> <arbitrary path>/aburner2.xml
8. cheat.7z -> megadriv/aburner2.xml
9. cheat.7z -> <arbitrary path>/megadriv/aburner2.xml Similar to zip, except now 7ZIP files.

[todo: ROM set loading is slightly more complicated, adding CRC. Get that documented in the next day or two.]

Shifter Toggle Disable

This is an advanced feature for alternative shifter handling for certain older arcade machines such as Spy Hunter and Outrun that used a two-way toggle switch for the shifter. By default, the shifter is treated as a toggle switch. One press of the mapped control for the shifter will switch it from low to high, and another will switch it back. This may not be

ideal if you have access to a physical shifter that works identically to how the original machines did. (The input is on when in one gear, the input is off otherwise)

Note that this feature will *not* help controller users and will not help with games that have more than two shifter states (e.g. five gears in modern racing games)

This feature is not exposed through the graphical user interface in any way, as it is an extremely advanced tweak intended explicitly for people who have this specific need, have the hardware to take advantage of it, and the knowledge to use it correctly.

Disabling and Enabling Shifter Toggle

This example will use the game Spy Hunter (set *spyhunt*) to demonstrate the exact change needed:

You will need to manually edit the game .CFG file in the CFG folder (e.g. *spyhunt.cfg*)

Start by loading MAME with the game in question. In our case, that will be **mame64 spyhunt**.

Set up the controls as you would please, including mapping the shifter. Exit MAME, open the .cfg file in your text editor of choice.

Inside the *spyhunt.cfg* file, you will find the following for the input. The actual input code in the middle can and will vary depending on the controller number and what input you have mapped.

<port tag=":ssio:IP0" type="P1_BUTTON2" mask="16" defvalue="16">
    <newseq type="standard">
        JOYCODE_1_RYAXIS_NEG_SWITCH OR JOYCODE_1_RYAXIS_POS_SWITCH
    </newseq>
</port>

The line you need to edit will be the port line defining the actual input. For Spy Hunter, that's going to be *P1_BUTTON2*. Add **toggle="no"** to the end of the tag, like follows:

<port tag=":ssio:IP0" type="P1_BUTTON2" mask="16" defvalue="16" toggle="no">
    <newseq type="standard">
        JOYCODE_1_RYAXIS_NEG_SWITCH OR JOYCODE_1_RYAXIS_POS_SWITCH
    </newseq>
</port>

Save and exit. To disable this, simply remove the **toggle="no"** from each desired .CFG input.

BGFX Effects for (nearly) Everyone

By default, MAME outputs an idealized version of the video as it would be on the way to the arcade cabinet's monitor, with minimal modification of the output (primarily to stretch the game image back to the aspect ratio the monitor would traditionally have, usually 4:3) – this works well, but misses some of the nostalgia factor. Arcade monitors

were never ideal, even in perfect condition, and the nature of a CRT display distorts that image in ways that change the appearance significantly.

Modern LCD monitors simply do not look the same, and even computer CRT monitors cannot match the look of an arcade monitor without help.

That's where the new BGFX renderer with HLSL comes into the picture.

HLSL simulates most of the effects that a CRT arcade monitor has on the video, making the result look a lot more authentic. However, HLSL requires some effort on the user's part: the settings you use are going to be tailored to your PC's system specs, and especially the monitor you're using. Additionally, there were hundreds of thousands of monitors out there in arcades. Each was tuned and maintained differently, meaning there is no one correct appearance to judge by either. Basic guidelines will be provided here to help you, but you may also wish to ask for opinions on popular MAME-centric forums.

Resolution and Aspect Ratio

Resolution is a very important subject for HLSL settings. You will want MAME to be using the native resolution of your monitor to avoid additional distortion and lag created by your monitor upscaling the display image.

While most arcade machines used a 4:3 ratio display (or 3:4 for vertically oriented monitors like Pac-Man), it's difficult to find a consumer display that is 4:3 at this point. The good news is that that extra space on the sides isn't wasted. Many arcade cabinets used bezel artwork around the main display, and should you have the necessary artwork files, MAME will display that artwork. Turn the artwork view to Cropped for best results.

Some older LCD displays used a native resolution of 1280x1024 and were a 5:4 aspect ratio. There's not enough extra space to display artwork, and you'll end up with some very slight pillarboxing, but the results will be still be good and on-par with a 4:3 monitor.

Getting Started with BGFX

You will need to have followed the initial MAME setup instructions elsewhere in this manual before beginning. Official MAME distributions include BGFX as of 172, so you don't need to download any additional files.

Open your MAME.INI in your text editor of choice (e.g. Notepad), and make sure the following options are set correctly:

- **video bgfx**

Now, you may want to take a moment to look below at the Configuration Settings section to see how to set up these next options.

As referenced in *Order of Config Loading*, MAME has a order in which it processes INI files. The BGFX settings can be edited in MAME.INI, but to take full advantage of the power of MAME's config files, you'll want to copy the BGFX settings from MAME.INI to one of the other config files and make changes there.)

In particular, you will want the **bgfx_screen_chains** to be specific to each game.

Save your .INI file(s) and you're ready to begin.

Configuration Settings

bgfx_path

> This is where your BGFX shader files are stored. By default, this will be the BGFX folder in your MAME installation.

bgfx_backend

Selects a rendering backend for BGFX to use. Possible choices include **d3d9**, **d3d11**, **opengl**, and **metal**. The default is **auto**, which will let MAME choose the best selection for you.

d3d9 – Direct3D 9.0 Renderer (Requires Windows XP or higher)
d3d11 – Direct3D 11.0 Renderer (Requires Windows Vista with D3D11 update or Windows 7 or higher)
opengl – OpenGL Renderer (Requires OpenGL drivers, may work better on some poorly designed video cards, supported on Linux/Mac OS X)
metal – Metal Apple Graphics API (Requires OS X 10.11 El Capitan or newer)

bgfx_debug

Enables BGFX debugging features. Most users will not need to use this.

bgfx_screen_chains

This dictates how to handle BGFX rendering on a per-display basis. Possible choices include **hlsl**, **unfiltered**, and **default**.

default – default bilinear filterered output
unfiltered – nearest neighbor unfiltered output
hlsl – HLSL display simulation through shaders

We make a distinction between emulated device screens (which we'll call a **screen**) and physical displays (which we'll call a **window**, set by **-numscreens**) here. We use colons (:) to seperate windows, and commas (,) to seperate screens. Commas always go on the outside of the chain (see House Mannequin example)

On a combination of a single window, single screen case, such as Pac-Man on one physical PC monitor, you can specify one entry like:

bgfx_screen_chains hlsl

Things get only slightly more complicated when we get to multiple windows and multiple screens.

On a single window, multiple screen game, such as Darius on one physical PC monitor, specify multiple entries (one per window) like:

bgfx_screen_chains hlsl,hlsl,hlsl

This also works with single screen games where you are mirroring the output to more than one physical display. For instance, you could set up Pac-Man to have one unfiltered output for use with video broadcasting while a second display is set up HLSL for playing on.

On a mulitple window, multiple screen game, such as Darius on three physical PC monitors, specify multiple entries (one per window) like:

bgfx_screen_chains hlsl:hlsl:hlsl

Another example game would be Taisen Hot Gimmick, which used two CRTs to show individual player hands to just that player. If using two windows (two physical displays):

bgfx_screen_chains hlsl:hlsl

One more special case is that Nichibutsu had a special cocktail mahjongg cabinet that used a CRT in the middle along with two LCD displays to show each player their hand. We would want the LCDs to be unfiltered and untouched as they were, while the CRT would be improved through HLSL. Since we want to give each player their own full screen display (two physical monitors) along with the LCD, we'll go with:

-numscreens 2 -view0 "Player 1" -view1 "Player 2" -video bgfx -bgfx_screen_chains hlsl,unfiltered,unfiltered:hlsl,unfiltered,unfiltered

This sets up the view for each display respectively, keeping HLSL effect on the CRT for each window (physical display) while going unfiltered for the LCD screens.

If using only one window (one display), keep in mind the game still has three screens, so we would use:

bgfx_screen_chains hlsl,unfiltered,unfiltered

Note that the commas are on the outside edges, and any colons are in the middle.

bgfx_shadow_mask

This specifies the shadow mask effect PNG file. By default this is **slot-mask.png**.

Tweaking BGFX HLSL Settings inside MAME

Warning: Currently BGFX HLSL settings are not saved or loaded from any configuration files. This is expected to change in the future.

Start by loading MAME with the game of your choice (e.g. **mame pacman**)

The tilde key (**~**) brings up the on-screen display options. Use up and down to go through the various settings, while left and right will allow you to change that setting. Results will be shown in real time as you're changing these settings.

Note that settings are individually changable on a per-screen basis.

HLSL Effects for Windows

By default, MAME outputs an idealized version of the video as it would be on the way to the arcade cabinet's monitor, with minimal modification of the output (primarily to stretch the game image back to the aspect ratio the monitor would traditionally have, usually 4:3) – this works well, but misses some of the nostalgia factor. Arcade monitors

were never ideal, even in perfect condition, and the nature of a CRT display distorts that image in ways that change the appearance significantly.

Modern LCD monitors simply do not look the same, and even computer CRT monitors cannot match the look of an arcade monitor without help.

That's where HLSL comes into the picture.

HLSL simulates most of the effects that a CRT arcade monitor has on the video, making the result look a lot more authentic. However, HLSL requires some effort on the user's part: the settings you use are going to be tailored to your PC's system specs, and especially the monitor you're using. Additionally, there were hundreds of thousands of monitors out there in arcades. Each was tuned and maintained differently, meaning there is no one correct appearance to judge by either. Basic guidelines will be provided here to help you, but you may also wish to ask for opinions on popular MAME-centric forums.

Resolution and Aspect Ratio

Resolution is a very important subject for HLSL settings. You will want MAME to be using the native resolution of your monitor to avoid additional distortion and lag created by your monitor upscaling the display image.

While most arcade machines used a 4:3 ratio display (or 3:4 for vertically oriented monitors like Pac-Man), it's difficult to find a consumer display that is 4:3 at this point. The good news is that that extra space on the sides isn't wasted. Many arcade cabinets used bezel artwork around the main display, and should you have the necessary artwork files, MAME will display that artwork. Turn the artwork view to Cropped for best results.

Some older LCD displays used a native resolution of 1280x1024 and were a 5:4 aspect ratio. There's not enough extra space to display artwork, and you'll end up with some very slight pillarboxing, but the results will be still be good and on-par with a 4:3 monitor.

Getting Started with HLSL

You will need to have followed the initial MAME setup instructions elsewhere in this manual before beginning. Official MAME distributions include HLSL by default, so you don't need to download any additional files.

Open your MAME.INI in your text editor of choice (e.g. Notepad), and make sure the following options are set correctly:

- **video d3d**
- **filter 0**

The former is required because HLSL requires Direct3D support. The latter turns off extra filtering that interferes with HLSL output.

Lastly, one more edit will turn HLSL on:

- **hlsl_enable 1**

Save the .INI file and you're ready to begin.

Several presets have been included in the INI folder with MAME, allowing for good quick starting points for Nintendo Game Boy, Nintendo Game Boy Advance, Raster, and Vector monitor settings.

Tweaking HLSL Settings inside MAME

For multiple, complicated to explain reasons, HLSL settings are no longer saved when you exit MAME. This means that while tweaking settings is a little more work on your part, the results will always come out as expected.

Start by loading MAME with the game of your choice (e.g. **mame pacman**)

The tilde key (**~**) brings up the on-screen display options. Use up and down to go through the various settings, while left and right will allow you to change that setting. Results will be shown in real time as you're changing these settings.

Once you've found settings you like, write the numbers down on a notepad and exit MAME.

Configuration Editing

As referenced in *Order of Config Loading*, MAME has a order in which it processes INI files. The HLSL settings can be edited in MAME.INI, but to take full advantage of the power of MAME's config files, you'll want to copy the HLSL settings from MAME.INI to one of the other config files and make changes there.

For instance, once you've found HLSL settings you think are appropriate for Neo Geo games, you can put those settings into neogeo.ini so that all Neo-Geo games will be able to take advantage of those settings without needing to add it to every game INI manually.

Configuration Settings

hlslpath

This is where your HLSL files are stored. By default, this will be the HLSL folder in your MAME installation.

hlsl_snap_width
hlsl_snap_height

Sets the resolution that Alt+F12 HLSL screenshots are output at.

shadow_mask_alpha (*Shadow Mask Amount*)

This defines how strong the effect of the shadowmask is. Acceptable range is from 0 to 1, where 0 will show no shadowmask effect, 1 will be a completely opaque shadowmask, and 0.5 will be 50% transparent.

shadow_mask_tile_mode (*Shadow Mask Tile Mode*)

This defines whether the shadowmask should be tiled based on the screen resolution of your monitor or based on the source resolution of the emulated system. Valid values are 0 for *Screen* mode and 1 for *Source* mode.

shadow_mask_texture
shadow_mask_x_count (*Shadow Mask Pixel X Count*)
shadow_mask_y_count (*Shadow Mask Pixel Y Count*)
shadow_mask_usize (*Shadow Mask U Size*)
shadow_mask_vsize (*Shadow Mask V Size*)
shadow_mask_x_count (*Shadow Mask U Offset*)
shadow_mask_y_count (*Shadow Mask V Offset*)

These settings need to be set in unison with one another. In particular, **shadow_mask_texture** sets rules for how you need to set the other options.

shadow_mask_texture sets the texture of the shadowmask effect. Three shadowmasks are included with MAME: *aperture-grille.png*, *shadow-mask.png*, and *slot-mask.png*

shadow_mask_usize and **shadow_mask_vsize** define the used size of the shadow_mask_texture in percentage, staring at the top-left corner. The means for a texture with the actual size of 24x24 pixel and an u/v size of 0.5,0.5 the top-left 12x12 pixel will be used. Keep in mind to define an u/v size that makes is possible to tile the texture without gaps or glitches. 0.5,0.5 is fine for any shadowmask texture that is included with MAME.

shadow_mask_x_count and **shadow_mask_y_count** define how many screen pixel should be used to display the u/v sized texture. e.g. if you use the example from above and define a x/y count of 12,12 every pixel of the texture will be displayed 1:1 on the screen, if you define a x/y count of 24,24 the texture will be displayed twice as large.

example settings for **shadow_mask.png**:

shadow_mask_texture shadow-mask.png
shadow_mask_x_count 12
shadow_mask_y_count 6 or 12
shadow_mask_usize 0.5
shadow_mask_vsize 0.5

example settings for **slot-mask.png**:

shadow_mask_texture slot-mask.png
shadow_mask_x_count 12
shadow_mask_y_count 8 or 16
shadow_mask_usize 0.5
shadow_mask_vsize 0.5

example settings for **aperture-grille**:

shadow_mask_texture aperture-grille.png
shadow_mask_x_count 12
shadow_mask_y_count 12 or any
shadow_mask_usize 0.5
shadow_mask_vsize 0.5

shadow_mask_uoffset and **shadow_mask_voffset** can be used to tweak the alignment of the final shadowmask in subpixel range. Range is from -1.00 to 1.00, where 0.5 moves the shadowmask by 50 percent of the u/v sized texture.

distortion (*Quadric Distortion Amount*)

This setting determines strength of the quadric distortion of the screen image.

cubic_distortion (*Cubic Distortion Amount*)

This setting determines strength of the qubic distortion of the screen image.

Both distortion factors can be negative to compensate each other. e.g. distortion 0.5 and cubic_distortion -0.5

distort_corner (*Distorted Corner Amount*)

This setting determines strength of distortion of the screen corners, which does not affect the distortion of screen image itself.

round_corner (*Rounded Corner Amount*)

The corners of the display can be rounded off through the use of this setting.

smooth_border (*Smooth Border Amount*)

Sets a smoothened/blurred border around the edges of the screen.

reflection (*Reflection Amount*)

If set above 0, this creates a white reflective blotch on the display. By default, this is put in the upper right corner of the display. By editing the POST.FX file's GetSpotAddend section, you can change the location. Range is from 0.00 to 1.00.

vignetting (*Vignetting Amount*)

When set above 0, will increasingly darken the outer edges of the display in a pseudo-3D effect. Range is from 0.00 to 1.00.

scanline_alpha (*Scanline Amount*)

This defines how strong the effect of the scanlines are. Acceptable range is from 0 to 1, where 0 will show no scanline effect, 1 will be a completely black line, and 0.5 will be 50% transparent. Note that arcade monitors did not have completely black scanlines.

scanline_size (*Overall Scanline Scale*)

The overall spacing of the scanlines is set with this option. Setting it at 1 represents consistent alternating spacing between display lines and scanlines.

scanline_height (*Individual Scanline Scale*)

This determines the overall size of each scanline. Setting lower than 1 makes them thinner, larger than 1 makes them thicker.

scanline_variation (*Scanline Variation*)

This affects the size of each scanline depending on its brightness. Brighter scanlines will be thicker than darker scanline. Acceptable range is from 0 to 2.0, with the default being 1.0. At 0.0 all scanlines will have the same

size independent of their brightness.

scanline_bright_scale (*Scanline Brightness Scale*)

Specifies how bright the scanlines are. Larger than 1 will make them brighter, lower will make them dimmer. Setting to 0 will make scanlines disappear entirely.

scanline_bright_offset (*Scanline Brightness Offset*)

This will give scanlines a glow/overdrive effect, softening and smoothing the top and bottom of each scanline.

scanline_jitter (*Scanline Jitter Amount*)

Specifies the wobble or jitter of the scanlines, causing them to jitter on the monitor. Warning: Higher settings may hurt your eyes.

hum_bar_alpha (*Hum Bar Amount*)

Defines the strength of the hum bar effect.

defocus (*Defocus*)

This option will defocus the display, blurring individual pixels like an extremely badly maintained monitor. Specify as X,Y values (e.g. **defocus 1,1**)

converge_x (*Linear Convergence X, RGB*)
converge_y (*Linear Convergence Y, RGB*)
radial_converge_x (*Radial Convergence X, RGB*)
radial_converge_y (*Radial Convergence Y, RGB*)

Adjust the convergence of the red, green, and blue channels in a given direction. Many badly maintained monitors with bad convergence would bleed colored ghosting off-center of a sprite, and this simulates that.

red_ratio (*Red Output from RGB*)
grn_ratio (*Green Output from RGB*)
blu_ratio (*Blue Output from RGB*)

Defines a 3x3 matrix that is multiplied with the RGB signals to simulate color channel interference. For instance, a green channel of (0.100, 1.000, 0.250) is weakened 10% by the red channel and strengthened 25% through the blue channel.

offset (*Signal Offset*)

Strengthen or weakens the current color value of a given channel. For instance, a red signal of 0.5 with an offset of 0.2 will be raised to 0.7

scale (*Signal Scale*)

Applies scaling to the current color value of the channel. For instance, a red signal of 0.5 with a scale of 1.1 will result in a red signal of 0.55

power (*Signal Exponent, RGB*)

Exponentiate the current color value of the channel, also called gamma. For instance, a red signal of 0.5 with red power of 2 will result in a red signal of 0.25

This setting also can be used to adjust line thickness in vector games.

floor (*Signal Floor, RGB*)

Sets the absolute minimum color value of a channel. For instance, a red signal of 0.0 (total absence of red) with a red floor of 0.2 will result in a red signal of 0.2

Typically used in conjunction with artwork turned on to make the screen have a dim raster glow.

phosphor_life (*Phosphor Persistence, RGB*)

How long the color channel stays on the screen, also called phosphor ghosting. 0 gives absolutely no ghost effect, and 1 will leave a contrail behind that is only overwritten by a higher color value.

This also affects vector games quite a bit.

saturation (*Color Saturation*)

Color saturation can be adjusted here.

bloom_blend_mode (*Bloom Blend Mode*)

Determines the mode of the bloom effect. Valid values are 0 for *Brighten* mode and 1 for *Darken* mode, last is only useful for systems with STN LCD.

bloom_scale (*Bloom Scale*)

Determines the intensity of bloom effect. Arcade CRT displays had a tendency towards bloom, where bright colors could bleed out into neighboring pixels. This effect is extremely graphics card intensive, and can be turned completely off to save GPU power by setting it to 0

bloom_overdrive (*Bloom Overdrive, RGB*)

Sets a RGB color, separated by commas, that has reached the brightest possible color and will be overdriven to white. This is only useful on color raster, color LCD, or color vector games.

bloom_lvl0_weight (*Bloom Level 0 Scale*)
bloom_lvl1_weight (*Bloom Level 1 Scale*)
. . . .
bloom_lvl7_weight (*Bloom Level 7 Scale*)

bloom_lvl8_weight (*Bloom Level 8 Scale*)

These define the bloom effect. Range is from 0.00 to 1.00. If used carefully in conjunction with phosphor_life, glowing/ghosting for moving objects can be achieved.

hlsl_write

Enables writing of an uncompressed AVI video with the HLSL effects included with set to *1*. This uses a massive amount of disk space very quickly, so a large HD with fast write speeds is highly recommended. Default is *0*, which is off.

Suggested defaults for raster-based games:

bloom_lvl0_weight 1.00	Bloom level 0 weight	Full-size target.
bloom_lvl1_weight 0.64	Bloom level 1 weight	1/4 smaller that level 0 target
bloom_lvl2_weight 0.32	Bloom level 2 weight	1/4 smaller that level 1 target
bloom_lvl3_weight 0.16	Bloom level 3 weight	1/4 smaller that level 2 target
bloom_lvl4_weight 0.08	Bloom level 4 weight	1/4 smaller that level 3 target
bloom_lvl5_weight 0.06	Bloom level 5 weight	1/4 smaller that level 4 target
bloom_lvl6_weight 0.04	Bloom level 6 weight	1/4 smaller that level 5 target
bloom_lvl7_weight 0.02	Bloom level 7 weight	1/4 smaller that level 6 target
bloom_lvl8_weight 0.01	Bloom level 8 weight	1/4 smaller that level 7 target

Vector Games

HLSL effects can also be used with vector games. Due to a wide variance of vector settings to optimize for each individual game, it is heavily suggested you add these to per-game INI files (e.g. tempest.ini)

Shadowmasks were only present on color vector games, and should not be used on monochrome vector games. Additionally, vector games did not use scanlines, so that should also be turned off.

Open your INI file in your text editor of choice (e.g. Notepad), and make sure the following options are set correctly:

- **video d3d**
- **filter 0**
- **hlsl_enable 1**

In the Core Vector Options section:

- **beam_width_min 1.0** (*Beam Width Minimum*)
- **beam_width_max 1.0** (*Beam Width Maximum*)
- **beam_intensity_weight 0.0** (*Beam Intensity Weight*)
- **flicker 0.0** (*Vector Flicker*)

In the Vector Post-Processing Options section:

- **vector_beam_smooth 0.0** (*Vector Beam Smooth Amount*)
- **vector_length_scale 0.5** (*Vector Attenuation Maximum*)
- **vector_length_ratio 0.5** (*Vector Attenuation Length Minimum*)

Suggested settings for vector games:

- **bloom_scale** should typically be set higher for vector games than raster games. Try between 0.4 and 1.0 for best effect.
- **bloom_overdrive** should only be used with color vector games.
- **bloom_lvl_weights** should be set as follows:

bloom_lvl0_weight 1.00	Bloom level 0 weight	Full-size target.
bloom_lvl1_weight 0.48	Bloom level 1 weight	1/4 smaller that level 0 target
bloom_lvl2_weight 0.32	Bloom level 2 weight	1/4 smaller that level 1 target
bloom_lvl3_weight 0.24	Bloom level 3 weight	1/4 smaller that level 2 target
bloom_lvl4_weight 0.16	Bloom level 4 weight	1/4 smaller that level 3 target
bloom_lvl5_weight 0.24	Bloom level 5 weight	1/4 smaller that level 4 target
bloom_lvl6_weight 0.32	Bloom level 6 weight	1/4 smaller that level 5 target
bloom_lvl7_weight 0.48	Bloom level 7 weight	1/4 smaller that level 6 target
bloom_lvl8_weight 0.64	Bloom level 8 weight	1/4 smaller that level 7 target

GLSL Effects for *nix, OS X, and Windows

By default, MAME outputs an idealized version of the video as it would be on the way to the arcade cabinet's monitor, with minimal modification of the output (primarily to stretch the game image back to the aspect ratio the monitor would traditionally have, usually 4:3) – this works well, but misses some of the nostalgia factor. Arcade monitors were never ideal, even in perfect condition, and the nature of a CRT display distorts that image in ways that change the appearance significantly.

Modern LCD monitors simply do not look the same, and even computer CRT monitors cannot match the look of an arcade monitor without help.

That's where GLSL comes into the picture.

GLSL simulates most of the effects that a CRT arcade monitor has on the video, making the result look a lot more authentic. However, GLSL requires some effort on the user's part: the settings you use are going to be tailored to your PC's system specs, and especially the monitor you're using. Additionally, there were hundreds of thousands of monitors out there in arcades. Each was tuned and maintained differently, meaning there is no one correct appearance to judge by either. Basic guidelines will be provided here to help you, but you may also wish to ask for opinions on popular MAME-centric forums.

Resolution and Aspect Ratio

Resolution is a very important subject for GLSL settings. You will want MAME to be using the native resolution of your monitor to avoid additional distortion and lag created by your monitor upscaling the display image.

While most arcade machines used a 4:3 ratio display (or 3:4 for vertically oriented monitors like Pac-Man), it's difficult to find a consumer display that is 4:3 at this point. The good news is that that extra space on the sides isn't wasted.

Many arcade cabinets used bezel artwork around the main display, and should you have the necessary artwork files, MAME will display that artwork. Turn the artwork view to Cropped for best results.

Some older LCD displays used a native resolution of 1280x1024, which is a 5:4 aspect ratio. There's not enough extra space to display artwork, and you'll end up with some very slight pillarboxing, but the results will be on-par with a 4:3 monitor.

Getting Started with GLSL

You will need to have followed the initial MAME setup instructions elsewhere in this manual before beginning. Official MAME distributions include GLSL support by default, but do NOT include the GLSL shader files. You will need to obtain the shader files from third party online sources.

Open your MAME.INI in your text editor of choice (e.g. Notepad), and make sure the following options are set correctly:

- **video opengl**
- **filter 0**

The former is required because GLSL requires OpenGL support. The latter turns off extra filtering that interferes with GLSL output.

Lastly, one more edit will turn GLSL on:

- **gl_glsl 1**

Save the .INI file and you're ready to begin.

Tweaking GLSL Settings inside MAME

For multiple, complicated to explain reasons, GLSL settings are no longer saved when you exit MAME. This means that while tweaking settings is a little more work on your part, the results will always come out as expected.

Start by loading MAME with the game of your choice (e.g. **mame pacman**)

The tilde key (**~**) brings up the on-screen display options. Use up and down to go through the various settings, while left and right will allow you to change that setting. Results will be shown in real time as you're changing these settings.

Once you've found settings you like, write the numbers down on a notepad and exit MAME.

Configuration Editing

As referenced in *Order of Config Loading*, MAME has a order in which it processes INI files. The GLSL settings can be edited in MAME.INI, but to take full advantage of the power of MAME's config files, you'll want to copy the GLSL settings from MAME.INI to one of the other config files and make changes there.

For instance, once you've found GLSL settings you think are appropriate for Neo Geo games, you can put those settings into neogeo.ini so that all Neo-Geo games will be able to take advantage of those settings without needing to add it to every game INI manually.

Configuration Settings

gl_glsl

Enables GLSL when set to 1, disabled if set to 0. Defaults to *0*.

gl_glsl_filter

Enables filtering to GLSL output. Reduces jagginess at the cost of blurriness.

glsl_shader_mame0

...

glsl_shader_mame9

Specifies the shaders to run, in the order from 0 to 9. See your shader pack author for details on which to run in which order for best effect.

glsl_shader_screen0

...

glsl_shader_screen9

Specifies screen to apply the shaders on.

Stable Controller IDs

By default, the mapping between devices and controller IDs is not stable. For instance, a gamepad controller may be assigned to "Joy 1" initially, but after a reboot, it may get re-assigned to "Joy 3".

The reason is that MAME enumerates attached devices and assigns controller IDs based on the enumeration order. Factors that can cause controller IDs to change include plugging / unplugging USB devices, changing ports / hubs and even system reboots.

It is quite cumbersome to ensure that controller IDs are always correct.

That's where the "mapdevice" configuration setting comes into the picture. This setting allows you to map a device id to a controller ID, ensuring that the specified device always maps to the same controller ID in MAME.

Usage of mapdevice

The "mapdevice" xml element is specified under the input xml element in the controller configuration file. It requires two attributes, "device" and "controller". NOTE: This setting only take effect when added to the **ctrlr** config file.

The "device" attribute specifies the id of the device to match. It may also be a substring of the id. To see the list of available devices, enable verbose output and available devices will then be listed to the console at startup (more on this below).

The "controller" attribute specifies the MAME controller ID. It is made up of a controller class (i.e. "JOYCODE", "GUNCODE", "MOUSECODE") and controller index. For example: "JOYCODE_1".

Example

Here's an example:

```
<mameconfig version="10">
    <system name="default">
        <input>
            <mapdevice device="VID_D209&amp;amp;PID_1601" controller="GUNCODE_1" />
            <mapdevice device="VID_D209&amp;amp;PID_1602" controller="GUNCODE_2" />
            <mapdevice device="XInput Player 1" controller="JOYCODE_1" />
            <mapdevice device="XInput Player 2" controller="JOYCODE_2" />

            <port type="P1_JOYSTICK_UP">
                <newseq type="standard">
                    JOYCODE_1_YAXIS_UP_SWITCH OR KEYCODE_8PAD
                </newseq>
            </port>
            ...
```

In the above example, we have four device mappings specified:

The first two mapdevice entries map player 1 and 2 lightguns to Gun 1 and Gun 2, respectively. We use a substring of the full raw device names to match each devices. Note that, since this is XML, we needed to escape the '&' using '&'.

The last two mapdevices entries map player 1 and player 2 gamepad controllers to Joy 1 and Joy 2, respectively. In this case, these are XInput devices.

Listing Available Devices

How did we obtain the device id's in the above example? Easy!

Run MAME with -v parameter to enable verbose output. It will then list available devices include the corresponding "device id" to the console.

Here an example:

```
Input: Adding Gun #0:
Input: Adding Gun #1:
Input: Adding Gun #2: HID-compliant mouse (device id:
\?HID#VID_045E&PID_0053#7&18297dcb&0&0000#{378de44c-56ef-11d1-bc8c-00a0c91405dd})
Input: Adding Gun #3: HID-compliant mouse (device id:
\?HID#IrDeviceV2&Col08#2&2818a073&0&0007#{378de44c-56ef-11d1-bc8c-00a0c91405dd})
Input: Adding Gun #4: HID-compliant mouse (device id:
\?HID#VID_D209&PID_1602&MI_02#8&389ab7f3&0&0000#{378de44c-56ef-11d1-bc8c-00a0c91405dd})
Input: Adding Gun #5: HID-compliant mouse (device id:
\?HID#VID_D209&PID_1601&MI_02#9&375eebb1&0&0000#{378de44c-56ef-11d1-bc8c-00a0c91405dd})
Input: Adding Gun #6: HID-compliant mouse (device id:
\?HID#VID_1241&PID_1111#8&198f3adc&0&0000#{378de44c-56ef-11d1-bc8c-00a0c91405dd})
Skipping DirectInput for XInput compatible joystick Controller (XBOX 360 For Windows).
Input: Adding Joy #0: ATRAK Device #1 (device id: ATRAK Device #1)
Skipping DirectInput for XInput compatible joystick Controller (XBOX 360 For Windows).
```

Input: Adding Joy #1: ATRAK Device #2 (**device id: ATRAK Device #2**)
Input: Adding Joy #2: XInput Player 1 (**device id: XInput Player 1**)
Input: Adding Joy #3: XInput Player 2 (**device id: XInput Player 2**)

Furthermore, when devices are mapped using mapdevice, you'll see that in the verbose logging too, such as:

Input: Remapped Gun #0: HID-compliant mouse (device id:
\?HID#VID_D209&PID_1601&MI_02#9&375eebb1&0&0000#{378de44c-56ef-11d1-bc8c-00a0c91405dd})
Input: Remapped Gun #1: HID-compliant mouse (device id:
\?HID#VID_D209&PID_1602&MI_02#8&389ab7f3&0&0000#{378de44c-56ef-11d1-bc8c-00a0c91405dd})
Input: Remapped Joy #0: XInput Player 1 (device id: XInput Player 1)
Input: Remapped Joy #1: XInput Player 2 (device id: XInput Player 2)

CHAPTER

SIX

MAME DEBUGGER

This section covers the built-in MAME debugger

General Debugger Commands

You can also type **help <command>** for further details on each command in the MAME Debugger interface.

do – evaluates the given expression
symlist – lists registered symbols
softreset – executes a soft reset
hardreset – executes a hard reset
print – prints one or more <item>s to the console
printf – prints one or more <item>s to the console using <format>
logerror – outputs one or more <item>s to the error.log
tracelog – outputs one or more <item>s to the trace file using <format>
tracesym – outputs one or more <item>s to the trace file
history – outputs a brief history of visited opcodes (**to fix: help missing for this command**)
trackpc – visually track visited opcodes [boolean to turn on and off, for the given cpu, clear]
trackmem – record which PC writes to each memory address [boolean to turn on and off, clear]
pcatmem – query which PC wrote to a given memory address for the current CPU
rewind – go back in time by loading the most recent rewind state
statesave – save a state file for the current driver
stateload – load a state file for the current driver
snap – save a screen snapshot.
source – reads commands from <filename> and executes them one by one
quit – exits MAME and the debugger

do

do <expression>

The do command simply evaluates the given <expression>. This is typically used to set or modify variables.

Examples:

do pc = 0

Sets the register 'pc' to 0.

Back to *General Debugger Commands*

symlist

symlist [<cpu>]

Lists registered symbols. If <cpu> is not specified, then symbols in the global symbol table are displayed; otherwise, the symbols for <cpu>'s specific CPU are displayed. Symbols are listed alphabetically. Read-only symbols are flagged with an asterisk.

Examples:

symlist

Displays the global symbol table.

symlist 2

Displays the symbols specific to CPU #2.

Back to *General Debugger Commands*

softreset

softreset

Executes a soft reset.

Examples:

softreset

Executes a soft reset.

Back to *General Debugger Commands*

hardreset

hardreset

Executes a hard reset.

Examples:

hardreset

Executes a hard reset.

Back to *General Debugger Commands*

print

print <item>[,...]

The print command prints the results of one or more expressions to the debugger console as hexadecimal values.

Examples:

print pc

Prints the value of 'pc' to the console as a hex number.

print a,b,a+b

Prints a, b, and the value of a+b to the console as hex numbers.

Back to *General Debugger Commands*

printf

printf <format>[,<item>[,...]]

The printf command performs a C-style printf to the debugger console. Only a very limited set of formatting options are available:

%[0][<n>]d – prints <item> as a decimal value with optional digit count and zero-fill

%[0][<n>]x – prints <item> as a hexadecimal value with optional digit count and zero-fill

All remaining formatting options are ignored. Use %% together to output a % character. Multiple lines can be printed by embedding a \n in the text.

Examples:

printf “PC=%04X”,pc

Prints PC=<pcval> where <pcval> is displayed in hexadecimal with 4 digits with zero-fill.

printf “A=%d, B=%dnC=%d”,a,b,a+b

Prints A=<aval>, B=<bval> on one line, and C=<a+bval> on a second line.

Back to *General Debugger Commands*

logerror

logerror <format>[,<item>[,...]]

The logerror command performs a C-style printf to the error log. Only a very limited set of formatting options are available:

%[0][<n>]d – logs <item> as a decimal value with optional digit count and zero-fill
%[0][<n>]x – logs <item> as a hexadecimal value with optional digit count and zero-fill

All remaining formatting options are ignored. Use %% together to output a % character. Multiple lines can be printed by embedding a \n in the text.

Examples:

logerror “PC=%04X”,pc

Logs PC=<pcval> where <pcval> is displayed in hexadecimal with 4 digits with zero-fill.

logerror “A=%d, B=%dnC=%d”,a,b,a+b

Logs A=<aval>, B=<bval> on one line, and C=<a+bval> on a second line.

Back to *General Debugger Commands*

tracelog

tracelog <format>[,<item>[,...]]

The tracelog command performs a C-style printf and routes the output to the currently open trace file (see the 'trace' command for details). If no file is currently open, tracelog does nothing. Only a very limited set of formatting options are available. See the *printf* help for details.

Examples:

tracelog "PC=%04X",pc

Outputs PC=<pcval> where <pcval> is displayed in hexadecimal with 4 digits with zero-fill.

printf "A=%d, B=%dnC=%d",a,b,a+b

Outputs A=<aval>, B=<bval> on one line, and C=<a+bval> on a second line.

Back to *General Debugger Commands*

tracesym

tracesym <item>[,...]

The tracesym command prints the specified symbols and routes the output to the currently open trace file (see the 'trace' command for details). If no file is currently open, tracesym does nothing.

Examples:

tracelog pc

Outputs PC=<pcval> where <pcval> is displayed in the default format.

printf a,b

Outputs A=<aval>, B=<bval> on one line.

Back to *General Debugger Commands*

trackpc

trackpc [<bool>,<cpu>,<bool>]

The trackpc command displays which program counters have already been visited in all disassembler windows. The first boolean argument toggles the process on and off. The second argument is a cpu selector; if no cpu is specified, the current cpu is automatically selected. The third argument is a boolean denoting if the existing data should be cleared or not.

Examples:

trackpc 1

Begin tracking the current cpu's pc.

trackpc 1, 0, 1

Continue tracking pc on cpu 0, but clear existing track info.

Back to *General Debugger Commands*

trackmem

trackmem [<bool>,<cpu>,<bool>]

The trackmem command logs the PC at each time a memory address is written to. The first boolean argument toggles the process on and off. The second argument is a cpu selector; if no cpu is specified, the current cpu is automatically selected. The third argument is a boolean denoting if the existing data should be cleared or not. Please refer to the pcatmem command for information on how to retrieve this data. Also, right clicking in a memory window will display the logged PC for the given address.

Examples:

trackmem

Begin tracking the current CPU's pc.

trackmem 1, 0, 1

Continue tracking memory writes on cpu 0, but clear existing track info.

Back to *General Debugger Commands*

pcatmem

pcatmem(p/d/i) <address>[,<cpu>]

pcatmemp <address>[,<cpu>] – query which PC wrote to a given program memory address for the current CPU
pcatmemd <address>[,<cpu>] – query which PC wrote to a given data memory address for the current CPU
pcatmemi <address>[,<cpu>] – query which PC wrote to a given I/O memory address for the current CPU (you can also query this info by right clicking in a memory window)

The pcatmem command returns which PC wrote to a given memory address for the current CPU. The first argument is the requested address. The second argument is a cpu selector; if no cpu is specified, the current cpu is automatically selected. Right clicking in a memory window will also display the logged PC for the given address.

Examples:

```
pcatmem 400000
```

Print which PC wrote this CPU's memory location 0x400000.

Back to *General Debugger Commands*

rewind

rewind[rw]

The rewind command loads the most recent RAM-based state. Rewind states, when enabled, are saved when "step", "over", or "out" command gets executed, storing the machine state as of the moment before actually stepping. Consecutively loading rewind states can work like reverse execution. Depending on which steps forward were taken previously, the behavior can be similar to GDB's "reverse-stepi" or "reverse-next". All output for this command is currently echoed into the running machine window. Previous memory and PC tracking statistics are cleared, actual reverse execution does not occur.

Back to *General Debugger Commands*

statesave

statesave[ss] <filename>

The statesave command creates a save state at this exact moment in time. The given state file gets written to the standard state directory (sta), and gets .sta added to it - no file extension necessary. All output for this command is currently echoed into the running machine window.

Examples:

```
statesave foo
```

Writes file 'foo.sta' in the default state save directory.

Back to *General Debugger Commands*

stateload

stateload[sl] <filename>

The stateload command retrieves a save state from disk. The given state file gets read from the standard state directory (sta), and gets .sta added to it - no file extension necessary. All output for this command is currently echoed into the running machine window. Previous memory and PC tracking statistics are cleared.

Examples:

stateload foo

Reads file 'foo.sta' from the default state save directory.

Back to *General Debugger Commands*

snap

snap [[<filename>], <scrnum>]

The snap command takes a snapshot of the current video display and saves it to the configured snapshot directory. If <filename> is specified explicitly, a single screenshot for <scrnum> is saved under the requested filename. If <filename> is omitted, all screens are saved using the same default rules as the "save snapshot" key in MAME proper.

Examples:

snap

Takes a snapshot of the current video screen and saves to the next non-conflicting filename in the configured snapshot directory.

snap shinobi

Takes a snapshot of the current video screen and saves it as 'shinobi.png' in the configured snapshot directory.

Back to *General Debugger Commands*

source

source <filename>

The source command reads in a set of debugger commands from a file and executes them one by one, similar to a batch file.

Examples:

source break_and_trace.cmd

Reads in debugger commands from break_and_trace.cmd and executes them.

Back to *General Debugger Commands*

quit

quit

The quit command exits MAME immediately.

Back to *General Debugger Commands*

Memory Debugger Commands

You can also type **help <command>** for further details on each command in the MAME Debugger interface.

dasm – disassemble to the given file
find – search program memory, data memory, or I/O memory for data
dump – dump program memory, data memory, or I/O memory as text
save – save binary program, data, or I/O memory to the given file
load – load binary program memory, data memory, or I/O memory from the given file
map – map logical program, data, or I/O address to physical address and bank

dasm

dasm <filename>,<address>,<length>[,<opcodes>[,<cpu>]]

The dasm command disassembles program memory to the file specified in the <filename> parameter. <address> indicates the address of the start of disassembly, and <length> indicates how much memory to disassemble. The range <address> through <address>+<length>-1 inclusive will be output to the file. By default, the raw opcode data

is output with each line. The optional <opcodes> parameter can be used to enable (1) or disable (0) this feature. Finally, you can disassemble code from another CPU by specifying the <cpu> parameter.

Examples:

dasm venture.asm,0,10000

Disassembles addresses 0-ffff in the current CPU, including raw opcode data, to the file 'venture.asm'.

dasm harddriv.asm,3000,1000,0,2

Disassembles addresses 3000-3fff from CPU #2, with no raw opcode data, to the file 'harddriv.asm'.

Back to *Memory Debugger Commands*

find

f[ind][{d|i}] <address>,<length>[,<data>[,...]]

The **find/findd/findi** commands search through memory for the specified sequence of data. 'find' will search program space memory, while 'findd' will search data space memory and 'findi' will search I/O space memory. <address> indicates the address to begin searching, and <length> indicates how much memory to search. <data> can either be a quoted string or a numeric value or expression or the wildcard character '?'. Strings by default imply a byte-sized search; non-string data is searched by default in the native word size of the CPU. To override the search size for non-strings, you can prefix the value with b. to force byte- sized search, w. for word-sized search, d. for dword-sized, and q. for qword-sized. Overrides are remembered, so if you want to search for a series of words, you need only to prefix the first value with a w. Note also that you can intermix sizes in order to perform more complex searches. The entire range <address> through <address>+<length>-1 inclusive will be searched for the sequence, and all occurrences will be displayed.

Examples:

find 0,10000,"HIGH SCORE",0

Searches the address range 0-ffff in the current CPU for the string "HIGH SCORE" followed by a 0 byte.

findd 3000,1000,w.abcd,4567

Searches the data memory address range 3000-3fff for the word-sized value abcd followed by the word-sized value 4567.

find 0,8000,"AAR",d.0,"BEN",w.0

Searches the address range 0000-7fff for the string "AAR" followed by a dword-sized 0 followed by the string "BEN", followed by a word-sized 0.

Back to *Memory Debugger Commands*

dump

dump[{d|i}] <filename>,<address>,<length>[,<size>[,<ascii>[,<cpu>]]]

The **dump/dumpd/dumpi** commands dump memory to the text file specified in the <filename> parameter.
'dump' will dump program space memory, while 'dumpd' will dump data space memory and 'dumpi' will dump I/O space memory.
<address> indicates the address of the start of dumping, and <length> indicates how much memory to dump. The range <address> through <address>+<length>-1 inclusive will be output to the file.
By default, the data will be output in byte format, unless the underlying address space is word/dword/qword-only. You can override this by specifying the <size> parameter, which can be used to group the data in 1, 2, 4 or 8-byte chunks.
The optional <ascii> parameter can be used to enable (1) or disable (0) the output of ASCII characters to the right of each line; by default, this is enabled.
Finally, you can dump memory from another CPU by specifying the <cpu> parameter.

Examples:

dump venture.dmp,0,10000

Dumps addresses 0-ffff in the current CPU in 1-byte chunks, including ASCII data, to the file 'venture.dmp'.

dumpd harddriv.dmp,3000,1000,4,0,3

Dumps data memory addresses 3000-3fff from CPU #3 in 4-byte chunks, with no ASCII data, to the file 'harddriv.dmp'.

Back to *Memory Debugger Commands*

save

save[{d|i}] <filename>,<address>,<length>[,<cpu>]

The **save/saved/savei** commands save raw memory to the binary file specified in the <filename> parameter.
'save' will save program space memory, while 'saved' will save data space memory and 'savei' will save I/O space memory.
<address> indicates the address of the start of saving, and <length> indicates how much memory to save. The range <address> through <address>+<length>-1 inclusive will be output to the file.
You can also save memory from another CPU by specifying the <cpu> parameter.

Examples:

save venture.bin,0,10000

Saves addresses 0-ffff in the current CPU to the binary file 'venture.bin'.

saved harddriv.bin,3000,1000,3

Saves data memory addresses 3000-3fff from CPU #3 to the binary file 'harddriv.bin'.

Back to *Memory Debugger Commands*

load

load[{d|i}] <filename>,<address>[,<length>,<cpu>]

The **load/loadd/loadi** commands load raw memory from the binary file specified in the <filename> parameter.
'load' will load program space memory, while 'loadd' will load data space memory and 'loadi' will load I/O space memory.
<address> indicates the address of the start of saving, and <length> indicates how much memory to load. The range <address> through <address>+<length>-1 inclusive will be read in from the file.
If you specify <length> = 0 or a length greater than the total length of the file it will load the entire contents of the file and no more.
You can also load memory from another CPU by specifying the <cpu> parameter.

NOTE: This will only actually write memory that is possible to overwrite in the Memory Window

Examples:

load venture.bin,0,10000

Loads addresses 0-ffff in the current CPU from the binary file 'venture.bin'.

loadd harddriv.bin,3000,1000,3

Loads data memory addresses 3000-3fff from CPU #3 from the binary file 'harddriv.bin'.

Back to *Memory Debugger Commands*

map

map[{d|i}] <address>

The map/mapd/mapi commands map a logical address in memory to the correct physical address, as well as specifying the bank.
'map' will map program space memory, while 'mapd' will map data space memory and 'mapi' will map I/O space memory.

Example:

map 152d0

Gives physical address and bank for logical address 152d0 in program memory

Back to *Memory Debugger Commands*

Execution Debugger Commands

You can also type **help <command>** for further details on each command in the MAME Debugger interface.

step – single steps for <count> instructions (F11)
over – single steps over <count> instructions (F10)
out – single steps until the current subroutine/exception handler is exited (Shift-F11)
go – resumes execution, sets temp breakpoint at <address> (F5)
gint – resumes execution, setting temp breakpoint if <irqline> is taken (F7)
gtime – resumes execution until the given delay has elapsed
gvblank – resumes execution, setting temp breakpoint on the next VBLANK (F8)
next – executes until the next CPU switch (F6)
focus – focuses debugger only on <cpu>
ignore – stops debugging on <cpu>
observe – resumes debugging on <cpu>
trace – trace the given CPU to a file (defaults to active CPU)
traceover – trace the given CPU to a file, but skip subroutines (defaults to active CPU)
traceflush – flushes all open trace files.

step

s[tep] [<count>=1]

The step command single steps one or more instructions in the currently executing CPU. By default, step executes one instruction each time it is issued. You can also tell step to step multiple instructions by including the optional <count> parameter.

Examples:

s

Steps forward one instruction on the current CPU.

step 4

Steps forward four instructions on the current CPU.

Back to *Execution Debugger Commands*

over

o[ver] [<count>=1]

The over command single steps "over" one or more instructions in the currently executing CPU, stepping over subroutine calls and exception handler traps and counting them as a single instruction. Note that when stepping over a subroutine call, code may execute on other CPUs before the subroutine call completes. By default, over executes one instruction each time it is issued. You can also tell step to step multiple instructions by including the optional <count> parameter.

Note that the step over functionality may not be implemented on all CPU types. If it is not implemented, then 'over' will behave exactly like 'step'.

Examples:

o

Steps forward over one instruction on the current CPU.

over 4

Steps forward over four instructions on the current CPU.

Back to *Execution Debugger Commands*

out

out

The out command single steps until it encounters a return from subroutine or return from exception instruction. Note that because it detects return from exception conditions, if you attempt to step out of a subroutine and an

interrupt/exception occurs before you hit the end, then you may stop prematurely at the end of the exception handler.

Note that the step out functionality may not be implemented on all CPU types. If it is not implemented, then 'out' will behave exactly like 'step'.

Examples:

out

Steps until the current subroutine or exception handler returns.

Back to *Execution Debugger Commands*

go

g[o] [<address>]

The go command resumes execution of the current code. Control will not be returned to the debugger until a breakpoint or watchpoint is hit, or until you manually break in using the assigned key. The go command takes an optional <address> parameter which is a temporary unconditional breakpoint that is set before executing, and automatically removed when hit.

Examples:

g

Resume execution until the next break/watchpoint or until a manual break.

g 1234

Resume execution, stopping at address 1234 unless something else stops us first.

Back to *Execution Debugger Commands*

gvblank

gv[blank]

The gvblank command resumes execution of the current code. Control will not be returned to the debugger until a breakpoint or watchpoint is hit, or until the next VBLANK occurs in the emulator.

Examples:

gv

Resume execution until the next break/watchpoint or until the next VBLANK.

Back to *Execution Debugger Commands*

gint

gi[nt] [<irqline>]

The gint command resumes execution of the current code. Control will not be returned to the debugger until a breakpoint or watchpoint is hit, or until an IRQ is asserted and acknowledged on the current CPU. You can specify <irqline> if you wish to stop execution only on a particular IRQ line being asserted and acknowledged. If <irqline> is omitted, then any IRQ line will stop execution.

Examples:

gi

Resume execution until the next break/watchpoint or until any IRQ is asserted and acknowledged on the current CPU.

gint 4

Resume execution until the next break/watchpoint or until IRQ line 4 is asserted and acknowledged on the current CPU.

Back to *Execution Debugger Commands*

gtime

gt[ime] <milliseconds>

The gtime command resumes execution of the current code. Control will not be returned to the debugger until a specified delay has elapsed. The delay is in milliseconds.

Example:

gtime #10000

Resume execution for ten seconds

Back to *Execution Debugger Commands*

next

n[ext]

The next command resumes execution and continues executing until the next time a different CPU is scheduled. Note that if you have used 'ignore' to ignore certain CPUs, you will not stop until a non-'ignore'd CPU is scheduled.

Back to *Execution Debugger Commands*

focus

focus <cpu>

Sets the debugger focus exclusively to the given <cpu>. This is equivalent to specifying 'ignore' on all other CPUs.

Example:

focus 1

Focus exclusively CPU #1 while ignoring all other CPUs when using the debugger.

Back to *Execution Debugger Commands*

ignore

ignore [<cpu>[,<cpu>[,...]]]

Ignores the specified <cpu> in the debugger. This means that you won't ever see execution on that CPU, nor will you be able to set breakpoints on that CPU. To undo this change use the 'observe' command. You can specify multiple <cpu>s in a single command. Note also that you are not permitted to ignore all CPUs; at least one must be active at all times.

Examples:

ignore 1

Ignore CPU #1 when using the debugger.

ignore 2,3,4

Ignore CPU #2, #3 and #4 when using the debugger.

ignore

List the CPUs that are currently ignored.

Back to *Execution Debugger Commands*

observe

observe [<cpu>[,<cpu>[,...]]]

Re-enables interaction with the specified <cpu> in the debugger. This command undoes the effects of the 'ignore' command. You can specify multiple <cpu>s in a single command.

Examples:

observe 1

Stop ignoring CPU #1 when using the debugger.

observe 2,3,4

Stop ignoring CPU #2, #3 and #4 when using the debugger.

observe

List the CPUs that are currently observed.

Back to *Execution Debugger Commands*

trace

trace {<filename>|OFF}[,<cpu>[,[noloop|logerror][,<action>]]]

Starts or stops tracing of the execution of the specified <cpu>. If <cpu> is omitted, the currently active CPU is specified.

When enabling tracing, specify the filename in the <filename> parameter. To disable tracing, substitute the keyword 'off' for <filename>.

<detectloops> should be either true or false.

If ‘noloop’ is omitted, the trace will have loops detected and condensed to a single line. If ‘noloop’ is specified, the trace will contain every opcode as it is executed.

If ‘logerror’ is specified, logerror output will augment the trace. If you wish to log additional information on each trace, you can append an <action> parameter which is a command that is executed before each trace is logged. Generally, this is used to include a ‘tracelog’ command. Note that you may need to embed the action within braces { } in order to prevent commas and semicolons from being interpreted as applying to the trace command itself.

Examples:

trace joust.tr

Begin tracing the currently active CPU, logging output to joust.tr.

trace dribling.tr,0

Begin tracing the execution of CPU #0, logging output to dribling.tr.

trace starswep.tr,0,noloop

Begin tracing the execution of CPU #0, logging output to starswep.tr, with loop detection disabled.

trace starswep.tr,0,logerror

Begin tracing the execution of CPU #0, logging output (along with logerror output) to starswep.tr.

trace starswep.tr,0,logerror|noloop

Begin tracing the execution of CPU #0, logging output (along with logerror output) to starswep.tr, with loop detection disabled.

trace >>pigskin.tr

Begin tracing the currently active CPU, appending log output to pigskin.tr.

trace off,0

Turn off tracing on CPU #0.

trace asteroid.tr,0,,{tracelog “A=%02X ”,a}

Begin tracing the execution of CPU #0, logging output to asteroid.tr. Before each line, output A=<aval> to the tracelog.

Back to *Execution Debugger Commands*

traceover

traceover {<filename>|OFF}[,<cpu>[,<detectloops>[,<action>]]]

Starts or stops tracing of the execution of the specified <cpu>.

When tracing reaches a subroutine or call, tracing will skip over the subroutine. The same algorithm is used as is used in the step over command. This means that traceover will not work properly when calls are recursive or the return address is not immediately following the call instruction.

<detectloops> should be either true or false. If <detectloops> is *true or omitted*, the trace will have loops detected and condensed to a single line. If it is false, the trace will contain every opcode as it is executed.
If <cpu> is omitted, the currently active CPU is specified.
When enabling tracing, specify the filename in the <filename> parameter.
To disable tracing, substitute the keyword 'off' for <filename>.
If you wish to log additional information on each trace, you can append an <action> parameter which is a command that is executed before each trace is logged. Generally, this is used to include a 'tracelog' command. Note that you may need to embed the action within braces { } in order to prevent commas and semicolons from being interpreted as applying to the trace command itself.

Examples:

traceover joust.tr

Begin tracing the currently active CPU, logging output to joust.tr.

traceover dribling.tr,0

Begin tracing the execution of CPU #0, logging output to dribling.tr.

traceover starswep.tr,0,false

Begin tracing the execution of CPU #0, logging output to starswep.tr, with loop detection disabled.

traceover off,0

Turn off tracing on CPU #0.

traceover asteroid.tr,0,true,{tracelog "A=%02X ",a}

Begin tracing the execution of CPU #0, logging output to asteroid.tr. Before each line, output A=<aval> to the tracelog.

Back to *Execution Debugger Commands*

traceflush

traceflush

Flushes all open trace files.

Back to *Execution Debugger Commands*

Breakpoint Debugger Commands

You can also type **help <command>** for further details on each command in the MAME Debugger interface.

bpset – sets breakpoint at <address>
bpclear – clears a given breakpoint or all if no <bpnum> specified
bpdisable – disables a given breakpoint or all if no <bpnum> specified
bpenable – enables a given breakpoint or all if no <bpnum> specified
bplist – lists all the breakpoints

bpset

bp[set] <address>[,<condition>[,<action>]]

Sets a new execution breakpoint at the specified <address>.
The optional <condition> parameter lets you specify an expression that will be evaluated each time the breakpoint is hit. If the result of the expression is true (non-zero), the breakpoint will actually halt execution; otherwise, execution will continue with no notification.
The optional <action> parameter provides a command that is executed whenever the breakpoint is hit and the <condition> is true. Note that you may need to embed the action within braces { } in order to prevent commas and semicolons from being interpreted as applying to the bpset command itself. Each breakpoint that is set is assigned an index which can be used in other breakpoint commands to reference this breakpoint.

Examples:

bp 1234

Set a breakpoint that will halt execution whenever the PC is equal to 1234.

bp 23456,a0 == 0 && a1 == 0

Set a breakpoint that will halt execution whenever the PC is equal to 23456 AND the expression (a0 == 0 && a1 == 0) is true.

bp 3456,1,{printf “A0=%08Xn”,a0; g}

Set a breakpoint that will halt execution whenever the PC is equal to 3456. When this happens, print A0=<a0val> and continue executing.

```
bp 45678,a0==100,{a0 = ff; g}
```

Set a breakpoint that will halt execution whenever the PC is equal to 45678 AND the expression (a0 == 100) is true. When that happens, set a0 to ff and resume execution.

```
temp0 = 0; bp 567890,++temp0 >= 10
```

Set a breakpoint that will halt execution whenever the PC is equal to 567890 AND the expression (++temp0 >= 10) is true. This effectively breaks only after the breakpoint has been hit 16 times.

Back to *Breakpoint Debugger Commands*

bpclear

bpclear [<bpnum>]

The bpclear command clears a breakpoint. If <bpnum> is specified, only the requested breakpoint is cleared, otherwise all breakpoints are cleared.

Examples:

```
bpclear 3
```

Clear breakpoint index 3.

```
bpclear
```

Clear all breakpoints.

Back to *Breakpoint Debugger Commands*

bpdisable

bpdisable [<bpnum>]

The bpdisable command disables a breakpoint. If <bpnum> is specified, only the requested breakpoint is disabled, otherwise all breakpoints are disabled. Note that disabling a breakpoint does not delete it, it just temporarily marks the breakpoint as inactive.

Examples:

bpdisable 3

Disable breakpoint index 3.

bpdisable

Disable all breakpoints.

Back to *Breakpoint Debugger Commands*

bpenable

bpenable [<bpnum>]

The bpenable command enables a breakpoint. If <bpnum> is specified, only the requested breakpoint is enabled, otherwise all breakpoints are enabled.

Examples:

bpenable 3

Enable breakpoint index 3.

bpenable

Enable all breakpoints.

Back to *Breakpoint Debugger Commands*

bplist

bplist

The bplist command lists all the current breakpoints, along with their index and any conditions or actions attached to them.

Back to *Breakpoint Debugger Commands*

Watchpoint Debugger Commands

You can also type **help <command>** for further details on each command in the MAME Debugger interface.

wpset – sets program, data, or I/O space watchpoint
wpclear – clears a given watchpoint or all if no <wpnum> specified
wpdisable – disables a given watchpoint or all if no <wpnum> specified
wpenable – enables a given watchpoint or all if no <wpnum> specified
wplist – lists all the watchpoints

wpset

wp[{d|i}][set] <address>,<length>,<type>[,<condition>[,<action>]]

Sets a new watchpoint starting at the specified <address> and extending for <length>. The inclusive range of the watchpoint is <address> through <address> + <length> - 1.
The 'wpset' command sets a watchpoint on program memory; the 'wpdset' command sets a watchpoint on data memory; and the 'wpiset' sets a watchpoint on I/O memory.
The <type> parameter specifies which sort of accesses to trap on. It can be one of three values: 'r' for a read watchpoint 'w' for a write watchpoint, and 'rw' for a read/write watchpoint.

The optional <condition> parameter lets you specify an expression that will be evaluated each time the watchpoint is hit. If the result of the expression is true (non-zero), the watchpoint will actually halt execution; otherwise, execution will continue with no notification.
The optional <action> parameter provides a command that is executed whenever the watchpoint is hit and the <condition> is true. Note that you may need to embed the action within braces { } in order to prevent commas and semicolons from being interpreted as applying to the wpset command itself.
Each watchpoint that is set is assigned an index which can be used in other watchpoint commands to reference this watchpoint.
In order to help <condition> expressions, two variables are available. For all watchpoints, the variable 'wpaddr' is set to the address that actually triggered the watchpoint. For write watchpoints, the variable 'wpdata' is set to the data that is being written.

Examples:

wp 1234,6,rw

Set a watchpoint that will halt execution whenever a read or write occurs in the address range 1234-1239 inclusive.

wp 23456,a,w,wpdata == 1

Set a watchpoint that will halt execution whenever a write occurs in the address range 23456-2345f AND the data written is equal to 1.

wp 3456,20,r,1,{printf "Read @ %08Xn",wpaddr; g}

Set a watchpoint that will halt execution whenever a read occurs in the address range 3456-3475. When this happens, print Read @ <wpaddr> and continue executing.

```
temp0 = 0; wp 45678,1,w,wpdata==f0,{temp0++; g}
```

Set a watchpoint that will halt execution whenever a write occurs to the address 45678 AND the value being written is equal to f0. When that happens, increment the variable temp0 and resume execution.

Back to *Watchpoint Debugger Commands*

wpclear

wpclear [<wpnum>]

The wpclear command clears a watchpoint. If <wpnum> is specified, only the requested watchpoint is cleared, otherwise all watchpoints are cleared.

Examples:

```
wpclear 3
```

Clear watchpoint index 3.

```
wpclear
```

Clear all watchpoints.

Back to *Watchpoint Debugger Commands*

wpdisable

wpdisable [<wpnum>]

The wpdisable command disables a watchpoint. If <wpnum> is specified, only the requested watchpoint is disabled, otherwise all watchpoints are disabled. Note that disabling a watchpoint does not delete it, it just temporarily marks the watchpoint as inactive.

Examples:

```
wpdisable 3
```

Disable watchpoint index 3.

wpdisable

Disable all watchpoints.

Back to *Watchpoint Debugger Commands*

wpenable

wpenable [<wpnum>]

The wpenable command enables a watchpoint. If <wpnum> is specified, only the requested watchpoint is enabled, otherwise all watchpoints are enabled.

Examples:

wpenable 3

Enable watchpoint index 3.

wpenable

Enable all watchpoints.

Back to *Watchpoint Debugger Commands*

wplist

wplist

The wplist command lists all the current watchpoints, along with their index and any conditions or actions attached to them.

Back to *Watchpoint Debugger Commands*

Registerpoints Debugger Commands

You can also type **help <command>** for further details on each command in the MAME Debugger interface.

rpset – sets a registerpoint to trigger on <condition>
rpclear – clears a given registerpoint or all if no <rpnum> specified

rpdisable – disabled a given registerpoint or all if no <rpnum> specified
rpenable – enables a given registerpoint or all if no <rpnum> specified
rplist – lists all the registerpoints

rpset

rp[set] {<condition>}[,<action>]]

Sets a new registerpoint which will be triggered when <condition> is met. The condition must be specified between curly braces to prevent the condition from being evaluated as an assignment.
The optional <action> parameter provides a command that is executed whenever the registerpoint is hit. Note that you may need to embed the action within braces { } in order to prevent commas and semicolons from being interpreted as applying to the rpset command itself.
Each registerpoint that is set is assigned an index which can be used in other registerpoint commands to reference this registerpoint.

Examples:

rp {PC==0150}

Set a registerpoint that will halt execution whenever the PC register equals 0x150.

temp0=0; rp {PC==0150},{temp0++; g}

Set a registerpoint that will increment the variable temp0 whenever the PC register equals 0x0150.

rp {temp0==5}

Set a registerpoint that will halt execution whenever the temp0 variable equals 5.

Back to *Registerpoints Debugger Commands*

rpclear

rpclear [<rpnum>]

The rpclear command clears a registerpoint. If <rpnum> is specified, only the requested registerpoint is cleared, otherwise all registerpoints are cleared.

Examples:

rpclear 3

Clear registerpoint index 3.

```
rpclear
```

Clear all registerpoints.

Back to *Registerpoints Debugger Commands*

rpdisable

rpdisable [<rpnum>]

The rpdisable command disables a registerpoint. If <rpnum> is specified, only the requested registerpoint is disabled, otherwise all registerpoints are disabled. Note that disabling a registerpoint does not delete it, it just temporarily marks the registerpoint as inactive.

Examples:

```
rpdisable 3
```

Disable registerpoint index 3.

```
rpdisable
```

Disable all registerpoints.

Back to *Registerpoints Debugger Commands*

rpenable

rpenable [<rpnum>]

The rpenable command enables a registerpoint. If <rpnum> is specified, only the requested registerpoint is enabled, otherwise all registerpoints are enabled.

Examples:

```
rpenable 3
```

Enable registerpoint index 3.

```
rpenable
```

Enable all registerpoints.

Back to *Registerpoints Debugger Commands*

rplist

rplist

The rplist command lists all the current registerpoints, along with their index and any actions attached to them.

Back to *Registerpoints Debugger Commands*

Code Annotation Debugger Commands

You can also type **help <command>** for further details on each command in the MAME Debugger interface.

comadd – adds a comment to the disassembled code at given address
comdelete – removes a comment from the given address
comsave – save the current comments to a file
comlist – print currently available comments from file
commit – gives a bulk comadd then comsave command

comadd

comadd[//] <address>,<comment>

Adds a string <comment> to the disassembled code at <address>. The shortcut for this command is simply '//'

Examples:

comadd 0, hello world.

Adds the comment 'hello world.' to the code at address 0x0

// 10, undocumented opcode!

Adds the comment 'undocumented opcode!' to the code at address 0x10

comdelete

comdelete

Deletes the comment at the specified memory offset. The comment which is deleted is in the currently active memory bank.

Examples:

comdelete 10

Deletes the comment at code address 0x10 (using the current memory bank settings)

comsave

comsave

Saves the working comments to the driver's XML comment file.

Examples:

comsave

Saves the comments to the driver's comment file

comlist

comlist

Prints the currently available comment file in human readable form in debugger output window.

Examples:

comlist

Shows currently available comments.

commit

commit[/*] <address>,<comment>

Adds a string <comment> to the disassembled code at <address> then saves to file. Basically same as comadd + comsave via a single line.
The shortcut for this command is simply '/*'

Examples:

    commit 0, hello world.

Adds the comment 'hello world.' to the code at address 0x0

    /* 10, undocumented opcode!

Adds the comment 'undocumented opcode!' to the code at address 0x10

Cheat Debugger Commands

You can also type **help <command>** for further details on each command in the MAME Debugger interface.

cheatinit – initialize the cheat search to the selected memory area
cheatrange – add to the cheat search the selected memory area
cheatnext – continue cheat search comparing with the last value
cheatnextf – continue cheat search comparing with the first value
cheatlist – show the list of cheat search matches or save them to <filename>
cheatundo – undo the last cheat search (state only)

cheatinit

    cheatinit [<sign><width><swap>,[<address>,<length>[,<cpu>]]]

The cheatinit command initializes the cheat search to the selected memory area.

If no parameter is specified the cheat search is initialized to all changeable memory of the main cpu.

<sign> can be s(signed) or u(unsigned)
<width> can be b(8 bit), w(16 bit), d(32 bit) or q(64 bit)
<swap> append s for swapped search

Examples:

    cheatinit ub,0x1000,0x10

Initialize the cheat search from 0x1000 to 0x1010 of the first CPU.

cheatinit sw,0x2000,0x1000,1

Initialize the cheat search with width of 2 byte in signed mode from 0x2000 to 0x3000 of the second CPU.

cheatinit uds,0x0000,0x1000

Initialize the cheat search with width of 4 byte swapped from 0x0000 to 0x1000.

Back to *Cheat Debugger Commands*

cheatrange

cheatrange <address>,<length>

The cheatrange command adds the selected memory area to the cheat search.

Before using cheatrange it is necessary to initialize the cheat search with cheatinit.

Examples:

cheatrange 0x1000,0x10

Add the bytes from 0x1000 to 0x1010 to the cheat search.

Back to *Cheat Debugger Commands*

cheatnext

cheatnext <condition>[,<comparisonvalue>]

The cheatnext command will make comparisons with the last search matches.

Possible <condition>:

all

No <comparisonvalue> needed.

Use to update the last value without changing the current matches.

equal [eq]

Without <comparisonvalue> search for all bytes that are equal to the last search.
With <comparisonvalue> search for all bytes that are equal to the <comparisonvalue>.

notequal [ne]

Without <comparisonvalue> search for all bytes that are not equal to the last search.
With <comparisonvalue> search for all bytes that are not equal to the <comparisonvalue>.

decrease [de, +]

Without <comparisonvalue> search for all bytes that have decreased since the last search.
With <comparisonvalue> search for all bytes that have decreased by the <comparisonvalue> since the last search.

increase [in, -]

Without <comparisonvalue> search for all bytes that have increased since the last search.
With <comparisonvalue> search for all bytes that have increased by the <comparisonvalue> since the last search.

decreaseorequal [deeq]

No <comparisonvalue> needed.

Search for all bytes that have decreased or have same value since the last search.

increaseorequal [ineq]

No <comparisonvalue> needed.

Search for all bytes that have decreased or have same value since the last search.

smallerof [lt]

Without <comparisonvalue> this condition is invalid
With <comparisonvalue> search for all bytes that are smaller than the <comparisonvalue>.

greaterof [gt]

Without <comparisonvalue> this condition is invalid
With <comparisonvalue> search for all bytes that are larger than the <comparisonvalue>.

changedby [ch, ~]

Without <comparisonvalue> this condition is invalid
With <comparisonvalue> search for all bytes that have changed by the <comparisonvalue> since the last search.

Examples:

cheatnext increase

Search for all bytes that have increased since the last search.

cheatnext decrease, 1

Search for all bytes that have decreased by 1 since the last search.

Back to *Cheat Debugger Commands*

cheatnextf

cheatnextf <condition>[,<comparisonvalue>]

The cheatnextf command will make comparisons with the initial search.

Possible <condition>:

all

No <comparisonvalue> needed.

Use to update the last value without changing the current matches.

equal [eq]

Without <comparisonvalue> search for all bytes that are equal to the initial search.
With <comparisonvalue> search for all bytes that are equal to the <comparisonvalue>.

notequal [ne]

Without <comparisonvalue> search for all bytes that are not equal to the initial search.
With <comparisonvalue> search for all bytes that are not equal to the <comparisonvalue>.

decrease [de, +]

Without <comparisonvalue> search for all bytes that have decreased since the initial search.
With <comparisonvalue> search for all bytes that have decreased by the <comparisonvalue> since the initial search.

increase [in, -]

Without <comparisonvalue> search for all bytes that have increased since the initial search.

With <comparisonvalue> search for all bytes that have increased by the <comparisonvalue> since the initial search.

decreaseorequal [deeq]

No <comparisonvalue> needed.

Search for all bytes that have decreased or have same value since the initial search.

increaseorequal [ineq]

No <comparisonvalue> needed.

Search for all bytes that have decreased or have same value since the initial search.

smallerof [lt]

Without <comparisonvalue> this condition is invalid.
With <comparisonvalue> search for all bytes that are smaller than the <comparisonvalue>.

greaterof [gt]

Without <comparisonvalue> this condition is invalid.
With <comparisonvalue> search for all bytes that are larger than the <comparisonvalue>.

changedby [ch, ~]

Without <comparisonvalue> this condition is invalid
With <comparisonvalue> search for all bytes that have changed by the <comparisonvalue> since the initial search.

Examples:

cheatnextf increase

Search for all bytes that have increased since the initial search.

cheatnextf decrease, 1

Search for all bytes that have decreased by 1 since the initial search.

Back to *Cheat Debugger Commands*

cheatlist

cheatlist [<filename>]

Without <filename> show the list of matches in the debug console.
With <filename> save the list of matches in basic XML format to <filename>.

Examples:

    cheatlist

Show the current matches in the debug console.

    cheatlist cheat.txt

Save the current matches in XML format to cheat.txt.

Back to *Cheat Debugger Commands*

cheatundo

    cheatundo

Undo the results of the last search.

The undo command has no effect on the last value.

Examples:

    cheatundo

Undo the last search (state only).

Back to *Cheat Debugger Commands*

Image Debugger Commands

You can also type **help <command>** for further details on each command in the MAME Debugger interface.

images – lists all image devices and mounted files
mount – mounts file to named device
unmount – unmounts file from named device

images

images

Used to display list of available image devices.

Examples:

images

Show list of devices and mounted files for current driver.

mount

mount <device>,<filename>

Mount <filename> to image <device>.

<filename> can be softlist item or full path to file.

Examples:

mount cart,aladdin

Mounts softlist item aladdin on cart device.

unmount

unmount <device>

Unmounts file from image <device>.

Examples:

unmount cart

Unmounts any file mounted on device named cart.

Debugger Expressions Guide

Expressions can be used anywhere a numeric parameter is expected. The syntax for expressions is very close to standard C-style syntax with full operator ordering and parentheses. There are a few operators missing (notably the trinary ? : operator), and a few new ones (memory accessors). The table below lists all the operators in their order, highest precedence operators first.

( ) : standard parentheses
++ – : postfix increment/decrement
++ – ~ ! - + b@ w@ d@ q@ : prefix inc/dec, binary NOT, logical NOT, unary +/-, memory access
* / % : multiply, divide, modulus
+ - : add, subtract
<< >> : shift left/right
< <= > >= : less than, less than or equal, greater than, greater than or equal
== != : equal, not equal
& : binary AND
^ : binary XOR
| : binary OR
&& : logical AND
|| : logical OR
= *= /= %= += -= <<= >>= &= |= ^= : assignment
, : separate terms, function parameters

Differences from C Behaviors

- First, all math is performed on full 64-bit unsigned values, so things like **a < 0** won't work as expected.
- Second, the logical operators **&&** and || do not have short-circuit properties – both halves are always evaluated.
- Finally, the new memory operators work like this:
  - **b!<addr>** refers to the byte at <addr> but does *NOT* suppress side effects such as reading a mailbox clearing the pending flag, or reading a FIFO removing an item.
  - **b@<addr>** refers to the byte at <addr> while suppressing side effects.
  - Similarly, **w@** and **w!** refer to a *word* in memory, **d@** and **d!** refer to a *dword* in memory, and **q@** and **q!** refer to a *qword* in memory.

    The memory operators can be used as both lvalues and rvalues, so you can write b@100 = ff to store a byte in memory. By default these operators read from the program memory space, but you can override that by prefixing them with a 'd' or an 'i'.

    As such, **dw@300** refers to data memory word at address 300 and **id@400** refers to an I/O memory dword at address 400.

CHAPTER

SEVEN

MAME EXTERNAL TOOLS

This section covers various extra tools that come with your MAME distribution (e.g. *imgtool*)

Imgtool - *A generic image manipulation tool for MAME*

Imgtool is a tool for the maintenance and manipulation of disk and other types of images that MAME users need to deal with. Functions include retrieving and storing files and CRC checking/validation.

Imgtool is part of the MAME project. It shares large portions of code with MAME, and its existence would not be if it were not for MAME. As such, the distribution terms are the same as MAME. Please read the MAME license thoroughly.

Using Imgtool

Imgtool is a command line program that contains several "subcommands" that actually do all of the work. Most commands are invoked in a manner along the lines of this:

imgtool <subcommand> <format> <image> ...

- **<subcommand>** is the name of the subcommand
- **<format>** is the format of the image
- **<image>** is the filename of the image

Example usage: imgtool dir coco_jvc_rsdos myimageinazip.zip

imgtool get coco_jvc_rsdos myimage.dsk myfile.bin mynewfile.txt

imgtool getall coco_jvc_rsdos myimage.dsk

Further details vary with each subcommand. Also note that not all subcommands are applicable or supported for different image formats.

Imgtool Subcommands

create

imgtool create <format> <imagename> [–(createoption)=value]

- <format> is the image format, e.g. coco_jvc_rsdos
- <imagename> is the image filename; can specify a ZIP file for image name

Creates an image

dir

imgtool dir <format> <imagename> [path]

- <format> is the image format, e.g. coco_jvc_rsdos
- <imagename> is the image filename; can specify a ZIP file for image name

Lists the contents of an image

get

imgtool get <format> <imagename> <filename> [newname] [–filter=filter] [–fork=fork]

- <format> is the image format, e.g. coco_jvc_rsdos
- <imagename> is the image filename; can specify a ZIP file for image name

Gets a single file from an image

put

imgtool put <format> <imagename> <filename>... <destname> [–(fileoption)==value] [–filter=filter] [–fork=fork]

- <format> is the image format, e.g. coco_jvc_rsdos
- <imagename> is the image filename; can specify a ZIP file for image name

Puts a single file on an image (wildcards supported)

getall

imgtool getall <format> <imagename> [path] [–filter=filter]

- <format> is the image format, e.g. coco_jvc_rsdos
- <imagename> is the image filename; can specify a ZIP file for image name

Gets all files off an image

del

imgtool del <format> <imagename> <filename>...

- <format> is the image format, e.g. coco_jvc_rsdos
- <imagename> is the image filename; can specify a ZIP file for image name

Deletes a file on an image

mkdir

imgtool mkdir <format> <imagename> <dirname>

- <format> is the image format, e.g. coco_jvc_rsdos
- <imagename> is the image filename; can specify a ZIP file for image name

Creates a subdirectory on an image

rmdir

imgtool rmdir <format> <imagename> <dirname>...

- <format> is the image format, e.g. coco_jvc_rsdos
- <imagename> is the image filename; can specify a ZIP file for image name

Deletes a subdirectory on an image

readsector

imgtool readsector <format> <imagename> <track> <head> <sector> <filename>

- <format> is the image format, e.g. coco_jvc_rsdos
- <imagename> is the image filename; can specify a ZIP file for image name

Read a sector on an image and output it to specified <filename>

writesector

imgtool writesector <format> <imagename> <track> <head> <sector> <filename>

- <format> is the image format, e.g. coco_jvc_rsdos
- <imagename> is the image filename; can specify a ZIP file for image name

Write a sector to an image from specified <filename>

identify

- <format> is the image format, e.g. coco_jvc_rsdos
- <imagename> is the image filename; can specify a ZIP file for image name

imgtool identify <imagename>

listformats

Lists all image file formats supported by imgtool

listfilters

Lists all filters supported by imgtool

listdriveroptions

imgtool listdriveroptions <format>

- <format> is the image format, e.g. coco_jvc_rsdos

Lists all format-specific options for the 'put' and 'create' commands

Imgtool Filters

Filters are a means to process data being written into or read out of an image in a certain way. Filters can be specified on the get, put, and getall commands by specifying –filter=xxxx on the command line. Currently, the following filters are supported:

ascii

Translates end-of-lines to the appropriate format

cocobas

Processes tokenized TRS-80 Color Computer (CoCo) BASIC programs

dragonbas

Processes tokenized Tano/Dragon Data Dragon 32/64 BASIC programs

macbinary

Processes Apple MacBinary-formatted (merged forks) files

vzsnapshot

[todo: VZ Snapshot? Find out what this is....]

vzbas

Processes Laser/VZ Tokenized Basic Files

thombas5

Thomson MO5 w/ BASIC 1.0, Tokenized Files (read-only, auto-decrypt)

thombas7

Thomson TO7 w/ BASIC 1.0, Tokenized Files (read-only, auto-decrypt)

thombas128

Thomson w/ BASIC 128/512, Tokenized Files (read-only, auto-decrypt)

thomcrypt

Thomson BASIC, Protected file encryption (no tokenization)

bm13bas

Basic Master Level 3 Tokenized Basic Files

Imgtool Format Info

Amiga floppy disk image (OFS/FFS format) - (*amiga_floppy*)

Driver specific options for module 'amiga_floppy':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–density	dd/hd	Density
–filesystem	ofs/ffs	File system
–mode	none/intl/dirc	File system options

Apple][DOS order disk image (ProDOS format) - (*apple2_do_prodos_525*)

Driver specific options for module 'apple2_do_prodos_525':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–heads	1	Heads
–tracks	35	Tracks
–sectors	16	Sectors
–sectorlength	256	Sector Bytes
–firstsectorid	0	First Sector

Apple][Nibble order disk image (ProDOS format) - (*apple2_nib_prodos_525*)

Driver specific options for module 'apple2_nib_prodos_525':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–heads	1	Heads
–tracks	35	Tracks
–sectors	16	Sectors
–sectorlength	256	Sector Bytes
–firstsectorid	0	First Sector

Apple][ProDOS order disk image (ProDOS format) - (*apple2_po_prodos_525*)

Driver specific options for module 'apple2_po_prodos_525':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–heads	1	Heads
–tracks	35	Tracks
–sectors	16	Sectors
–sectorlength	256	Sector Bytes
–firstsectorid	0	First Sector

Apple][gs 2IMG disk image (ProDOS format) - (*apple35_2img_prodos_35*)

Driver specific options for module 'apple35_2img_prodos_35':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–heads	1-2	Heads
–tracks	80	Tracks
–sectorlength	512	Sector Bytes
–firstsectorid	0	First Sector

Apple DiskCopy disk image (Mac HFS Floppy) - (*apple35_dc_mac_hfs*)

Driver specific options for module 'apple35_dc_mac_hfs':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–heads	1-2	Heads
–tracks	80	Tracks
–sectorlength	512	Sector Bytes
–firstsectorid	0	First Sector

Apple DiskCopy disk image (Mac MFS Floppy) - (*apple35_dc_mac_mfs*)

Driver specific options for module 'apple35_dc_mac_mfs':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–heads	1-2	Heads
–tracks	80	Tracks
–sectorlength	512	Sector Bytes
–firstsectorid	0	First Sector

Apple DiskCopy disk image (ProDOS format) - (*apple35_dc_prodos_35*)

Driver specific options for module 'apple35_dc_prodos_35':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–heads	1-2	Heads
–tracks	80	Tracks
–sectorlength	512	Sector Bytes
–firstsectorid	0	First Sector

Apple raw 3.5" disk image (Mac HFS Floppy) - (*apple35_raw_mac_hfs*)

Driver specific options for module 'apple35_raw_mac_hfs':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–heads	1-2	Heads
–tracks	80	Tracks
–sectorlength	512	Sector Bytes
–firstsectorid	0	First Sector

Apple raw 3.5" disk image (Mac MFS Floppy) - (*apple35_raw_mac_mfs*)

Driver specific options for module 'apple35_raw_mac_mfs':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–heads	1-2	Heads
–tracks	80	Tracks
–sectorlength	512	Sector Bytes
–firstsectorid	0	First Sector

Apple raw 3.5” disk image (ProDOS format) - (*apple35_raw_prodos_35*)

Driver specific options for module ‘apple35_raw_prodos_35’:

No image specific file options

Image specific creation options (usable on the ‘create’ command):

Option	Allowed values	Description
–heads	1-2	Heads
–tracks	80	Tracks
–sectorlength	512	Sector Bytes
–firstsectorid	0	First Sector

CoCo DMK disk image (OS-9 format) - (*coco_dmk_os9*)

Driver specific options for module ‘coco_dmk_os9’:

No image specific file options

Image specific creation options (usable on the ‘create’ command):

Option	Allowed values	Description
–heads	1-2	Heads
–tracks	35-255	Tracks
–sectors	1-18	Sectors
–sectorlength	128/256/512/1024/2048/4096/8192	Sector Bytes
–interleave	0-17	Interleave
–firstsectorid	0-1	First Sector

CoCo DMK disk image (RS-DOS format) - (*coco_dmk_rsdos*)

Driver specific options for module ‘coco_dmk_rsdos’:

Image specific file options (usable on the ‘put’ command):

Option	Allowed values	Description
–ftype	basic/data/binary/assembler	File type
–ascii	ascii/binary	Ascii flag

Image specific creation options (usable on the ‘create’ command):

Option	Allowed values	Description
–heads	1-2	Heads
–tracks	35-255	Tracks
–sectors	1-18	Sectors
–sectorlength	128/256/512/1024/2048/4096/8192	Sector Bytes
–interleave	0-17	Interleave
–firstsectorid	0-1	First Sector

CoCo JVC disk image (OS-9 format) - (*coco_jvc_os9*)

Driver specific options for module ‘coco_jvc_os9’:

No image specific file options

Image specific creation options (usable on the ‘create’ command):

Option	Allowed values	Description
–heads	1-2	Heads
–tracks	35-255	Tracks
–sectors	1-255	Sectors
–sectorlength	128/256/512/1024	Sector Bytes
–firstsectorid	0-1	First Sector

CoCo JVC disk image (RS-DOS format) - (*coco_jvc_rsdos*)

Driver specific options for module 'coco_jvc_rsdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	basic/data/binary/assembler	File type
–ascii	ascii/binary	Ascii flag

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–heads	1-2	Heads
–tracks	35-255	Tracks
–sectors	1-255	Sectors
–sectorlength	128/256/512/1024	Sector Bytes
–firstsectorid	0-1	First Sector

CoCo OS-9 disk image (OS-9 format) - (*coco_os9_os9*)

Driver specific options for module 'coco_os9_os9':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–heads	1-2	Heads
–tracks	35-255	Tracks
–sectors	1-255	Sectors
–sectorlength	128/256/512/1024	Sector Bytes
–firstsectorid	1	First Sector

CoCo VDK disk image (OS-9 format) - (*coco_vdk_os9*)

Driver specific options for module 'coco_vdk_os9':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–heads	1-2	Heads
–tracks	35-255	Tracks
–sectors	18	Sectors
–sectorlength	256	Sector Bytes
–firstsectorid	1	First Sector

CoCo VDK disk image (RS-DOS format) - (*coco_vdk_rsdos*)

Driver specific options for module 'coco_vdk_rsdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	basic/data/binary/assembler	File type
–ascii	ascii/binary	Ascii flag

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–heads	1-2	Heads
–tracks	35-255	Tracks
–sectors	18	Sectors
–sectorlength	256	Sector Bytes
–firstsectorid	1	First Sector

Concept floppy disk image - (*concept*)

Driver specific options for module 'concept':

No image specific file options

No image specific creation options

CopyQM floppy disk image (Basic Master Level 3 format) - (*cqm_bml3*)

Driver specific options for module 'cqm_bml3':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	basic/data/binary/assembler	File type
–ascii	ascii/binary	Ascii flag

No image specific creation options

CopyQM floppy disk image (FAT format) - (*cqm_fat*)

Driver specific options for module 'cqm_fat':

No image specific file options

No image specific creation options

CopyQM floppy disk image (Mac HFS Floppy) - (*cqm_mac_hfs*)

Driver specific options for module 'cqm_mac_hfs':

No image specific file options

No image specific creation options

CopyQM floppy disk image (Mac MFS Floppy) - (*cqm_mac_mfs*)

Driver specific options for module 'cqm_mac_mfs':

No image specific file options

No image specific creation options

CopyQM floppy disk image (OS-9 format) - (*cqm_os9*)

Driver specific options for module 'cqm_os9':

No image specific file options

No image specific creation options

CopyQM floppy disk image (ProDOS format) - (*cqm_prodos_35*)

Driver specific options for module 'cqm_prodos_35':

No image specific file options

No image specific creation options

CopyQM floppy disk image (ProDOS format) - (*cqm_prodos_525*)

Driver specific options for module 'cqm_prodos_525':

No image specific file options

No image specific creation options

CopyQM floppy disk image (RS-DOS format) - (*cqm_rsdos*)

Driver specific options for module 'cqm_rsdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	basic/data/binary/assembler	File type
–ascii	ascii/binary	Ascii flag

No image specific creation options

CopyQM floppy disk image (VZ-DOS format) - (*cqm_vzdos*)

Driver specific options for module 'cqm_vzdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	basic/binary/data	File type
–fname	intern/extern	Filename

No image specific creation options

Cybiko Classic File System - (*cybiko*)

Driver specific options for module 'cybiko':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–flash	AT45DB041/AT45DB081/AT45DB161	Flash Type

Cybiko Xtreme File System - (*cybikoxt*)

Driver specific options for module 'cybikoxt':

No image specific file options

No image specific creation options

D88 Floppy Disk image (Basic Master Level 3 format) - (*d88_bml3*)

Driver specific options for module 'd88_bml3':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	basic/data/binary/assembler	File type
–ascii	ascii/binary	Ascii flag

No image specific creation options

D88 Floppy Disk image (FAT format) - (*d88_fat*)

Driver specific options for module 'd88_fat':

No image specific file options

No image specific creation options

D88 Floppy Disk image (Mac HFS Floppy) - (*d88_mac_hfs*)

Driver specific options for module 'd88_mac_hfs':

No image specific file options

No image specific creation options

D88 Floppy Disk image (Mac MFS Floppy) - (*d88_mac_mfs*)

Driver specific options for module 'd88_mac_mfs':

No image specific file options

No image specific creation options

D88 Floppy Disk image (OS-9 format) - (*d88_os9*)

Driver specific options for module 'd88_os9':

No image specific file options

No image specific creation options

D88 Floppy Disk image (OS-9 format) - (*d88_os9*)

Driver specific options for module 'd88_prodos_35':

No image specific file options

No image specific creation options

D88 Floppy Disk image (ProDOS format) - (*d88_prodos_525*)

Driver specific options for module 'd88_prodos_525':

No image specific file options

No image specific creation options

D88 Floppy Disk image (RS-DOS format) - (*d88_rsdos*)

Driver specific options for module 'd88_rsdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	basic/data/binary/assembler	File type
–ascii	ascii/binary	Ascii flag

No image specific creation options

D88 Floppy Disk image (VZ-DOS format) - (*d88_vzdos*)

Driver specific options for module 'd88_vzdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	basic/binary/data	File type
–fname	intern/extern	Filename

No image specific creation options

DSK floppy disk image (Basic Master Level 3 format) - (*dsk_bml3*)

Driver specific options for module 'dsk_bml3':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	basic/data/binary/assembler	File type
–ascii	ascii/binary	Ascii flag

No image specific creation options

DSK floppy disk image (FAT format) - (*dsk_fat*)

Driver specific options for module 'dsk_fat':

No image specific file options

No image specific creation options

DSK floppy disk image (Mac HFS Floppy) - (*dsk_mac_hfs*)

Driver specific options for module 'dsk_mac_hfs':

No image specific file options

No image specific creation options

DSK floppy disk image (Mac MFS Floppy) - (*dsk_mac_mfs*)

Driver specific options for module 'dsk_mac_mfs':

No image specific file options

No image specific creation options

DSK floppy disk image (OS-9 format) - (*dsk_os9*)

Driver specific options for module 'dsk_os9':

No image specific file options

No image specific creation options

DSK floppy disk image (ProDOS format) - (*dsk_prodos_35*)

Driver specific options for module 'dsk_prodos_35':

No image specific file options

No image specific creation options

DSK floppy disk image (ProDOS format) - (*dsk_prodos_525*)

Driver specific options for module 'dsk_prodos_525':

No image specific file options

No image specific creation options

DSK floppy disk image (RS-DOS format) - (*dsk_rsdos*)

Driver specific options for module 'dsk_rsdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	basic/data/binary/assembler	File type
–ascii	ascii/binary	Ascii flag

No image specific creation options

DSK floppy disk image (VZ-DOS format) - (*dsk_vzdos*)

Driver specific options for module 'dsk_vzdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	basic/binary/data	File type
–fname	intern/extern	Filename

No image specific creation options

Formatted Disk Image (Basic Master Level 3 format) - (*fdi_bml3*)

Driver specific options for module 'fdi_bml3':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	basic/data/binary/assembler	File type
–ascii	ascii/binary	Ascii flag

No image specific creation options

Formatted Disk Image (FAT format) - (*fdi_fat*)

Driver specific options for module 'fdi_fat':

No image specific file options

No image specific creation options

Formatted Disk Image (Mac HFS Floppy) - (*fdi_mac_hfs*)

Driver specific options for module 'fdi_mac_hfs':

No image specific file options

No image specific creation options

Formatted Disk Image (Mac MFS Floppy) - (*fdi_mac_mfs*)

Driver specific options for module 'fdi_mac_mfs':

No image specific file options

No image specific creation options

Formatted Disk Image (OS-9 format) - (*fdi_os9*)

Driver specific options for module 'fdi_os9':

No image specific file options

No image specific creation options

Formatted Disk Image (ProDOS format) - (*fdi_prodos_35*)

Driver specific options for module 'fdi_prodos_35':

No image specific file options

No image specific creation options

Formatted Disk Image (ProDOS format) - (*fdi_prodos_525*)

Driver specific options for module 'fdi_prodos_525':

No image specific file options

No image specific creation options

Formatted Disk Image (RS-DOS format) - (*fdi_rsdos*)

Driver specific options for module 'fdi_rsdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	basic/data/binary/assembler	File type
–ascii	ascii/binary	Ascii flag

No image specific creation options

Formatted Disk Image (VZ-DOS format) - (*fdi_vzdos*)

Driver specific options for module 'fdi_vzdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	basic/binary/data	File type
–fname	intern/extern	Filename

No image specific creation options

HP48 SX/GX memory card - (*hp48*)

Driver specific options for module 'hp48':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–size	32/64/128/256/512/1024/2048/4096	Size in KB

IMD floppy disk image (Basic Master Level 3 format) - (*imd_bml3*)

Driver specific options for module 'imd_bml3':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	basic/data/binary/assembler	File type
–ascii	ascii/binary	Ascii flag

No image specific creation options

IMD floppy disk image (FAT format) - (*imd_fat*)

Driver specific options for module 'imd_fat':

No image specific file options

No image specific creation options

IMD floppy disk image (Mac HFS Floppy) - (*imd_mac_hfs*)

Driver specific options for module 'imd_mac_hfs':

No image specific file options

No image specific creation options

IMD floppy disk image (Mac MFS Floppy) - (*imd_mac_mfs*)

Driver specific options for module 'imd_mac_mfs':

No image specific file options

No image specific creation options

IMD floppy disk image (OS-9 format) - (*imd_os9*)

Driver specific options for module 'imd_os9':

No image specific file options

No image specific creation options

IMD floppy disk image (ProDOS format) - (*imd_prodos_35*)

Driver specific options for module 'imd_prodos_35':

No image specific file options

No image specific creation options

IMD floppy disk image (ProDOS format) - (*imd_prodos_525*)

Driver specific options for module 'imd_prodos_525':

No image specific file options

No image specific creation options

IMD floppy disk image (RS-DOS format) - (*imd_rsdos*)

Driver specific options for module 'imd_rsdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	basic/data/binary/assembler	File type
–ascii	ascii/binary	Ascii flag

No image specific creation options

IMD floppy disk image (VZ-DOS format) - (*imd_vzdos*)

Driver specific options for module 'imd_vzdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	basic/binary/data	File type
–fname	intern/extern	Filename

No image specific creation options

MESS hard disk image - (*mess_hd*)

Driver specific options for module 'mess_hd':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–blocksize	1-2048	Sectors Per Block
–cylinders	1-65536	Cylinders
–heads	1-64	Heads
–sectors	1-4096	Total Sectors
–seclen	128/256/512/1024/2048/4096/8192/16384/32768/65536	Sector Bytes

TI99 Diskette (PC99 FM format) - (*pc99fm*)

Driver specific options for module 'pc99fm':

No image specific file options

No image specific creation options

TI99 Diskette (PC99 MFM format) - (*pc99mfm*)

Driver specific options for module 'pc99mfm':

No image specific file options

No image specific creation options

PC CHD disk image - (*pc_chd*)

Driver specific options for module 'pc_chd':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–cylinders	10/20/30/40/50/60/70/80/90/100/110/120/130/140/150/160/170/180/190/200	Cylinders
–heads	1-16	Heads
–sectors	1-63	Sectors

PC floppy disk image (FAT format) - (*pc_dsk_fat*)

Driver specific options for module 'pc_dsk_fat':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–heads	1-2	Heads
–tracks	40/80	Tracks
–sectors	8/9/10/15/18/36	Sectors

Psion Organiser II Datapack - (*psionpack*)

Driver specific options for module 'psionpack':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–type	OB3/OPL/ODB	file type
–id	0/145-255	File ID

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–size	8k/16k/32k/64k/128k	datapack size
–ram	0/1	EPROM/RAM datapack
–paged	0/1	linear/paged datapack
–protect	0/1	write-protected datapack
–boot	0/1	bootable datapack
–copy	0/1	copyable datapack

Teledisk floppy disk image (Basic Master Level 3 format) - (*td0_bml3*)

Driver specific options for module 'td0_bml3':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	basic/data/binary/assembler	File type
–ascii	ascii/binary	Ascii flag

No image specific creation options

Teledisk floppy disk image (FAT format) - (*td0_fat*)

Driver specific options for module 'td0_fat':

No image specific file options

No image specific creation options

Teledisk floppy disk image (Mac HFS Floppy) - (*td0_mac_hfs*)

Driver specific options for module 'td0_mac_hfs':

No image specific file options

No image specific creation options

Teledisk floppy disk image (Mac MFS Floppy) - (*td0_mac_mfs*)

Driver specific options for module 'td0_mac_mfs':

No image specific file options

No image specific creation options

Teledisk floppy disk image (OS-9 format) - (*td0_os9*)

Driver specific options for module 'td0_os9':

No image specific file options

No image specific creation options

Teledisk floppy disk image (ProDOS format) - (*td0_prodos_35*)

Driver specific options for module 'td0_prodos_35':

No image specific file options

No image specific creation options

Teledisk floppy disk image (ProDOS format) - (*td0_prodos_525*)

Driver specific options for module 'td0_prodos_525':

No image specific file options

No image specific creation options

Teledisk floppy disk image (RS-DOS format) - (*td0_rsdos*)

Driver specific options for module 'td0_rsdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	basic/data/binary/assembler	File type
–ascii	ascii/binary	Ascii flag

No image specific creation options

Teledisk floppy disk image (VZ-DOS format) - (*td0_vzdos*)

Driver specific options for module 'td0_vzdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	basic/binary/data	File type
–fname	intern/extern	Filename

No image specific creation options

Thomson .fd disk image, BASIC format - (*thom_fd*)

Driver specific options for module 'thom_fd':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	auto/B/D/M/A	File type
–format	auto/B/A	Format flag
–comment	(string)	Comment

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–heads	1-2	Heads
–tracks	40/80	Tracks
–density	SD/DD	Density
–name	(string)	Floppy name

Thomson .qd disk image, BASIC format - (*thom_qd*)

Driver specific options for module 'thom_qd':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	auto/B/D/M/A	File type
–format	auto/B/A	Format flag
–comment	(string)	Comment

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–heads	1-2	Heads
–tracks	25	Tracks
–density	SD/DD	Density
–name	(string)	Floppy name

Thomson .sap disk image, BASIC format - (*thom_sap*)

Driver specific options for module 'thom_sap':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	auto/B/D/M/A	File type
–format	auto/B/A	Format flag
–comment	(string)	Comment

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–heads	1	Heads
–tracks	40/80	Tracks
–density	SD/DD	Density
–name	(string)	Floppy name

TI990 Hard Disk - (*ti990hd*)

Driver specific options for module 'ti990hd':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–cylinders	1-2047	Cylinders
–heads	1-31	Heads
–sectors	1-256	Sectors
–bytes per sector	(typically 25256-512 256-512	Bytes Per Sector [Todo: This section is glitched in imgtool]

TI99 Diskette (old MESS format) - (*ti99_old*)

Driver specific options for module 'ti99_old':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–sides	1-2	Sides
–tracks	1-80	Tracks
–sectors	1-36	Sectors (1->9 for SD, 1->18 for DD, 1->36 for HD)
–protection	0-1	Protection (0 for normal, 1 for protected)
–density	Auto/SD/DD/HD	Density

TI99 Harddisk - (*ti99hd*)

Driver specific options for module 'ti99hd':

No image specific file options

No image specific creation options

TI99 Diskette (V9T9 format) - (*v9t9*)

Driver specific options for module 'v9t9':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–sides	1-2	Sides
–tracks	1-80	Tracks
–sectors	1-36	Sectors (1->9 for SD, 1->18 for DD, 1->36 for HD)
–protection	0-1	Protection (0 for normal, 1 for protected)
–density	Auto/SD/DD/HD	Density

Laser/VZ disk image (VZ-DOS format) - (*vtech1_vzdos*)

Driver specific options for module 'vtech1_vzdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
–ftype	basic/binary/data	File type
–fname	intern/extern	Filename

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
–heads	1	Heads
–tracks	40	Tracks
–sectors	16	Sectors
–sectorlength	154	Sector Bytes
–firstsectorid	0	First Sector

[todo: fill out the command structures, describe commands better. These descriptions came from the imgtool.txt file and are barebones]

Castool - *A generic casette image manipulation tool for MAME*

Castool is a tool for the maintenance and manipulation of cassette images that MAME users need to deal with. MAME directly supports .WAV audio formatted images, but many of the existing images out there may come in forms such as .TAP for Commodore 64 tapes, .CAS for Tandy Color Computer tapes, and so forth. Castool will convert these other formats to .WAV for use in MAME.

Castool is part of the MAME project. It shares large portions of code with MAME, and its existence would not be if it were not for MAME. As such, the distribution terms are the same as MAME. Please read the MAME license thoroughly.

Using Castool

Castool is a command line program that contains a simple set of instructions. Commands are invoked in a manner along the lines of this:

castool convert <format> <inputfile> <outputfile>

- **<format>** is the format of the image
- **<inputfile>** is the filename of the image you're converting from
- **<outputfile>** is the filename of the output WAV file

Example usage: castool convert coco zaxxon.cas zaxxon.wav

castool convert cbm arkanoid.tap arkanoid.wav

castool convert ddp mybasicprogram.ddp mybasicprogram.wav

Castool Formats

These are the formats supported by Castool for conversion to .WAV files.

A26

Atari 2600 SuperCharger image

File extension: a26

APF

APF Imagination Machine

File extensions: cas, cpf, apt

ATOM

Acorn Atom

File extensions: tap, csw, uef

BBC

Acorn BBC & Electron

File extensions: csw, uef

CBM

Commodore 8-bit series

File extensions: tap

CDT

Amstrad CPC

File extensions: cdt

CGENIE

EACA Colour Genie

File extensions: cas

COCO

Tandy Radio Shack Color Computer

File extensions: cas

CSW

Compressed Square Wave

File extensions: csw

DDP

Coleco ADAM

File extensions: ddp

FM7

Fujitsu FM-7

File extensions: t77

FMSX

MSX

File extensions: tap, cas

GTP

Elektronika inzenjering Galaksija

File extensions: gtp

HECTOR

Micronique Hector & Interact Family Computer

File extensions: k7, cin, for

JUPITER

Jupiter Cantab Jupiter Ace

File extensions: tap

KC85

VEB Mikroelektronik KC 85

File extensions: kcc, kcb, tap, 853, 854, 855, tp2, kcm, sss

KIM1

MOS KIM-1

File extensions: kim, kim1

LVIV

PK-01 Lviv

File extensions: lvt, lvr, lv0, lv1, lv2, lv3

MO5

Thomson MO-series

File extensions: k5, k7

MZ

Sharp MZ-700

File extensions: m12, mzf, mzt

ORAO

PEL Varazdin Orao

File extensions: tap

ORIC

Tangerine Oric

File extensions: tap

PC6001

NEC PC-6001

File extensions: cas

PHC25

Sanyo PHC-25

File extensions: phc

PMD85

Tesla PMD-85

File extensions: pmd, tap, ptp

PRIMO

Microkey Primo

File extensions: ptp

RKU

UT-88

File extensions: rku

RK8

Mikro-80

File extensions: rk8

RKS

Specialist

File extensions: rks

RKO

Orion

File extensions: rko

RKR

Radio-86RK

File extensions: rk, rkr, gam, g16, pki

RKA

Zavod BRA Apogee BK-01

File extensions: rka

RKM

Mikrosha

File extensions: rkm

RKP

SAM SKB VM Partner-01.01

File extensions: rkp

SC3000

Sega SC-3000

File extensions: bit

SOL20

PTC SOL-20

File extensions: svt

SORCERER

Exidy Sorcerer

File extensions: tape

SORDM5

Sord M5

File extensions: cas

SPC1000

Samsung SPC-1000

File extensions: tap, cas

SVI

Spectravideo SVI-318 & SVI-328

File extensions: cas

TO7

Thomson TO-series

File extensions: k7

TRS8012

TRS-80 Level 2

File extensions: cas

TVC64

Videoton TVC 64

File extensions: cas

TZX

Sinclair ZX Spectrum

File extensions: tzx, tap, blk

VG5K

Philips VG 5000

File extensions: k7

VTECH1

Video Technology Laser 110-310

File extensions: cas

VTECH2

Video Technology Laser 350-700

File extensions: cas

X07

Canon X-07

File extensions: k7, lst, cas

X1

Sharp X1

File extensions: tap

ZX80_O

Sinclair ZX80

File extensions: o, 80

ZX81_P

Sinclair ZX81

File extensions: p, 81

Floptool - *A generic floppy image manipulation tool for MAME*

Floptool is a tool for the maintenance and manipulation of floppy images that MAME users need to deal with. MAME directly supports .WAV audio formatted images, but many of the existing images out there may come in forms such as .TAP for Commodore 64 tapes, .CAS for Tandy Color Computer tapes, and so forth. Castool will convert these other formats to .WAV for use in MAME.

Floptool is part of the MAME project. It shares large portions of code with MAME, and its existence would not be if it were not for MAME. As such, the distribution terms are the same as MAME. Please read the MAME license thoroughly.

Using Floptool

Floptool is a command line program that contains a simple set of instructions. Commands are invoked in a manner along the lines of this:

floptool identify <inputfile> [<inputfile> ...] floptool convert [input_format|auto] output_format <inputfile> <outputile>

- **<format>** is the format of the image
- **<input_format>** is the format of the inputfile, use auto if not known
- **<output_format>** is the format of the converted file
- **<inputfile>** is the filename of the image you're identifying/converting from
- **<outputfile>** is the filename of the converted file

Example usage: floptool convert coco zaxxon.cas zaxxon.wav

floptool convert cbm arkanoid.tap arkanoid.wav

floptool convert ddp mybasicprogram.ddp mybasicprogram.wav

Floptool Formats

These are the formats supported by Floptool for conversion to other formats.

MFI

MAME floppy image

File extension: mfi

DFI

DiscFerret flux dump format

File extensions: dfi

IPF

SPS floppy disk image

File extensions: ipf

MFM

HxC Floppy Emulator floppy disk image

File extensions: mfm

ADF

Amiga ADF floppy disk image

File extensions: adf

ST

Atari ST floppy disk image

File extensions: st

MSA

Atari MSA floppy disk image

File extensions: msa

PASTI

Atari PASTI floppy disk image

File extensions: stx

DSK

CPC DSK format

File extensions: dsk

D88

D88 disk image

File extensions: d77, d88, 1dd

IMD

IMD disk image

File extensions: imd

TD0

Teledisk disk image

File extensions: td0

CQM

CopyQM disk image

File extensions: cqm, cqi, dsk

PC

PC floppy disk image

File extensions: dsk, ima, img, ufi, 360

NASLITE

NASLite disk image

File extensions: img

DC42

DiskCopy 4.2 image

File extensions: dc42

A2_16SECT

Apple II 16-sector disk image

File extensions: dsk, do, po

A2_RWTS18

Apple II RWTS18-type image

File extensions: rti

A2_EDD

Apple II EDD image

File extensions: edd

ATOM

Acorn Atom disk image

File extensions: 40t, dsk

SSD

Acorn SSD disk image

File extensions: ssd, bbc, img

DSD

Acorn DSD disk image

File extensions: dsd

DOS

Acorn DOS disk image

File extensions: img

ADFS_O

Acorn ADFS (OldMap) disk image

File extensions: adf, ads, adm, adl

ADFS_N

Acorn ADFS (NewMap) disk image

File extensions: adf

ORIC_DSK

Oric disk image

File extensions: dsk

APPLIX

Applix disk image

File extensions: raw

HPI

HP9845A floppy disk image

File extensions: hpi

Other tools included with MAME

ledutil.exe/ledutil.sh

On Microsoft Windows, ledutil.exe can take control of your keyboard LEDs to mirror those that were present on some early arcade games (e.g. Asteroids)

Start **ledutil.exe** from the command line to enable LED handling. Run **ledutil.exe -kill** to stop the handler.

On SDLMAME platforms such as Mac OS X and Linux, **ledutil.sh** can be used. Use **ledutil.sh -a** to have it automatically close when you exit SDLMAME.

Developer-focused tools included with MAME

pngcmp

This tool is used in regression testing to compare PNG screenshot results with the runtest.cmd script found in the source archive. This script works only on Microsoft Windows.

nltool

Discrete component conversion tool. Most users will not need to work with this.

nlwav

Discrete component conversion and testing tool. Most users will not need to work with this.

jedutil

PAL/PLA/PLD/GAL dump handling tool. It can convert between the industry-standard JED format and MAME's proprietary packed binary format and it can show logic equations for the types of devices it knows the internal logic of. Most users will not need to work with this.

ldresample

This tool recompresses video data for laserdisc and VHS dumps. Most users will not need to work with this.

ldverify

This tool is used for comparing laserdisc or VHS CHD images with the source AVI. Most users will not need to work with this.

unidasm

Universal disassembler for many of the architectures supported in MAME. Most users will not need to work with this.

CHAPTER

EIGHT

TECHNICAL SPECIFICATIONS

This section covers technical specifications useful to programmers working on MAME's source or working on LUA scripts that run within the MAME framework.

The device_memory_interface

1. Capabilities

The device memory interface provides devices with the capability of creating address spaces, to which address maps can be associated. It's used for any device that provides a (logically) address/data bus other devices can be connected to. It's mainly, but not only, cpus.

The interface allows for an unlimited set of address spaces, numbered with small positive values. The IDs should stay small because they index vectors to keep the lookup fast. Spaces number 0-3 have an associated constant name:

ID	Name
0	AS_PROGRAM
1	AS_DATA
2	AS_IO
3	AS_OPCODES

Spaces 0 and 3, e.g. AS_PROGRAM and AS_OPCODE, are special for the debugger and some CPUs. AS_PROGRAM is use by the debugger and the cpus as the space from with the cpu reads its instructions for the disassembler. When present, AS_OPCODE is used by the debugger and some cpus to read the opcode part of the instruction. What opcode means is device-dependant, for instance for the z80 it's the initial byte(s) which are read with the M1 signal asserted. For the 68000 is means every instruction word plus the PC-relative accesses. The main, but not only, use of AS_OPCODE is to implement hardware decrypting instructions separately from the data.

2. Setup

std::vector<std::pair<int, const address_space_config *>>**memory_space_config**(int spacenum) const

The device must override that method to provide a vector of pairs comprising of a space number and its associated **address_space_config** describing its configuration. Some examples to look up when needed:

- Standard two-space vector: v60_device
- Conditional AS_OPCODE: z80_device
- Inherit config and add a space: m6801_device
- Inherit config and patch a space: tmpz84c011_device

bool **has_configured_map**() const
bool **has_configured_map**(int index) const

The **has_configured_map** method allows to test in the **memory_space_config** method whether an **address_map** has been associated with a given space. That allows to implement optional memory spaces, such as AS_OPCODES in certain cpu cores. The parameterless version tests for space 0.

3. Associating maps to spaces

Associating maps to spaces is done at the machine config level, after the device declaration.

MCFG_DEVICE_ADDRESS_MAP(_space, _map)
MCFG_DEVICE_PROGRAM_MAP(_map)
MCFG_DEVICE_DATA_MAP(_map)
MCFG_DEVICE_IO_MAP(_map)
MCFG_DEVICE_DECRYPTED_OPCODES_MAP(_map)

The generic macro and the four specific ones associate a map to a given space. Address maps associated to non-existing spaces are ignored (no warning given). devcpu.h defines MCFG_CPU_*_MAP aliases to the specific macros.

MCFG_DEVICE_REMOVE_ADDRESS_MAP(_space)

That macro removes a memory map associated to a given space. Useful when removing a map for an optional space in a machine config derivative.

4. Accessing the spaces

address_space &**space**() const
address_space &**space**(int index) const

Returns a given address space post-initialization. The parameterless version tests for AS_PROGRAM/AS_0. Aborts if the space doesn't exist.

bool **has_space**() const
bool **has_space**(int index) const

Indicates whether a given space actually exists. The parameterless version tests for AS_PROGRAM/AS_0.

5. MMU support for disassembler

bool **translate**(int spacenum, int intention, offs_t &address)

Does a logical to physical address translation through the device's MMU. spacenum gives the space number, intention the type of the future access (TRANSLATE_(READ|WRITE|FETCH)(|_USER|_DEBUG)) and address is an inout parameter with the address to translate and its translated version. Should return true if the translation went correctly, false if the address is unmapped.

Note that for some historical reason the device itself must override the virtual method **memory_translate** with the same signature.

The device_rom_interface

1. Capabilities

This interface is designed for devices which expect to have a rom connected to them on a dedicated bus. It's mostly designed for sound chips. Other devices types may be interested but other considerations may make it impratical (graphics decode caching for instance). The interface provides the capability of either connecting a ROM_REGION, connecting an ADDRESS_MAP or dynamically setting up a block of memory as rom. In the region/block cases, banking is automatically handled.

2. Setup

device_rom_interface(const machine_config &mconfig, device_t &device, u8 addrwidth, endianness_t endian = ENDIANNESS_LITTLE, u8 datawidth = 8)

The constructor of the interface wants, in addition to the standard parameters, the address bus width of the dedicated bus. In addition the endianness (if not little endian or byte-sized bus) and data bus width (if not byte) can be provided.

MCFG_DEVICE_ADDRESS_MAP(AS_0, map)

Use that method at machine config time to provide an address map for the bus to connect to. It has priority over a rom region if one is also present.

MCFG_DEVICE_ROM(tag)

Used to select a rom region to use if a device address map is not given. Defaults to DEVICE_SELF, e.g. the device tag.

ROM_REGION(length, tag, flags)

If a rom region with a tag as given with **MCFG_DEVICE_ROM** if present, or identical to the device tag otherwise, is provided in the rom description for the system, it will be automatically picked up as the connected rom. An address map has priority over the region if present in the machine config.

void **set_rom_endianness**(endianness_t endian)

void **set_rom_data_width**(u8 width)
void **set_rom_addr_width**(u8 width)

These methods, intended for generic devices with indefinite hardware specifications, override the endianness, data bus width and address bus width assigned through the constructor. They must be called from within the device before **config_complete** time.

void **set_rom**(const void *base, u32 size);

At any time post- **interface_pre_start**, a memory block can be setup as the connected rom with that method. It overrides any previous setup that may have been provided. It can be done multiple times.

3. Rom access

u8 **read_byte**(offs_t byteaddress)
u16 **read_word**(offs_t byteaddress)
u32 **read_dword**(offs_t byteaddress)
u64 **read_qword**(offs_t byteaddress)

These methods provide read access to the connected rom. Out-of-bounds access results in standard unmapped read logerror messages.

4. Rom banking

If the rom region or the memory block in set_rom is larger than the address bus, banking is automatically setup.

void **set_rom_bank**(int bank)

That method selects the current bank number.

5. Caveats

Using that interface makes the device derive from **device_memory_interface**. If the device wants to actually use the memory interface for itself, remember that AS_0/AS_PROGRAM is used by the rom interface, and don't forget to upcall **memory_space_config**.

For devices which have outputs that can be used to address ROMs but only to forward the data to another device for processing, it may be helpful to disable the interface when it is not required. This can be done by overriding **memory_space_config** to return an empty vector.

The device_disasm_interface and the disassemblers

1. Capabilities

The disassemblers are classes that provide disassembly and opcode meta-information for the cpu cores and **unidasm**. The **device_disasm_interface** connects a cpu core with its disassembler.

2. The disassemblers

2.1. Definition

A disassembler is a class that derives from **util::disasm_interface**. It then has two required methods to implement, **opcode_alignment** and **disassemble**, and 6 optional, **interface_flags**, **page_address_bits**, **pc_linear_to_real**, **pc_real_to_linear**, and one with four possible variants, **decrypt8/16/32/64**.

2.2. opcode_alignment

u32 **opcode_alignment**() const

Returns the required alignment of opcodes by the cpu, in PC-units. In other words, the required alignment for the PC register of the cpu. Tends to be 1 (almost everything), 2 (68000...), 4 (mips, ppc...), which an exceptional 8 (tms 32082 parallel processor) and 16 (tms32010, instructions are 16-bits aligned and the PC targets bits). It must be a power-of-two or things will break.

Note that processors like the tms32031 which have 32-bits instructions but where the PC targets 32-bits values have an alignment of 1.

2.3. disassemble

offs_t **disassemble**(std::ostream &stream, offs_t pc, const data_buffer &opcodes, const data_buffer &params)

This is the method where the real work is done. This method must disassemble the instruction at address *pc* and write the result to *stream*. The values to decode are retrieved from the *opcode* buffer. A **data_buffer** object offers four accessor methods:

u8 util::disasm_interface::data_buffer::**r8** (offs_t pc) const
u16 util::disasm_interface::data_buffer::**r16**(offs_t pc) const
u32 util::disasm_interface::data_buffer::**r32**(offs_t pc) const
u64 util::disasm_interface::data_buffer::**r64**(offs_t pc) const

They read the data at a given address and take endianness and nonlinear PCs for larger-than-bus-width accesses. The debugger variant also caches the read data in one block, so for that reason one should not read data too far from the base pc (e.g. stay within 16K or so, careful when trying to follow indirect accesses).

A number of CPUs have an external signal that splits fetches into an opcode part and a parameter part. This is for instance the M1 signal of the z80 or the SYNC signal of the 6502. Some systems present different values to the cpu depending on whether that signal is active, usually for protection purposes. On these cpus the opcode part should be

read from the *opcode* buffer, and the parameter part from the *params* buffer. They will or will not be the same buffer depending on the system itself.

The method returns the size of the instruction in PC units, with a maximum of 65535. In addition, if possible, the disassembler should give some meta-information about the opcode by OR-ing in into the result:

- **STEP_OVER** for subroutine calls or auto-decrementing loops. If there is some delay slots, also OR with **step_over_extra**(n) where n is the number of instruction slots.
- **STEP_OUT** for the return-from-subroutine instructions

In addition, to indicated that these flags are supported, OR the result with **SUPPORTED**. An annoying number of disassemblers lies about that support (e.g. they do a or with **SUPPORTED** without even generating the **STEP_OVER** or **STEP_OUT** information). Don't do that, it breaks the step over/step out functionality of the debugger.

2.4. interface_flags

u32 **interface_flags**() const

That optional method indicates specifics of the disassembler. Default of zero is correct most of the time. Possible flags, which need to be OR-ed together, are:

- **NONLINEAR_PC**: stepping to the next opcode or the next byte of the opcode is not adding one to pc. Used for old LFSR-based PCs.
- **PAGED**: PC wraps at a page boundary
- **PAGED2LEVEL**: not only PC wraps at some kind of page boundary, but there are two levels of paging
- **INTERNAL_DECRYPTION**: there is some decryption tucked between reading from AS_PROGRAM and the actual disassembler
- **SPLIT_DECRYPTION**: there is some decryption tucked between reading from AS_PROGRAM and the actual disassembler, and that decryption is different for opcodes and parameters

Note that in practice non-linear pc systems are also paged, that **PAGED2LEVEL** implies **PAGED**, and that **SPLIT_DECRYPTION** implies **DECRYPTION**.

2.5. pc_linear_to_real and pc_real_to_linear

offs_t **pc_linear_to_real**(offs_t pc) const
offs_t **pc_real_to_linear**(offs_t pc) const

These methods should be present only when **NONLINEAR_PC** is set in the interface flags. They must convert pc to and from a value to a linear domain where the instruction parameters and next instruction are reached by incrementing the value. **pc_real_to_linear** converts to that domain, **pc_linear_to_real** converts back from that domain.

2.6. page_address_bits

u32 **page_address_bits**() const

Present on when **PAGED** or **PAGED2LEVEL** is set, gives the number of address bits in the lowest page.

2.7. page2_address_bits

u32 **page2_address_bits**() const

Present on when **PAGED2LEVEL** is set, gives the number of address bits in the upper page.

2.8. decryptnn

u8 **decrypt8** (u8 value, offs_t pc, bool opcode) const
u16 **decrypt16**(u16 value, offs_t pc, bool opcode) const
u32 **decrypt32**(u32 value, offs_t pc, bool opcode) const
u64 **decrypt64**(u64 value, offs_t pc, bool opcode) const

One of these must be defined when **INTERNAL_DECRYPTION** or **SPLIT_DECRYPTION** is set. The chosen one is the one which takes what **opcode_alignment** represents in bytes.

That method decrypts a given value read from address pc (from AS_PROGRAM) and gives the result which will be passed to the disassembler. In the split decryption case, opcode indicates whether we're in the opcode (true) or parameter (false) part of the instruction.

3. Disassembler interface, device_disasm_interface

3.1. Definition

A CPU core derives from **device_disasm_interface** through **cpu_device**. One method has to be implemented, **create_disassembler**.

3.2. create_disassembler

util::disasm_interface ***create_disassembler**()

That method must return a pointer to a newly allocated disassembler object. The caller takes ownership and handles the lifetime.

THis method will be called at most one in the lifetime of the cpu object.

4. Disassembler configuration and communication

Some disassemblers need to be configured. Configuration can be unchanging (static) for the duration of the run (cpu model type for instance) or dynamic (state of a flag or a user preference). Static configuration can be done through either (a) parameter(s) to the disassembler constructor, or through deriving a main disassembler class. If the information is short and its semantics obvious (like a model name), feel free to use a parameter. Otherwise derive the class.

Dynamic configuration must be done by first defining a nested public struct called "config" in the disassembler, with virtual destructor and pure virtual methods to pull the required information. A pointer to that struct should be passed to the disassembler constructor. The cpu core should then add a derivation from that config struct and implement the methods. Unidasm will have to derive a small class from the config class to give the information.

5. Missing stuff

There currently is no way for the debugger GUI to add per-core configuration. It is needed for in particular the s2650 and the saturn cores. It should go through the cpu core class itself, since it's pulled from the config struct.

There is support missing in unidasm for per-cpu configuration. That's needed for a lot of things, see the unidasm source code for the current list ("Configuration missing" comments).

The new floppy subsystem

1. Introduction

The new floppy subsystem aims at emulating the behaviour of floppies and floppy controllers at a level low enough that protections work as a matter of course. It reaches its goal by following the real hardware configuration:

- a floppy image class keeps in memory the magnetic state of the floppy surface and its physical characteristics
- an image handler class talks with the floppy image class to simulate the floppy drive, providing all the signals you have on a floppy drive connector
- floppy controller devices talk with the image handler and provide the register interfaces to the host we all know and love
- format handling classes are given the task of statelessly converting to and from an on-disk image format to the in-memory magnetic state format the floppy image class manages

2. Floppy storage 101

2.1. Floppy disk

A floppy disk is a disc that stores magnetic orientations on their surface disposed in a series on concentric circles called tracks or cylinders [1]. Its main characteristics are its size (goes from a diameter of around 2.8" to 8") , its number of writable sides (1 or 2) and its magnetic resistivity. The magnetic resistivity indicates how close magnetic orientation changes can happen and the information kept. That's one third of what defines the term "density" that is so often used for floppies (the other two are floppy drive head size and bit-level encoding).

The magnetic orientations are always binary, e.g. they're one way or the opposite, there's no intermediate state. Their direction can either be tengentially to the track, e.g in the direction or opposite to the rotation, or in the case of perpendicular recording the direction is perpendicular to the disc surface (hence the name). Perpendicular recording allows for closer orientation changes by writing the magnetic information more deeply, but arrived late in the technology lifetime. 2.88Mb disks and the floppy children (Zip drives, etc) used perpendicular recording. For simulation purposes the direction is not important, only the fact that only two orientations are possible is. Two more states are possible though: a portion of a track can be demagnetized (no orientation) or damaged (no orientation and can't be written to).

A specific position in the disk rotation triggers an index pulse. That position can be detected through a hole in the surface (very visible in 5.25" and 3" floppies for instance) or through a specific position of the rotating center (3.5" floppies, perhaps others). This index pulse is used to designate the beginning of the track, but is not used by every system. Older 8" floppies have multiple index holes used to mark the beginning of sectors (called hard sectoring) but one of them is positioned differently to be recognized as the track start, and the others are at fixed positions relative to the origin one.

[1] Cylinder is a hard-drive term somewhat improperly used for floppies. It comes from the fact that hard-drives are similar to floppies but include a series of stacked disks with a read/write head on each. The heads are physically linked and all point to the same circle on every disk at a given time, making the accessed area look like a cylinder. Hence the name.

2.2. Floppy drive

A floppy drive is what reads and writes a floppy disk. It includes an assembly capable of rotating the disk at a fixed speed and one or two magnetic heads tied to a positioning motor to access the tracks.

The head width and positioning motor step size decides how many tracks are written on the floppy. Total number of tracks goes from 32 to 84 depending on the floppy and drive, with the track 0 being the most exterior (longer) one of the concentric circles, and the highest numbered the smallest interior circle. As a result the tracks with the lowest numbers have the lowest physical magnetic orientation density, hence the best reliability. Which is why important and/or often changed structures like the boot block or the fat allocation table are at track 0. That is also where the terminology "stepping in" to increase the track number and "stepping out" to decrease it comes from. The number of tracks available is the second part of what is usually behind the term "density".

A sensor detects when the head is on track 0 and the controller is not supposed to try to go past it. In addition physical blocks prevent the head from going out of the correct track range. Some systems (Apple II, some C64) do not take the track 0 sensor into account and just wham the head against the track 0 physical block, giving a well-known crash noise and eventually damaging the head alignment.

Also, some systems (Apple II and C64 again) have direct access to the phases of the head positioning motor, allowing to trick the head into going between tracks, in middle or even quarter positions. That was not usable to write more tracks, since the head width did not change, but since reliable reading was only possible with the correct position it was used for some copy protection systems.

The disk rotates at a fixed speed for a given track. The most usual speed is 300 rpm for every track, with 360 rpm found for HD 5.25" floppies and most 8" ones, and a number of different values like 90 rpm for the earlier floppies or 150 rpm for an HD floppy in an Amiga. Having a fixed rotational speed for the whole disk is called Constant Angular Velocity (CAV, almost everybody) or Zoned Constant Angular Velocity (ZCAV, C64) depending on whether the read/write bitrate is constant or track-dependant. Some systems (Apple II, Mac) vary the rotational speed depending on the track (something like 394 rpm up to 590 rpm) to end up with a Constant Linear Velocity (CLV). The idea behind ZCAV/CLV is to get more bits out of the media by keeping the minimal spacing between magnetic orientation transitions close to the best the support can do. It seems that the complexity was not deemed worth it since almost no system does it.

Finally, after the disc rotates and the head is over the proper track reading happens. The reading is done through an inductive head, which gives it the interesting characteristic of not reading the magnetic orientation directly but instead of being sensitive to orientation inversions, called flux transitions. This detection is weak and somewhat uncalibrated, so an amplifier with Automatic Gain Calibration (AGC) and a peak detector are put behind the head to deliver clean pulses. The AGC slowly increases the amplification level until a signal goes over the threshold, then modulates its gain so that said signal is at a fixed position over the threshold. Afterwards the increase happens again. This makes the amplifier calibrate itself to the signals read from the floppy as long as flux transitions happen often enough. Too long and the amplification level will reach a point where the random noise the head picks from the environment is amplified over the threshold, creating a pulse where none should be. Too long in our case happens to be around 16-20us with no transitions. That means a long enough zone with a fixed magnetic orientation or no orientation at all (demagnetized or damaged) is going to be read as a series of random pulses after a brief delay. This is used by protections and is known as "weak bits", which read differently each time they're accessed.

A second level of filtering happens after the peak detector. When two transitions are a little close (but still over the media threshold) a bouncing effect happens between them giving two very close pulses in the middle in addition to the two normal pulses. The floppy drive detects when pulses are too close and filter them out, leaving the normal ones. As a result, if one writes a train of high-frequency pulses to the floppy they will be read back as a train of too close pulses (weak because they're over the media tolerance, but picked up by the AGC anyway, only somewhat unreliably) they will be all filtered out, giving a large amount of time without any pulse in the output signal. This is used by some protections since it's not writable with a normally clocked controller.

Writing is symmetrical, with a series of pulses sent which make the write head invert the magnetic field orientation each time a pulse is received.

So, in conclusion, the floppy drive provides inputs to control disk rotation and head position (and choice when double-sided), and the data goes both way as a train of pulses representing magnetic orientation inversions. The absolute value

of the orientation itself is never known.

2.3. Floppy controller

The task of the floppy controller is to turn the signals to/from the floppy drive into something the main CPU can digest. The level of support actually done by the controller is extremely variable from one device to the other, from pretty much nothing (Apple II, C64) through minimal (Amiga) to complete (Western Digital chips, uPD765 family). Usual functions include drive selection, motor control, track seeking and of course reading and writing data. Of these only the last two need to be described, the rest is obvious.

The data is structured at two levels: how individual bits (or nibbles, or bytes) are encoded on the surface, and how these are grouped in individually-addressable sectors. Two standards exist for these, called FM and MFM, and in addition a number of systems use their home-grown variants. Moreover, some systems such as the Amiga use a standard bit-level encoding (MFM) but a homegrown sector-level organisation.

2.3.1. Bit-level encodings

2.3.1.1. Cell organization All floppy controllers, even the wonkiest like the Apple II one, start by dividing the track in equally-sized cells. They're angular sections in the middle of which a magnetic orientation inversion may be present. From a hardware point of view the cells are seen as durations, which combined with the floppy rotation give the section. For instance the standard MFM cell size for a 3" double-density floppy is 2us, which combined with the also standard 300 rpm rotational speed gives an angular size of 1/100000th of a turn. Another way of saying it is that there are 100K cells in a 3" DD track.

In every cell there may or may not be a magnetic orientation transition, e.g. a pulse coming from (reading) or going to (writing) the floppy drive. A cell with a pulse is traditionally noted '1', and one without '0'. Two constraints apply to the cell contents though. First, pulses must not be too close together or they'll blur each-other and/or be filtered out. The limit is slightly better than 1/50000th of a turn for single and double density floppies, half that for HD floppys, and half that again for ED floppies with perpendicular recording. Second, they must not be too away from each other or either the AGC is going to get wonky and introduce phantom pulses or the controller is going to lose sync and get a wrong timing on the cells on reading. Conservative rule of thumb is not to have more than three consecutive '0' cells.

Of course protections play with that to make formats not reproducible by the system controller, either breaking the three-zeroes rule or playing with the cells durations/sizes.

Bit endocing is then the art of transforming raw data into a cell 0/1 configuration that respects the two constraints.

2.3.1.2. FM encoding The very first encoding method developed for floppies is called Frequency Modulation, or FM. The cell size is set at slightly over the physical limit, e.g. 4us. That means it is possible to reliably have consecutive '1' cells. Each bit is encoded on two cells:

- the first cell, called the clock bit, is '1'
- the second cell, called data bit, is the bit

Since every other cell at least is '1' there is no risk of going over three zeroes.

The name Frequency Modulation simply derives from the fact that a 0 is encoded with one period of a 125Khz pulse train while a 1 is two periods of a 250Khz pulse train.

2.3.1.3. MFM encoding The FM encoding has been superseded by the Modified Frequency Modulation encoding, which can cram exactly twice as much data on the same surface, hence its other name of "double density". The cell size is set at slightly over half the physical limit, e.g. 2us usually. The constraint means that two '1' cells must be separated by at least one '0' cell. Each bit is once again encoded on two cells:

- the first cell, called the clock bit, is '1' if both the previous and current data bits are 0, '0' otherwise

- the second cell, called data bit, is the bit

The minimum space rule is respected since a '1' clock bit is by definition surrounded by two '0' data bits, and a '1' data bit is surrounded by two '0' clock bits. The longest '0'-cell string possible is when encoding 101 which gives x10001, respecting the maximum of three zeroes.

2.3.1.4. GCR encodings Group Coded Recording, or GCR, encodings are a class of encodings where strings of bits at least nibble-size are encoded into a given cell stream given by a table. It has been used in particular by the Apple II, the Mac and the C64, and each system has its own table, or tables.

2.3.1.5. Other encodings Other encodings exist, like M2FM, but they're very rare and system-specific.

2.3.1.6. Reading back encoded data Writing encoded data is easy, you only need a clock at the appropriate frequency and send or not a pulse on the clock edges. Reading back the data is where the fun is. Cells are a logical construct and not a physical measurable entity. Rotational speeds very around the defined one (+/- 2% is not rare) and local perturbations (air turbulence, surface distance...) make the instant speed very variable in general. So to extract the cell values stream the controller must dynamically synchronize with the pulse train that the floppy head picks up. The principle is simple: a cell-sized duration window is build within which the presence of at least one pulse indicates the cell is a '1', and the absence of any a '0'. After reaching the end of the window the starting time is moved appropriately to try to keep the observed pulse at the exact middle of the window. This allows to correct the phase on every '1' cell, making the synchronization work if the rotational speed is not too off. Subsequent generations of controllers used a Phase-Locked Loop (PLL) which vary both phase and window duration to adapt better to wrong rotational speeds, with usually a tolerance of +/- 15%.

Once the cell data stream is extracted decoding depends on the encoding. In the FM and MFM case the only question is to recognize data bits from clock bits, while in GCR the start position of the first group should be found. That second level of synchronization is handled at a higher level using patterns not found in a normal stream.

2.3.2. Sector-level organization

Floppies have been designed for read/write random access to reasonably sized blocks of data. Track selection allows for a first level of random access and sizing, but the ~6K of a double density track would be too big a block to handle. 256/512 bytes are considered a more appropriate value. To that end data on a track is organized as a series of (sector header, sector data) pairs where the sector header indicates important information like the sector number and size, and the sector data contains the data. Sectors have to be broken in two parts because while reading is easy, read the header then read the data if you want it, writing requires reading the header to find the correct place then once that is done switching on the writing head for the data. Starting writing is not instantaneous and will not be perfectly phase-aligned with the read header, so space for synchronization is required between header and data.

In addition somewhere in the sector header and in the sector data are pretty much always added some kind of checksum allowing to know whether the data was damaged or not.

FM and MFM have (not always used) standard sector layout methods.

2.3.2.1. FM sector layout The standard "PC" track/sector layout for FM is as such:

- A number of FM-encoded 0xff (usually 40)
- 6 FM-encoded 0x00 (giving a steady 125KHz pulse train)
- The 16-cell stream 1111011101111010 (f77a, clock 0xd7, data 0xfc)
- A number of FM-encoded 0xff (usually 26, very variable)

Then for each sector: - 6 FM-encoded 0x00 (giving a steady 125KHz pulse train)

- The 16-cell stream 1111010101111110 (f57e, clock 0xc7, data 0xfe)
- Sector header, e.g. FM encoded track, head, sector, size code and two bytes of crc
- 11 FM-encoded 0xff
- 6 FM-encoded 0x00 (giving a steady 125KHz pulse train)
- The 16-cell stream 1111010101101111 (f56f, clock 0xc7, data 0xfb)
- FM-encoded sector data followed by two bytes of crc
- A number of FM-encoded 0xff (usually 48, very variable)

The track is finished with a stream of '1' cells.

The 125KHz pulse trains are used to lock the PLL to the signal correctly. The specific 16-cells streams allow to distinguish between clock and data bits by providing a pattern that is not supposed to happen in normal FM-encoded data. In the sector header track numbers start at 0, heads are 0/1 depending on the size, sector numbers usually start at 1 and size code is 0 for 128 bytes, 1 for 256, 2 for 512, etc.

The CRC is a cyclic redundancy check of the data bits starting with the mark just after the pulse train using polynom 0x11021.

The Western Digital-based controllers usually get rid of everything but some 0xff before the first sector and allow a better use of space as a result.

2.3.2.2. MFM sector layout The standard "PC" track/sector layout for MFM is as such:

- A number of MFM-encoded 0x4e (usually 80)
- 12 FM-encoded 0x00 (giving a steady 250KHz pulse train)
- 3 times the 16-cell stream 0101001000100100 (5224, clock 0x14, data 0xc2)
- The MFM-encoded value 0xfc
- A number of MFM-encoded 0x4e (usually 50, very variable)

Then for each sector:

- 12 FM-encoded 0x00 (giving a steady 250KHz pulse train)
- 3 times the 16-cell stream 0100010010001001 (4489, clock 0x0a, data 0xa1)
- Sector header, e.g. MFM-encoded 0xfe, track, head, sector, size code and two bytes of crc
- 22 MFM-encoded 0x4e
- 12 MFM-encoded 0x00 (giving a steady 250KHz pulse train)
- 3 times the 16-cell stream 0100010010001001 (4489, clock 0x0a, data 0xa1)
- MFM-encoded 0xfb, sector data followed by two bytes of crc
- A number of MFM-encoded 0x4e (usually 84, very variable)

The track is finished with a stream of MFM-encoded 0x4e.

The 250KHz pulse trains are used to lock the PLL to the signal correctly. The cell pattern 4489 does not appear in normal MFM-encoded data and is used for clock/data separation.

As for FM, the Western Digital-based controllers usually get rid of everything but some 0x4e before the first sector and allow a better use of space as a result.

2.3.2.3. Formatting and write splices To be usable, a floppy must have the sector headers and default sector data written on every track before using it. The controller starts writing at a given place, often the index pulse but on some systems whenever the command is sent, and writes until a complete turn is done. That's called formatting the floppy. At the point where the writing stops there is a synchronization loss since there is no chance the cell stream clock warps around perfectly. This brutal phase change is called a write splice, specifically the track write splice. It is the point where writing should start if one wants to raw copy the track to a new floppy.

Similarly two write splices are created when a sector is written at the start and end of the data block part. They're not supposed to happen on a mastered disk though, even if there are some rare exceptions.

3. The new implementation

3.1. Floppy disk representation

The floppy disk contents are represented by the class floppy_image. It contains information of the media type and a representation of the magnetic state of the surface.

The media type is divided in two parts. The first half indicates the physical form factor, i.e. all medias with that form factor can be physically inserted in a reader that handles it. The second half indicates the variants which are usually detectable by the reader, such as density and number of sides.

Track data consists of a series of 32-bits lsb-first values representing magnetic cells. Bits 0-27 indicate the absolute position of the start of the cell (not the size), and bits 28-31 the type. Type can be:

- 0, MG_A -> Magnetic orientation A
- 1, MG_B -> Magnetic orientation B
- 2, MG_N -> Non-magnetized zone (neutral)
- 3, MG_D -> Damaged zone, reads as neutral but cannot be changed by writing

The position is in angular units of 1/200,000,000th of a turn. It corresponds to one nanosecond when the drive rotates at 300 rpm.

The last cell implicit end position is of course 200,000,000.

Unformatted tracks are encoded as zero-size.

The "track splice" information indicates where to start writing if you try to rewrite a physical disk with the data. Some preservation formats encode that information, it is guessed for others. The write track function of fdcs should set it. The representation is the angular position relative to the index.

3.2. Converting to and from the internal representation

3.2.1. Class and interface

We need to be able to convert on-disk formats of the floppy data to and from the internal representation. This is done through classes derived from floppy_image_format_t. The interface to be implemented includes: - **name()** gives the short name of the on-disk format

- **description()** gives a short description of the format
- **extensions()** gives a comma-separated list of file name extensions found for that format
- **supports_save()** returns true is converting to that external format is supported
- **identify(file, form factor)** gives a 0-100 score for the file to be of that format:
  - **0** = not that format

  - **100** = certainly that format
  - **50** = format identified from file size only
- **load(file, form factor, floppy_image)** loads an image and converts it into the internal representation
- **save(file, floppy_image)** (if implemented) converts from the internal representation and saves an image

All of these methods are supposed to be stateless.

3.2.2. Conversion helper methods

A number of methods are provided to simplify writing the converter classes.

3.2.2.1. Load-oriented conversion methods

generate_track_from_bitstream(track number,
head number,
UINT8 *cell stream,
int cell count,
floppy image)

Takes a stream of cell types (0/1), MSB-first, converts it to the internal format and stores it at the given track and head in the given image.

generate_track_from_levels(track number,
head number,
UINT32 *cell levels,
int cell count,
splice position,
floppy image)

Takes a variant of the internal format where each value represents a cell, the position part of the values is the size of the cell and the level part is MG_0, MG_1 for normal cell types, MG_N, MG_D for unformatted/damaged cells, and MG_W for Dungeon-Master style weak bits. Converts it into the internal format. The sizes are normalized so that they total to a full turn.

normalize_times(UINT32 *levels,
int level_count)

Takes an internal-format buffer where the position part represents angle until the next change and turns it into a normal positional stream, first ensuring that the total size is normalized to a full turn.

3.2.2.2. Save-oriented conversion methods

generate_bitstream_from_track(track number,
head number,
base cell size,
UINT8 *cell stream,

int &cell_stream_size,
floppy image)

Extract a cell 0/1 stream from the internal format using a PLL setup with an initial cell size set to 'base cell size' and a +/- 25% tolerance.

struct desc_xs { int track, head, size; const UINT8 *data }
extract_sectors_from_bitstream_mfm_pc(...)
extract_sectors_from_bitstream_fm_pc(const UINT8 *cell stream,
int cell_stream_size,
desc_xs *sectors,
UINT8 *sectdata,
int sectdata_size)

Extract standard mfm or fm sectors from a regenerated cell stream. Sectors must point to an array of 256 desc_xs.

An existing sector is recognizable by having ->data non-null. Sector data is written in sectdata up to sectdata_size bytes.

get_geometry_mfm_pc(...)
get_geometry_fm_pc(floppy image,
base cell size,
int &track_count,
int &head_count,
int &sector_count)

Extract the geometry (heads, tracks, sectors) from a pc-ish floppy image by checking track 20.

get_track_data_mfm_pc(...)
get_track_data_fm_pc(track number,
head number,
floppy image,
base cell size,
sector size,
sector count,
UINT8 *sector data)

Extract what you'd get by reading in order 'sector size'-sized sectors from number 1 to sector count and put the result in sector data.

3.3. Floppy drive

The class floppy_image_interface simulates the floppy drive. That includes a number of control signals, reading, and writing. Control signal changes must be synchronized, e.g. fired off a timer to ensure the current time is the same for all devices.

3.3.1. Control signals

Due to the way they're usually connected to CPUs (e.g. directly on an I/O port), the control signals work with physical instead of logical values. Which means than in general 0 means active, 1 means inactive. Some signals also have a callback associated called when they change.

mon_w(state) / mon_r()

Motor on signal, rotates on 0.

idx_r() / setup_index_pulse_cb(cb)

Index signal, goes 0 at start of track for about 2ms. Callback is synchronized. Only happens when a disk is in and the motor is running.

ready_r() / setup_ready_cb(cb)

Ready signal, goes to 1 when the disk is removed or the motor stopped. Goes to 0 after two index pulses.

wpt_r() / setup_wpt_cb(cb)

Write protect signal (1 = readonly). Callback is unsynchronized.

dskchg_r()

Disk change signal, goes to 1 when a disk is change, goes to 0 on track change.

dir_w(dir)

Selects track stepping direction (1 = out = decrease track number).

stp_w(state)

Step signal, moves by one track on 1->0 transition.

trk00_r()

Track 0 sensor, returns 0 when on track 0.

ss_w(ss) / ss_r()

Side select

3.3.2. Read/write interface

The read/write interface is designed to work asynchronously, e.g. somewhat independently of the current time.

The new SCSI subsystem

1. Introduction

The **nscsi** subsystem was created to allow an implementation to be closer to the physical reality, making it easier (hopefully) to implement new controller chips from the documentations.

2. Global structure

Parallel SCSI is built around a symmetric bus to which a number of devices are connected. The bus is composed of 9 control lines (for now, later SCSI versions may have more) and up to 32 data lines (but currently implemented chips only handle 8). All the lines are open collector, which means that either one or multiple chip connect the line to ground and the line, of course, goes to ground, or no chip drives anything and the line stays at Vcc. Also, the bus uses inverted

logic, where ground means 1. SCSI chips traditionally work in logical and not physical levels, so the nscsi subsystem also works in logical levels and does a logical-or of all the outputs of the devices.

Structurally, the implementation is done around two main classes: **nscsi_bus_devices** represents the bus, and **nscsi_device** represents an individual device. A device only communicate with the bus, and the bus takes care of transparently handling the device discovery and communication. In addition the **nscsi_full_device** class proposes a SCSI device with the SCSI protocol implemented making building generic SCSI devices like hard drives or CD-ROM readers easier.

3. Plugging in a SCSI bus in a driver

The nscsi subsystem leverages the slot interfaces and the device naming to allow for a configurable yet simple bus implementation.

First you need to create a list of acceptable devices to plug on the bus. This usually comprises of **cdrom**, **harddisk** and the controller chip. For instance:

```
static SLOT_INTERFACE_START( next_scsi_devices )
    SLOT_INTERFACE("cdrom", NSCSI_CDROM)
    SLOT_INTERFACE("harddisk", NSCSI_HARDDISK)
    SLOT_INTERFACE_INTERNAL("ncr5390", NCR5390)
SLOT_INTERFACE_END
```

The **_INTERNAL** interface indicates a device that is not user-selectable, which is useful for the controller.

Then in the machine config (or in a fragment config) you need to first add the bus, and then the (potential) devices as sub-devices of the bus with the SCSI ID as the name. For instance you can use:

```
    MCFG_NSCSI_BUS_ADD("scsibus")
    MCFG_NSCSI_ADD("scsibus:0", next_scsi_devices, "cdrom", 0, 0, 0, false)
    MCFG_NSCSI_ADD("scsibus:1", next_scsi_devices, "harddisk", 0, 0, 0, false)
    MCFG_NSCSI_ADD("scsibus:2", next_scsi_devices, 0, 0, 0, 0, false)
    MCFG_NSCSI_ADD("scsibus:3", next_scsi_devices, 0, 0, 0, 0, false)
    MCFG_NSCSI_ADD("scsibus:4", next_scsi_devices, 0, 0, 0, 0, false)
    MCFG_NSCSI_ADD("scsibus:5", next_scsi_devices, 0, 0, 0, 0, false)
    MCFG_NSCSI_ADD("scsibus:6", next_scsi_devices, 0, 0, 0, 0, false)
    MCFG_NSCSI_ADD("scsibus:7", next_scsi_devices, "ncr5390", 0, &next_ncr5390_interface, 10000000,
    true)
```

That configuration puts as default a CD-ROM reader on SCSI ID 0 and a hard drive on SCSI ID 1, and forces the controller on ID 7. The parameters of add are:

- device tag, comprised of bus-tag:scsi-id
- the list of acceptable devices
- the device name as per the list, if one is to be there by default

- the device input config, if any (and there usually isn't one)
- the device configuration structure, usually for the controller only
- the frequency, usually for the controller only
- "**false**" for a user-modifiable slot, "**true**" for a fixed slot

The full device name, for mapping purposes, will be **bus-tag:scsi-id:device-type**, i.e. "*scsibus:7:ncr5390*" for our controller here.

4. Creating a new SCSI device using nscsi_device

The base class "**nscsi_device**" is to be used for down-to-the-metal devices, i.e. SCSI controller chips. The class provides three variables and one method. The first variable, **scsi_bus**, is a pointer to the **nscsi_bus_device**. The second, **scsi_refid**, is an opaque reference to pass to the bus on some operations. Finally, **scsi_id** gives the SCSI ID as per the device tag. It's written once at startup and never written or read afterwards, the device can do whatever it wants with the value or the variable.

The virtual method **scsi_ctrl_changed** is called when watched-for control lines change. Which lines are watched is defined through the bus.

The bus proposes five methods to access the lines. The read methods are **ctrl_r()** and **data_r()**. The meaning of the control bits are defined in the **S_*** enum of **nscsi_device**. The bottom three bits (**INP**, **CTL** and **MSG**) are setup so that masking with 7 (**S_PHASE_MASK**) gives the traditional numbers for the phases, which are also available with the **S_PHASE_*** enum.

Writing the data lines is done with **data_w(scsi_refid, value)**.

Writing the control lines is done with **ctrl_w(scsi_refid, value, mask-of-lines-to-change)**. To change all control lines in one call use **S_ALL** as the mask.

Of course, what is read is the logical-or of all of what is driven by all devices.

Finally, the method **ctrl_wait_w(scsi_id, value, mask-of-wait-lines-to-change)** allows to select which control lines are watched. The watch mask is per-device, and the device method **scsi_ctrl_changed** is called whenever a control line in the mask changes due to an action of another device (not itself, to avoid an annoying and somewhat useless recursion).

Implementing the controller is then just a matter of following the state machines descriptions, at least if they're available. The only part often not described is the arbitration/selection, which is documented in the SCSI standard though. For an initiator (which is what the controller essentially always is), it goes like this:

- wait for the bus to be idle
- assert the data line which number is your scsi_id (1 << scsi_id)
- assert the busy line
- wait the arbitration time
- check that the of the active data lines the one with the highest number is yours
  - if no, the arbitration was lost, stop driving anything and restart at the beginning
- assert the select line (at that point, the bus is yours)
- wait a short while
- keep your data line asserted, assert the data line which number is the SCSI ID of the target
- wait a short while
- assert the atn line if needed, de-assert busy
- wait for busy to be asserted or timeout

  - timeout means nobody is answering at that id, de-assert everything and stop
- wait a short while for de-skewing
- de-assert the data bus and the select line
- wait a short while

and then you're done, you're connected with the target until the target de-asserts the busy line, either because you asked it to or just to annoy you. The de-assert is called a disconnect.

The **ncr5390** is an example of how to use a two-level state machine to handle all the events.

5. Creating a new SCSI device using **nscsi_full_device**

The base class "**nscsi_full_device**" is used to create HLE-d SCSI devices intended for generic uses, like hard drives, CD-ROMs, perhaps scanners, etc. The class provides the SCSI protocol handling, leaving only the command handling and (optionally) the message handling to the implementation.

The class currently only support target devices.

The first method to implement is **scsi_command()**. That method is called when a command has fully arrived. The command is available in **scsi_cmdbuf[]**, and its length is in **scsi_cmdsize** (but the length is generally useless, the command first byte giving it). The 4096-bytes **scsi_cmdbuf** array is then freely modifiable.

In **scsi_command()**, the device can either handle the command or pass it up with **nscsi_full_device::scsi_command()**.

To handle the command, a number of methods are available:

- **get_lun(lua-set-in-command)** will give you the LUN to work on (the in-command one can be overriden by a message-level one).
- **bad_lun()** replies to the host that the specific LUN is unsupported.
- **scsi_data_in(buffer-id, size)** sends size bytes from buffer *buffer-id*
- **scsi_data_out(buffer-id, size)** receives size bytes into buffer *buffer-id*
- **scsi_status_complete(status)** ends the command with a given status.
- **sense(deferred, key)** prepares the sense buffer for a subsequent request-sense command, which is useful when returning a check-condition status.

The **scsi_data_*** and **scsi_status_complete** commands are queued, the command handler should call them all without waiting.

buffer-id identifies a buffer. 0, aka **SBUF_MAIN**, targets the **scsi_cmdbuf** buffer. Other acceptable values are 2 or more. 2+ ids are handled through the **scsi_get_data** method for read and **scsi_put_data** for write.

UINT8 device::scsi_get_data(int id, int pos) must return byte pos of buffer id, upcalling in nscsi_full_device for id < 2.

void device::scsi_put_data(int id, int pos, UINT8 data) must write byte pos in buffer id, upcalling in nscsi_full_device for id < 2.

scsi_get_data and **scsi_put_data** should do the external reads/writes when needed.

The device can also override **scsi_message** to handle SCSI messages other than the ones generically handled, and it can also override some of the timings (but a lot of them aren't used, beware).

A number of enums are defined to make things easier. The **SS_*** enum gives status returns (with **SS_GOOD** for all's well). The **SC_*** enum gives the scsi commands. The **SM_*** enum gives the SCSI messages, with the exception of identify (which is **80-ff**, doesn't really fit in an enum).

6. What's missing 6.1. What's missing in **scsi_full_device**

Initiator support - we have no initiator device to HLE at that point.

Delays - a scsi_delay command would help giving more realistic timings to the CD-ROM reader in particular.

Disconnected operation - would first require delays and in addition an emulated OS that can handle it.

16-bits wide operation - needs an OS and an initiator that can handle it.

6.2. What's missing in the ncr5390 (and probably future other controllers)

Bus free detection. Right now the bus is considered free if the controllers isn't using it, which is true. This may change once disconnected operation is in.

Target commands, we don't emulate (vs. HLE) any target yet.

Scripting MAME via LUA

Introduction

It is now possible to externally drive MAME via LUA scripts. This feature initially appeared in version 0.148, when a minimal `luaengine` was implemented. Nowadays, the LUA interface is rich enough to let you inspect and manipulate devices state, access CPU registers, read and write memory, and draw a custom HUD on screen.

Internally, MAME makes extensive use of `luabridge` to implement this feature: the idea is to transparently expose as many of the useful internals as possible.

Finally, a warning: The LUA API is not yet declared stable and may suddenly change without prior notice. However, we expose methods to let you know at runtime which API version you are running against, and you can introspect most of the objects at runtime.

Features

The API is not yet complete, but this is a partial list of capabilities currently available to LUA scripts:

- machine metadata (app version, current rom, rom details)
- machine control (starting, pausing, resetting, stopping)
- machine hooks (on frame painting and on user events)
- devices introspection (device tree listing, memory and register enumeration)
- screens introspection (screens listing, screen details, frames counting)
- screen HUD drawing (text, lines, boxes on multiple screens)
- memory read/write (8/16/32/64 bits, signed and unsigned)
- registers and states control (states enumeration, get and set)

Usage

MAME supports external scripting via LUA (>= 5.3) scripts, either written on the interactive console or loaded as a file. To reach the console, just run MAME with **-console** and you will be greeted by a naked > prompt where you can input your script.

To load a whole script at once, store it in a plain text file and pass it via **-autoboot_script**. Please note that script loading may be delayed (few seconds by default), but you can override the default with the **-autoboot_delay** argument.

To control the execution of your code, you can use a loop-based or an event-based approach. The former is not encouraged as it is resource-intensive and makes control flow unnecessarily complex. Instead, we suggest to register custom hooks to be invoked on specific events (eg. at each frame rendering).

Walkthrough

Let's first run MAME in a terminal to reach the LUA console:

```
$ mame -console YOUR_ROM
M.A.M.E. v0.158 (Feb  5 2015) - Multiple Arcade Machine Emulator
Copyright Nicola Salmoria and the MAME team
Lua 5.3.0  Copyright (C) 1994-2015 Lua.org, PUC-Rio

>
```

At this point, your game is probably running in demo mode, let's pause it:

```
> emu.pause()
>
```

Even without textual feedback on the console, you'll notice the game is now paused. In general, commands are quiet and only print back error messages.

You can check at runtime which version of MAME you are running, with:

```
> print(emu.app_name() .. " " .. emu.app_version())
mame 0.158
```

We now start exploring screen related methods. First, let's enumerate available screens:

```
> for i,v in pairs(manager:machine().screens) do print(i) end
:screen
```

manager:machine() is the root object of your currently running machine: we will be using this often. **screens** is a table with all available screens; most machines only have one main screen. In our case, the main and only screen is tagged as **:screen**, and we can further inspect it:

```
> -- let's define a shorthand for the main screen
> s = manager:machine().screens[":screen"]
> print(s:width() .. "x" .. s:height())
320x224
```

We have several methods to draw on the screen a HUD composed of lines, boxes and text:

```
> -- we define a HUD-drawing function, and then call it
> function draw_hud()
>> s:draw_text(40, 40, "foo"); -- (x0, y0, msg)
>> s:draw_box(20, 20, 80, 80, 0, 0xff00ffff); -- (x0, y0, x1, y1, fill-color, line-color)
>> s:draw_line(20, 20, 80, 80, 0xff00ffff); -- (x0, y0, x1, y1, line-color)
>> end
> draw_hud();
```

This will draw some useless art on the screen. However, when unpausing the game, your HUD needs to be refreshed otherwise it will just disappear. In order to do this, you have to register your hook to be called on every frame repaint:

```
> emu.register_frame_done(draw_hud, "frame")
```

All colors are expected in ARGB format (32b unsigned), while screen origin (0,0) normally corresponds to the top-left corner.

Similarly to screens, you can inspect all the devices attached to a machine:

```
> for k,v in pairs(manager:machine().devices) do print(k) end
:audiocpu
:maincpu
:saveram
:screen
:palette
[...]
```

On some of them, you can also inspect and manipulate memory and state:

```
> cpu = manager:machine().devices[":maincpu"]
> -- enumerate, read and write state registers
> for k,v in pairs(cpu.state) do print(k) end
D5
SP
A4
A3
D0
PC
[...]
> print(cpu.state["D0"].value)
303
> cpu.state["D0"].value = 255
> print(cpu.state["D0"].value)
255
```

```
> -- inspect memory
> for k,v in pairs(cpu.spaces) do print(k) end
program
> mem = cpu.spaces["program"]
> print(mem:read_i8(0xC000))
41
```

The new 6502 family implementation

Introduction

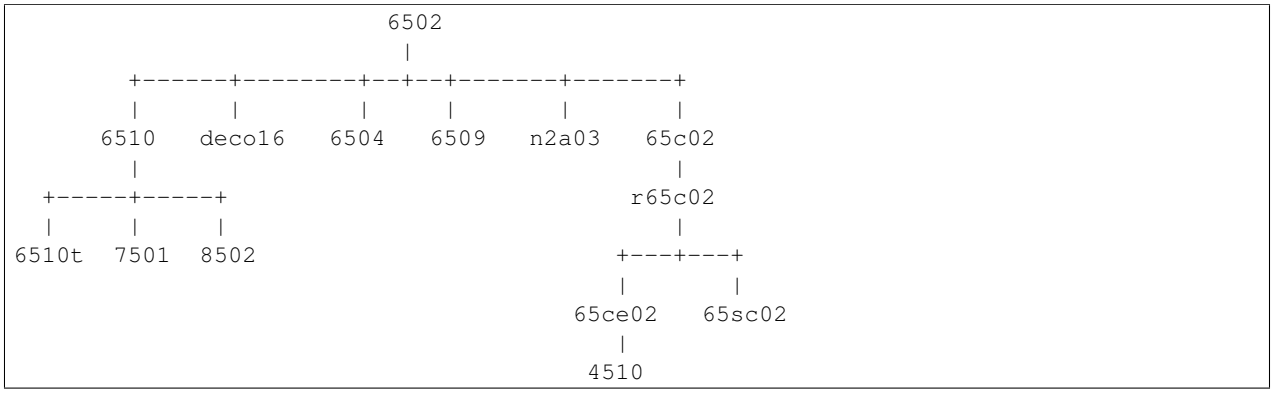
The new 6502 family implementation has been created to reach sub-instruction accuracy in observable behaviour. It is designed with 3 goals in mind:

- every bus cycle must happen at the exact time it would happen in a real CPU, and every access the real CPU does is done
- instructions can be interrupted at any time in the middle then restarted at that point transparently
- instructions can be interrupted even from within a memory handler for bus contention/wait states emulation purposes

Point 1 has been ensured through bisimulation with the gate-level simulation perfect6502. Point 2 has been ensured structurally through a code generator which will be explained in section 8. Point 3 is not done yet due to lack of support on the memory subsystem side, but section 9 shows how it will be handled.

The 6502 family

The MOS 6502 family has been large and productive. A large number of variants exist, varying on bus sizes, I/O, and even opcodes. Some offshots (g65c816, hu6280) even exist that live elsewhere in the mame tree. The final class hierarchy is this:



The 6510 adds an up to 8 bits I/O port, with the 6510t, 7501 and 8502 being software-identical variants with different pin count (hence I/O count), die process (NMOS, HNMOS, etc) and clock support.

The deco16 is a Deco variant with a small number of not really understood additional instructions and some I/O.

The 6504 is a pin and address-bus reduced version.

The 6509 adds internal support for paging.

The n2a03 is the NES variant with the D flag disabled and sound functionality integrated.

The 65c02 is the very first cmos variant with some additional instructions, some fixes, and most of the undocumented instructions turned into nops. The R (Rockwell, but eventually produced by WDC too among others) variant adds a number of bitwise instructions and also stp and wai. The SC variant, used by the Lynx portable console, looks identical to the R variant. The 'S' probably indicates a static-ram-cell process allowing full DC-to-max clock control.

The 65ce02 is the final evolution of the ISA in this hierarchy, with additional instructions, registers, and removals of a lot of dummy accesses that slowed the original 6502 down by at least 25%. The 4510 is a 65ce02 with integrated MMU and GPIO support.

Usage of the classes

All the CPUs are standard modern CPU devices, with all the normal interaction with the device infrastructure. To include one of these CPUs in your driver you need to include "**CPU/m6502/<CPU>.h**" and then do a **MCFG_CPU_ADD("tag", <CPU>, clock)**.

6510 variants port I/O callbacks are setup through: MCFG_<CPU>_PORT_CALLBACKS(READ8(type, read_method), WRITE8(type, write_method))

And the pullup and floating lines mask is given through: MCFG_<CPU>_PORT_PULLS(pullups, floating)

In order to see all bus accesses on the memory handlers it is possible to disable accesses through the direct map (at a CPU cost, MCFG_M6502_DISABLE_DIRECT()

In that case, transparent decryption support is also disabled, everything goes through normal memory-map read/write calls. The state of the sync line is given by the CPU method **get_sync()**, making implementing the decryption in the handler possible.

Also, as for every executable device, the CPU method **total_cycles()** gives the current time in cycles since the start of the machine from the point of view of the CPU. Or, in other words, what is usually called the cycle number for the

CPU when somebody talks about bus contention or wait states. The call is designed to be fast (no system-wide sync, no call to **machine.time**()) and is precise. Cycle number for every access is exact at the sub-instruction level.

The 4510 special nomap line is accessible through **get_nomap**().

Other than these specifics, these are perfectly normal CPU classes.

General structure of the emulations

Each variant is emulated through up to 4 files:

- <CPU>.h = header for the CPU class
- <CPU>.c = implementation of most of the CPU class
- d<CPU>.lst = dispatch table for the CPU
- o<CPU>.lst = opcode implementation code for the CPU

The last two are optional. They're used to generate a **<CPU>.inc** file in the object directory which is included by the .c file.

At a minimum, the class must include a constructor and an enum picking up the correct input line ids. See m65sc02 for a minimalist example. The header can also include specific configuration macros (see m8502) and also the class can include specific memory accessors (more on these later, simple example in m6504).

If the CPU has its own dispatch table, the class must also include the declaration (but not definition) of **disasm_entries**, **do_exec_full** and **do_exec_partial**, the declaration and definition of **disasm_disassemble** (identical for all classes but refers to the class-specific **disasm_entries** array) and include the .inc file (which provides the missing definitions). Support for the generation must also be added to CPU.mak.

If the CPU has in addition its own opcodes, their declaration must be done through a macro, see f.i. m65c02. The .inc file will provide the definitions.

Dispatch tables

Each d<CPU>.lst is the dispatch table for the CPU. Lines starting with '#' are comments. The file must include 257 entries, the first 256 being opcodes and the 257th what the CPU should do on reset. In the 6502 irq and nmi actually magically call the "brk" opcode, hence the lack of specific description for them.

Entries 0 to 255, i.e. the opcodes, must have one of these two structures:

- opcode_addressing-mode
- opcode_middle_addressing-mode

Opcode is traditionally a three-character value. Addressing mode must be a 3-letter value corresponding to one of the DASM_* macros in m6502.h. Opcode and addressing mode are used to generate the disassembly table. The full entry text is used in the opcode description file and the dispatching methods, allowing for per-CPU variants for identical-looking opcodes.

An entry of "." was usable for unimplemented/unknown opcodes, generating "???" in the disassembly, but is not a good idea at this point since it will infloop in execute() if encountered.

Opcode descriptions

Each **o<CPU>.lst** file includes the CPU-specific opcodes descriptions. An opcode description is a series of lines starting by an opcode entry by itself and followed by a series of indented lines with code executing the opcode.

For instance the asl <absolute address> opcode looks like this:

```
asl_aba
        TMP = read_pc();
        TMP = set_h(TMP, read_pc());
        TMP2 = read(TMP);
        write(TMP, TMP2);
        TMP2 = do_asl(TMP2);
        write(TMP, TMP2);
        prefetch();
```

First the low part of the address is read, then the high part (**read_pc** is auto-incrementing). Then, now that the address is available the value to shift is read, then re-written (yes, the 6502 does that), shifted then the final result is written (do_asl takes care of the flags). The instruction finishes with a prefetch of the next instruction, as all non-CPU-crashing instructions do.

Available bus-accessing functions are:

read(adr)	standard read
read_direct(adr)	read from program space
read_pc()	read at the PC address and increment it
read_pc_noinc()	read at the PC address
read_9()	6509 indexed-y banked read
write(adr, val)	standard write
prefetch()	instruction prefetch
prefetch_noirq()	instruction prefetch without irq check

Cycle counting is done by the code generator which detects (through string matching) the accesses and generates the appropriate code. In addition to the bus-accessing functions a special line can be used to wait for the next event (irq or whatever). "**eat-all-cycles;**" on a line will do that wait then continue. It is used by wai_imp and stp_imp for the m65c02.

Due to the constraints of the code generation, some rules have to be followed:

- in general, stay with one instruction/expression per line
- there must be no side effects in the parameters of a bus-accessing function
- local variables lifetime must not go past a bus access. In general, it's better to leave them to helper methods (like **do_asl**) which do not do bus accesses. Note that "TMP" and "TMP2" are not local variables, they're variables of the class.
- single-line then or else constructs must have braces around them if they're calling a bus-accessing function

The per-opcode generated code are methods of the CPU class. As such they have complete access to other methods of the class, variables of the class, everything.

Memory interface

For better opcode reuse with the MMU/banking variants, a memory access subclass has been created. It's called **memory_interface**, declared in m6502_device, and provides the following accessors:

UINT8 read(UINT16 adr)	normal read
UINT8 read_sync(UINT16 adr)	opcode read with sync active (first byte of opcode)
UINT8 read_arg(UINT16 adr)	opcode read with sync inactive (rest of opcode)
void write(UINT16 adr, UINT8 val)	normal write

UINT8 read_9(UINT16 adr)	special y-indexed 6509 read, defaults to read()
void write_9(UINT16 adr, UINT8 val);	special y-indexed 6509 write, defaults to write()

Two implementations are given by default, one usual, **mi_default_normal**, one disabling direct access, **mi_default_nd**. A CPU that wants its own interface (see 6504 or 6509 for instance) must override device_start, intialize mintf there then call **init()**.

The generated code

A code generator is used to support interrupting and restarting an instruction in the middle. This is done through a two-level state machine with updates only at the boundaries. More precisely, inst_state tells you which main state you're in. It's equal to the opcode byte when 0-255, and 0xff00 means reset. It's always valid and used by instructions like rmb. inst_substate indicates at which step we are in an instruction, but it set only when an instruction has been interrupted. Let's go back to the asl <abs> code:

```
asl_aba
        TMP = read_pc();
        TMP = set_h(TMP, read_pc());
        TMP2 = read(TMP);
        write(TMP, TMP2);
        TMP2 = do_asl(TMP2);
        write(TMP, TMP2);
        prefetch();
```

The complete generated code is:

```
void m6502_device::asl_aba_partial()
{
switch(inst_substate) {
case 0:
        if(icount == 0) { inst_substate = 1; return; }
case 1:
        TMP = read_pc();
        icount–;
        if(icount == 0) { inst_substate = 2; return; }
case 2:
        TMP = set_h(TMP, read_pc());
        icount–;
        if(icount == 0) { inst_substate = 3; return; }
case 3:
```

```
        TMP2 = read(TMP);
        icount–;
        if(icount == 0) { inst_substate = 4; return; }
case 4:
        write(TMP, TMP2);
        icount–;
        TMP2 = do_asl(TMP2);
        if(icount == 0) { inst_substate = 5; return; }
case 5:
        write(TMP, TMP2);
        icount–;
        if(icount == 0) { inst_substate = 6; return; }
case 6:
        prefetch();
        icount–;
}
        inst_substate = 0;
}
```

One can see that the initial switch() restarts the instruction at the appropriate substate, that icount is updated after each access, and upon reaching 0 the instruction is interrupted and the substate updated. Since most instructions are started from the beginning a specific variant is generated for when inst_substate is known to be 0:

```
void m6502_device::asl_aba_full()
{
        if(icount == 0) { inst_substate = 1; return; }
        TMP = read_pc();
        icount–;
        if(icount == 0) { inst_substate = 2; return; }
        TMP = set_h(TMP, read_pc());
        icount–;
        if(icount == 0) { inst_substate = 3; return; }
        TMP2 = read(TMP);
        icount–;
        if(icount == 0) { inst_substate = 4; return; }
        write(TMP, TMP2);
        icount–;
        TMP2 = do_asl(TMP2);
        if(icount == 0) { inst_substate = 5; return; }
        write(TMP, TMP2);
        icount–;
        if(icount == 0) { inst_substate = 6; return; }
        prefetch();
        icount–;
```

```
}
```

That variant removes the switch, avoiding a costly computed branch and also an inst_substate write. There is in addition a fair chance that the decrement-test with zero pair is compiled into something efficient.

All these opcode functions are called through two virtual methods, **do_exec_full** and **do_exec_partial**, which are generated into a 257-entry switch statement. Pointers-to-methods being expensive to call, a virtual function implementing a switch has a fair chance of being better.

The execute main call ends up very simple:

```
void m6502_device::execute_run()
{
        if(inst_substate)
                do_exec_partial();

        while(icount > 0) {
                if(inst_state < 0x100) {
                        PPC = NPC;
                        inst_state = IR;
                        if(machine().debug_flags & DEBUG_FLAG_ENABLED)
                                debugger_instruction_hook(this, NPC);
                }
                do_exec_full();
        }
}
```

If an instruction was partially executed finish it (icount will then be zero if it still doesn't finish). Then try to run complete instructions. The NPC/IR dance is due to the fact that the 6502 does instruction prefetching, so the instruction PC and opcode come from the prefetch results.

Future bus contention/delay slot support

Supporting bus contention and delay slots in the context of the code generator only requires being able to abort a bus access when not enough cycles are available into icount, and restart it when cycles have become available again. The implementation plan is to:

- Have a delay() method on the CPU that removes cycles from icount. If icount becomes zero or less, having it throw a **suspend()** exception.
- Change the code generator to generate this:

```
void m6502_device::asl_aba_partial()
{
switch(inst_substate) {
case 0:
        if(icount == 0) { inst_substate = 1; return; }
```

```
case 1:
        try {
        TMP = read_pc();
        } catch(suspend) { inst_substate = 1; return; }
        icount–;
        if(icount == 0) { inst_substate = 2; return; }
case 2:
        try {
        TMP = set_h(TMP, read_pc());
        } catch(suspend) { inst_substate = 2; return; }
        icount–;
        if(icount == 0) { inst_substate = 3; return; }
case 3:
        try {
        TMP2 = read(TMP);
        } catch(suspend) { inst_substate = 3; return; }
        icount–;
        if(icount == 0) { inst_substate = 4; return; }
case 4:
        try {
        write(TMP, TMP2);
        } catch(suspend) { inst_substate = 4; return; }
        icount–;
        TMP2 = do_asl(TMP2);
        if(icount == 0) { inst_substate = 5; return; }
case 5:
        try {
        write(TMP, TMP2);
        } catch(suspend) { inst_substate = 5; return; }
        icount–;
        if(icount == 0) { inst_substate = 6; return; }
case 6:
        try {
        prefetch();
        } catch(suspend) { inst_substate = 6; return; }
        icount–;
}
        inst_substate = 0;
}
```

A modern try/catch costs nothing if an exception is not thrown. Using this the control will go back to the main loop, which will then look like this:

```
void m6502_device::execute_run()
{
        if(waiting_cycles) {
```

```
        icount -= waiting_cycles;
        waiting_cycles = 0;
    }

    if(icount > 0 && inst_substate)
        do_exec_partial();

    while(icount > 0) {
        if(inst_state < 0x100) {
            PPC = NPC;
            inst_state = IR;
            if(machine().debug_flags & DEBUG_FLAG_ENABLED)
                debugger_instruction_hook(this, NPC);
        }
        do_exec_full();
    }

    waiting_cycles = -icount;
    icount = 0;
}
```

A negative icount means that the CPU won't be able to do anything for some time in the future, because it's either waiting for the bus to be free or for a peripheral to answer. These cycles will be counted until elapsed and then normal processing will go on. It's important to note that the exception path only happens when the contention/wait state goes further than the scheduling slice of the CPU. That should not usually be the case, so the cost should be minimal.

Multi-dispatch variants

Some variants currently in the process of being supported change instruction set depending on an internal flag, either switching to a 16-bits mode or changing some register accesses to memory accesses. This is handled by having multiple dispatch tables for the CPU, the d<CPU>.lst not being 257 entries anymore but 256*n+1. The variable **inst_state_base** must select which instruction table to use at a given time. It must be a multiple of 256, and is in fact simply OR-ed to the first instruction byte to get the dispatch table index (aka inst_state).

Current TO-DO:

- Implement the bus contention/wait states stuff, but that requires support on the memory map side first.
- Integrate the I/O subsystems in the 4510
- Possibly integrate the sound subsytem in the n2a03
- Add decent hookups for the Apple 3 madness

CHAPTER

NINE

MAME AND SECURITY CONCERNS

MAME is not intended or designed to run in secure sites. It has not been security audited for such types of usage, and has been known in the past to have flaws that could be used for malicious intent if run as administator or root.

We do not suggest or condone the use of MAME as administrator or root and use as such is done at your own risk.

Bug reports, however, are always welcome.

CHAPTER

TEN

THE MAME LICENSE

The MAME project as a whole is distributed under the terms of the GNU General Public License, version 2 or later (GPL-2.0+), since it contains code made available under multiple GPL-compatible licenses. A great majority of files (over 90% including core files) are under the BSD-3-Clause License and we would encourage new contributors to distribute files under this license.

Please note that MAME is a registered trademark of Gregory Ember, and permission is required to use the "MAME" name, logo, or wordmark.

> Copyright (C) 1997-2018 MAMEDev and contributors
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Please see LICENSE.md for further details.

CHAPTER

ELEVEN

CONTRIBUTE

The documentation on this site is the handiwork of our many contributors.