# MuMapRom

Thomas Richter

| COLLABORATORS | | | |
|---|---|---|---|
| | *TITLE* :<br><br>MuMapRom | | |
| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
| WRITTEN BY | Thomas Richter | July 16, 2022 | |

| REVISION HISTORY | | | |
|---|---|---|---|
| NUMBER | DATE | DESCRIPTION | NAME |
| | | | |

# Contents

# Chapter 1

# MuMapRom

## 1.1 MuMapRom Guide

´### ,#. ,#### ,### ####’ #### ,#### ,###’ ####’ #### _____ ,#### ,###’ || ####’ #### | ___ ___ | ___ ,#### ,###’ ---- | | | | | | |___ ####’ #### | | | | | | | ,#####. ,###’ . | |___| |___| |_ ___| ####’##. ,#### ,## ,#### ########’ # ,##’ ####’ `####’ `###’ ,#### #####  ####’ Thomas Richter `##’

---

MuMapRom Guide

Guide Version 1.00 MuFastRom Version 40.1



Warning:

MuMapRom requires a lot of "magic" to do its job, and it does not necessarely run on each board due to its very hacky nature. Before you play with MuMapRom, make sure you read the installation chapter of this guide. If there is any other way to install an alternative ROM, for example by a CPU-board specific program, you will most likely want to use this alternative program. The purpose of MuMapRom is mainly to install a different kickstart temporarely for testing purposes - it should not be run permanently. If you need a different/newer Kickstart release, you should buy the ROM instead.

© THOR-Software

Thomas Richter

Rühmkorffstraße 10A

12209 Berlin

Germany

EMail: thor@math.tu-berlin.de

## 1.2 The THOR-Software Licence

The THOR-Software Licence (v2, 24th June 1998)

This License applies to the computer programs known as "MuMapRom" and the "MuMapRom.guide". The "Program", below, refers to such program. The "Archive" refers to the package of distribution, as prepared by the author of the Program, Thomas Richter. Each licensee is addressed as "you".

The Program and the data in the archive are freely distributable under the restrictions stated below, but are also Copyright (c) Thomas Richter.

Distribution of the Program, the Archive and the data in the Archive by a commercial organization without written permission from the author to any third party is prohibited if any payment is made in connection with such distribution, whether directly (as in payment for a copy of the Program) or indirectly (as in payment for some service related to the Program, or payment for some product or service that includes a copy of the Program "without charge"; these are only examples, and not an exhaustive enumeration of prohibited activities).

However, the following methods of distribution involving payment shall not in and of themselves be a violation of this restriction:

(i) Posting the Program on a public access information storage and retrieval service for which a fee is received for retrieving information (such as an on-line service), provided that the fee is not content-dependent (i.e., the fee would be the same for retrieving the same volume of information consisting of random data).

(ii) Distributing the Program on a CD-ROM, provided that

a) the Archive is reproduced entirely and verbatim on such CD-ROM, including especially this licence agreement;

b) the CD-ROM is made available to the public for a nominal fee only,

c) a copy of the CD is made available to the author for free except for shipment costs, and

d) provided further that all information on such CD-ROM is re-distributable for non-commercial purposes without charge.

Redistribution of a modified version of the Archive, the Program or the contents of the Archive is prohibited in any way, by any organization, regardless whether commercial or non-commercial. Everything must be kept together, in original and unmodified form.

Limitations.

THE PROGRAM IS PROVIDED TO YOU "AS IS", WITHOUT WARRANTY. THERE IS NO WARRANTY FOR THE PROGRAM, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY RIGHTS. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IF YOU DO NOT ACCEPT THIS LICENCE, YOU MUST DELETE THE PROGRAM, THE ARCHIVE AND ALL DATA OF THIS ARCHIVE FROM YOUR STORAGE SYSTEM. YOU ACCEPT THIS LICENCE BY USING OR REDISTRIBUTING THE PROGRAM.

Thomas Richter

## 1.3 What's the MMU.library?

All "modern" Amiga computers come with a special hardware component called the "MMU" for short, "Memory Management Unit". The MMU is a very powerful piece of hardware that can be seen as a translator between the CPU that carries out the actual calculation, and the surrounding hardware: Memory and IO devices. Each external access of the CPU is filtered by the MMU, checked whether the memory region is available, write protected, can be hold in the CPU internal cache and more. The MMU can be told to translate the addresses as seen from the CPU to different addresses, hence it can be used to "re-map" parts of the memory without actually touching the memory itself.

A series of programs is available that make use of the MMU: First of all, it's needed by the operating system to tell the CPU not to hold "chip memory", used by the Amiga custom chips, in its cache; second, several tools re-map the Kickstart ROM to

faster 32Bit RAM by using the MMU to translate the ROM addresses - as seen from the CPU - to the RAM addresses where the image of the ROM is kept. Third, a number of debugging tools make use of it to detect accesses to physically unavailable memory regions, and hence to find bugs in programs; amongst them is the "Enforcer" by Michael Sinz. Fourth, the MMU can be used to create the illusion of "almost infinite memory", with so-called "virtual memory systems". Last but not least, a number of miscellaneous applications have been found for the MMU as well, for example for display drivers of emulators.

Unfortunately, the Amiga Os does not provide ANY interface to the MMU, everything boils down to hardware hacking and every program hacks the MMU table as it wishes. Needless to say this prevents program A from working nicely together with program B, Enforcer with FastROM or VMM, and other combinations have been impossible up to now.

THIS HAS TO CHANGE! There has to be a documented interface to the MMU that makes accesses transparent, easy and compatible. This is the goal of the "mmu.library". In one word, COMPATIBILITY.

Unfortunately, old programs won't use this library automatically, so things have to be rewritten. The "MuTools" are a collection of programs that take over the job of older applications that hit the hardware directly. The result are programs that operate hardware independent, without any CPU or MMU specific parts, no matter what kind of MMU is available, and programs that nicely co-exist with each other.

I hope other program authors choose to make use of the library in the future and provide powerful tools without the compatibility headache. The MuTools are just a tiny start, more has to follow.

## 1.4 What's the job of MuMapRom?

MuMapRom is a mmu.library compatible ROM kicker. Hence, it replaces the ROM resident Kickstart - the operating system of the Amiga - by a disk-based version, and reboots the computer with this ROM image in RAM, redirecting acesses to the ROM to the RAM image. MuMapRom uses the MMU and the mmu.library for this trick.

MuMapRom itself cannot write-protect the ROM image in RAM for various reasons, but you should run "MuMapRom" in the startup-sequence after a remapping process to enable ROM write protection, see the installation chapter.

MuMapRom requires a lot of "magic" to do its job, and it does not necessarely run on each board due to its very hacky nature. Before you play with MuMapRom, make sure you read the installation chapter of this guide. If there is any other way to install an alternative ROM, for example by a CPU-board specific program, you will most likely want to use this alternative program.

See also: Caveats of MuMapRom.

MuMapRom may work hand in hand with other reset-resident programs provided these programs are "robust" enough. The problem is that some of these programs may see "two different ROM revisions" when booting up. On first invocation, they see the board native ROM, on their second invocation, the new ROM is mapped in. This may cause problems.

"LoadModule" of the same author accepts this behaivour, provided the kicked ROM has a revision of v37 or above.

Kicking v33 or v34 will typically break a lot of reset resident programs and hence may not work or lead to a guru in case you want to make programs resident under v33. MuMapRom may kick-in v33 or v34 ROMs, but will not run under these anchient releases itself.

MuMapRom contains already a "sort-of" BPPCFix. At the time of writing, it is unclear how effective this implementation is and whether it helps to avoid some P5 specific problems. BPPCFix alone may or may not cooperate MuMapRom.

MuMapRom contains already a "PrepareEmul" patch for ShapeShifter usage. "MuMove4K PREPAREEMUL" is not required, and would in fact collide with MuMapRom due to the hacky nature of both programs. "MuMove4K" without the "PREPAREEMUL" argument - just as preparation step for "MuFastZero" will work, though.

## 1.5 Installation of MuMapRom

Please make sure that you read this chapter carefully and completely before trying to use MuMapRom; if you don't follow the instructions herein, MuMapRom may fail to work.

- Make sure that your system fulfills the requirements of MuMapRom:

2MB chip memory recommended

MuMapRom requires extensive amounts of chip memory for technical reasons .

No ROM persistent 68040, 68060 or ppc.library

The above means that MuMapRom will most likely not work on the boards of some hardware suppliers. I do not own a board of this kind and wasn't able to test MuMapRom on this platform, but I tried to implement mechanisms to make MuMapRom working here as well. You might want to try.

mmu.library supplied 68040 or 68060.library installed

This means that MuMapRom will not work if you want to or have to use a different CPU library than the MMU library supplied versions. You find these separately in the Aminet.

For a 68030 based board, it is recommended to install the mmu.library based 68030.library.

- If you haven't done so, you must now install the mmu.library in LIBS:, the mmu.library supplied 68040 or 68060.library and an ENVARC:MMU-Configuration file. You find further instructions about these issues in the "MMU.guide" in the complete distribution of the mmu.library package.

- Copy "MuMapRom" to whereever you want. It should typically go to the C: directory.

- Remove all other ROM rekick programs from the startup-sequence.

- Remove "PREPAREEMUL" like programs from your startup sequence in case you want to run MuMapRom permanentely. MuMapRom includes this feature already. In case you use "MuMove4K" for this, just drop its "PREPAREEMUL" command line switch, "MuMove4K" itself need not to go.

- Place ROM images of alternative kickstart images in a directory "DEVS:Kickstarts" or any other path you like. Due to copyright reasons, no kickstart images are supplied in this archive.

- Copy the ROM patch files from the "DEVS/Kickstarts" drawer of this archive into the same directory you placed the kickstart images in. These patch files are required by MuMapRom to remove some offending instructions that would otherwise hinder the remapping process. Note that these patch files are specific to the ROM version, you cannot patch a ROM by a patch file that was created for a different release. The patch files should get the name of the corresponding kickstart image file they patch, plus a trailing ".pch". Hence, if the kickstart image file is called "Kick40.63", the corresponding patch file is called "Kick40.63.pch".

Note that kick images are not only version/revision specific, but also hardware specific. An A2000 kick image will NOT work with in an A4000. Similar, patch files are hardware specific. The following patch files are currently available:

Kick33.180.pch is the file for the A500/A2000 v33 (1.2) ROM image. Kick34.5.pch is the file for the A500/A2000 v34 (1.3) ROM image. Kick37.175.pch is the file for the A500/A500+/A600/A2000 v37 (2.04) ROM image. Kick40.63.pch is the file for the A500/A500+/A600/A2000 v40 (3.1) ROM image.

In some cases, you may get around these patch files since MuMapRom has a build-in automatic patch mode that might work for most existing ROM releases.

Experts may want to create their own patch files .

- For temporary usage of MuMapRom, you can run it from the shell, see the command line arguments . Alternatively, MuMapRom can be run from the workbench by taking essentially the same tool types.

In case MuMapRom is able to reboot your system with a different kickstart release, you will notice that booting is noticably slower. This is because the alternative kickstart must be placed in chip ram. Once the system is up, you can remap the mirrored ROM image from there to fast memory. This should be done by running MuMapRom again, either from the command line or from the workbench. You may optionally add the command line switch/tooltype "RELEASECHIP" on this second invocation - this will release the chip memory was required by MuMapRom for the kickstart image, but it will also prevent MuMapRom from being resident. Hence, the next reboot will again happen with the native ROM.

For Short:

- MuMapRom must be run twice for optimal performance. Once to install the alternative kickstart, and once again to speed up the system by placing the ROM image in Fast RAM. The command line for the two invocations should be identical for both runs, even though "RELEASECHIP" is simply ignored on the first run.

The easiest way to guarantee that MuMapRom is really invoked twice is to place it in the startup sequence like this:

MuMapRom DEVS:Kickstarts/Kick40.63 RELEASECHIP

where "Kick40.63" should be replaced by the name of the alternative kickstart to boot with, and "RELEASECHIP" is optional. For further options, check the command line options chapter.

In case no patch file is available for the kickstart version you want to install, the following command line syntax may work:

MuMapRom DEVS:Kickstarts/Kick40.63 RELEASECHIP PATCHFILE=AUTOMATIC

This tells MuMapRom to try to patch the kickstart on the fly; it may or may not work since the patch algorithm depends on some heuristics.

## 1.6 Example Invocations

- For a "dry run", you should simply invoke "MuMapRom" like this if a specific kickstart and its patch file is available:

1.SYS:> MuMapRom DEVS:Kickstarts/Kick37.175

will re-kick the 2.04 of the A2000 ROM.

- For a "dry run" if you do not own a kick patch file, you should try the following:

1.SYS:> MuMapRom DEVS:Kickstarts/Kick40.70 PATCHFILE=AUTOMATIC

will re-kick the 3.1 of the A4000 ROM and leaves the patching to the internal wiring of MuMapRom.

- For removing a resident MuMapRom run it like this:

1.SYS:> MuMapRom DEVS:Kickstarts/Kick40.70 RELEASECHIP

The system will then come up with the native (installed) ROM on the next reboot.

- After ROM-kicking, you should again(!) invoke MuMapRom to speedup the ROM access.

1.SYS:> MuMapRom DEVS:KickStarts/Kick37.175

(to be issued after having MuMapRom'd successfully already)

This will install another fast-RAM based ROM-mirror, leaving MuMapRom resident.

- In case you need your valuable chip memory back after ROM-kicking, but do not care about that MuMapRom is then no longer resident, try the following instead:

1.SYS:> MuMapRom DEVS:KickStarts/Kick37.175 RELEASECHIP

(to be issued after MuMapRom'd)

## 1.7 Example Configurations

We present here how MuMapRom could be used in the startup-sequence to bootstrap the system with an alternative ROM permanentely:

- For systems where chip mem usage is rather unimportant, the following command should be added to "S:Startup-Sequence" behind SetPatch:

MuMapRom DEVS:Kickstarts/<name of the kick image> <alternative patch config>

where "<name of the kick image>" is the name of the kickstart image to bootstrap from, and "<alternative patch config>" either blank, or "PATCHFILE=AUTOMATIC" in case no ROM specific patch file is available.

This command will load the ROM image on first invocation, will reboot the system and will boot up with the alternative ROM. On its second invocation, it will generate a ROM mirror in Fast RAM, hence retaining full speed. The ROM image and the MMU configuration for the ROM image will remain in the chip memory, hence leaving MuMapRom resident.

- For systems where chip mem usage is crucial, however, the following alternative command should be added to "S:Startup-Sequence" behind SetPatch:

MuMapRom DEVS:Kickstarts/<name of the kick image> RELEASECHIP <alternative patch config>

where "<name of the kick image>" is the name of the kickstart image to bootstrap from, and "<alternative patch config>" either blank, or "PATCHFILE=AUTOMATIC" in case no ROM specific patch file is available.

The full command has to go into one line (no line break).

This command will load the ROM image on first invocation, will reboot the system and will boot up with the alternative ROM. On its second invocation, it will generate a ROM mirror in Fast RAM, hence retaining full speed. The ROM image and the MMU configuration for the ROM image will be released on the second reboot, giving back the full chip memory.

## 1.8 Caveats when using MuMapRom

MuMapRom is hack as it hardly can be hackier. On a scale from 0 to 9 ranging from harmless to ugly, this program rates at least 11. I'm not making this up.

Top reasons why not to use this program:

- It is a hack. MuMapRom assumes several non-granted behaivours of the current Kickstart implementation, of the CPU and motherboard hardware. If you don't like hacks, don't run it. There is no other way to replace the ROM by a different ROM than by a hack, sorry.

- MuMapRom will not work in all available environments. It may cause some compatibility problems, depending on the board type, the hardware and the memory configuration. Problems may arise if your board requires a custom MMU setup because various hardware is not integrated in an optimal and conformal way. This goes for boards with the 68040 or 68060.library in ROM, or with the ppc.library in ROM. MuMapRom tries to work around these problems, but it may very well fail. MuMapRom may cause the expansion boards to disappear for some boards, even though I'm trying to prevent this as best as possible. It may even fail completely due to its nature it depends critically on some rather undocumented side effects. MuMapRom will only cooperate with the MMU 68040 or 68060 libraries. MuMapRom will not work wih any other 68040 or 68060 library - simply because they would overwrite the MMU setup installed by MuMapRom.

- MuMapRom requires a lot of chip memory - see Implementation for details why this is so. In fact, it will only run on machines with 2MB chip memory. A graphic card is recommended as otherwise not much chip memory is left. It may work with less chip memory on some rare boards, required that the fast RAM behaives "nicely enough". This is, however, all beyond any board specification.

- MuMapRom requires the board to be reset in an early stage of the boot-up process. On some hardware architectures, the setup of the expansion boards is very critical, and these boards may "disappear" after ROM-kicking. I tried to avoid these problems as best as possible, but do not guarantee for success. If it doesn't work, I will be most likely not able to fix it.

- In case of a "deadend-alert" (i.e. a "Guru") MuMapRom might be unable to reset the system properly. In case the board hangs after a guru-meditation, you may have to reset the board manually. This is a restriction due to the way how the Kickstart handles "Gurus". The advanced hacker will find information about this problem in the Implemenation chapter.

The same problem goes for various "Reset" programs that do not use the proper Os function to trigger a board reset.

- MuMapRom might be unable to support the "Ranger Memory" of some 4.x A2000s. As it is unlikely that MuMapRom can deal with less than 1MB chip memory, this need not to be a restriction.

- MuMapRom may place the system base library "exec.library" in a different place than the native ROM would. This is due to the build-in "PREPAREEMUL" feature, but may break some not-so well written games.

Imporant:

If you need a new/different ROM for your system, you should rather:

Buy the ROM chip and install it if you need it permanentely. This will definitely work, will be most robust and won't cause any problem. If you need two different ROM versions, hardware solutions are available that toggle between two ROMs by means of a little switch. MuMapRom is not supposed to replace these switch-boards.

If you just need an upgrade of several system modules, other less hacky solutions exist. For example, to replace one ROM module by a ROM module of a later edition, you are encouraged to use the "LoadModule" and "MuProtectModules" by the same author.

If you need short turn-around times for software testing, try to find a solution that is adapted to your board hardware (e.g. CyberMap). These programs do not need to touch the MMU, but rather use board specific hardware for ROM remapping. This will typically do better since it will not require as much chip memory as this program, and they know the side conditions of the hardware they have been designed for. MuMapRom tries to be as general as possible, and hence cannot take advantage of board specific features.

## 1.9 Command line options and tooltypes

MuMapRom can be started either from the workbench or from the shell. In the first case, it reads its arguments from the "tooltypes" of its icon; you may alter these settings by selecting the "MuMapRom" icon and choosing "Information..." from the workbench "Icon" menu. In the second case, the arguments are taken from the command line. No matter how the program is run, the arguments are identically.

MuMapRom KICKFILE/A,PATCHFILE,RELEASECHIP/S,ROMINFAST/S

---

KICKFILE/A

Specifies the complete path to an alternative kickstart image to map in. Typically, these kickstart images should go to DEVS:Kickstarts.

This parameter also selects the name of the patch file for the kickstart image by appending a ".pch" to its name. In case you CANNOT supply a proper patch file, you need to enable the automatic patch mode explicitly, see below.

The format of the Kickfile is a complete binary dump of the ROM area without any extensions.

This argument is ignored in case MuMapRom is run at a time an alternative kickstart image is already loaded.

---

PATCHFILE

Specifies an alternative patch file for the kickstart. If you do not give this argument, MuMapRom assumes that the name of the patch file is the name of the kick image plus ".pch". I.e. the patch file of "Kick40.63" would be "Kick40.63.pch". The patch file is required to bypass some early bootstrap code in the kickfile during bootup.

Patch files are specifically for one and only one ROM revision. MuMapRom will check for the correct version of the patch file and will warn you in case the version doesn't match.

This option takes also two special arguments:

PATCHFILE=AUTOMATIC

Tells MuMapRom that no patch file is available and it should try to patch the kickstart image by some heuristics. These heuristics seem to work for all images I've tried so far, but I do not give any guarantee.

PATCHFILE=NONE

Do not apply any patches at all. This will not work for the native ROM images you get for all existing Amiga models since these ROMs typically overwrite the MMU setup very early during booting. Hence, this requires well prepared hand tooled ROM image files. See Implementation for details about what must be patched and why.

This argument is ignored in case MuMapRom is run at a time an alternative kickstart image is already loaded.

---

RELEASECHIP/S

This switch is ignored on the first invocation of MuMapRom. It will be kept care about as soon as MuMapRom finds an alternative ROM image already active.

This switch will release the chip memory occupied by the ROM image and the MuMapRom MMU tables. As these can then be overwritten, MuMapRom cannot remain reset-resident anymore. This side effect can be used to disable MuMapRom, too.

---

ROMINFAST/S

Tells MuMapRom that it should try to place the ROM image in fast memory instead. This requires, however, that the board you're working with supplies its fast memory in a very special and non-guaranteed way; namely, that this ROM never goes away during a reset. This is NOT true for autoconfig RAM, hence it requires a "hacky RAM setup" of the board that bypasses the autoconfig mechanism.

This flag may or may not work, depending on the board you own. It may also depend on how you configured your board.

---

When started from the workbench, MuMapRom knows one additional tooltype, namely:

WINDOW=<path>

where <path> is a file name path where the program should print its output. This should be a console window specification, i.e. something like

CON:0/0/640/100/MuMapRom

This argument defaults to NIL:, i.e. all outputs will be thrown away.

---

## 1.10 Implementation: The Advanced Hacker's Guide To MuMapRom

This chapter contains some implementation details of MuMapRom, and the documentation of the patch files and other miscellaneous data.

Why this exhaustive chip memory usage? Couldn't you have placed the ROM image in any other memory?

No, unfortunately not. There are two problems concering the type of memory, and MuMapRom has to deal with them at once:

- First of all, the RAM must be available at early bootstrap base, before the expansion cards get initialized. RAM on expansion boards will typically go away on a reset, taking a possible ROM image and the MMU table with them. This is unsuitable since this is exactly the ROM that is required for configuring the expansion board in first place. Other CPU boards configure their memory in the "diag.init" and make the RAM available even later. Some other boards make the RAM available immediately, and would allow bootstrap from this RAM, wouldn't there be the second problem:

- The board firmware (and even the kickstart ROM) may run RAM test to check out how much RAM is actually installed. On performing this test, the board firmware and the kickstart may overwrite themselfes, or - even worse - the MMU tables. (Horror!) This would cause an immediate crash.

MuMapRom avoids this second problem by bypassing the chip ram test partially or completely by patching the kickstart ROM image.

What is the format of the patch file?

The patch file format is a standard "SPatch" format, i.e. patch files come in the encoding of the SAS/C "SPatch" utility. They can be created by the SAS "SCompare" tool.

MuMapRom does, however, not implement the "insert bytes" command of "SPatch" completely.

Can I generate my patch file myself?

Yes, provided you know exactly what you're doing and what has to be patched how. You need to provide a monitor, assembly knowledge and the general hacker's wisdom about the kickstart. Save the modified kick image to RAM:, and use the SAS "SCompare" utility to generate the patch. This utility is not contained in this package due to copyright restrictions.

The following issues must be addressed by a patch:

- The initial kickstart checksumming (>v36) or delay loop (<v36) must be removed. MuMapRom must use a re-calibrated delay loop for critical expansion boards.

- The jump into the board-specific F-space must be removed. The F-space gets initialized by the first (native) boot-up and must not be touched on the second boot-up as it may re-configure the MMU.

- The toggle of the CIA overlay bit must be removed as this may temporarely overlay the mirrored ROM by parts of the original ROM and may make the code run into the desert.

- The chip ram upper bound of 0x00080000 for v34 and below must be replaced by an upper bound of 0x00200000 as otherwise MuMapRom could not remain resident.

- The chip mem checksum test must be removed, or replaced by a checksum that ends at the lower bound of the MMU tables and the ROM image in chip memory. See below how to tell MuMapRom where to place the chip mem upper bound.

- The chip mem lower bound must be changed from 0x400 resp. 0x1000 by 0x4000 for the PrepareEmul feature.

- The chip mem upper bound, resp. the upper bound of the chip ram test must be patched by the lower bound of the MMU tables/ kick image. Do do so, you need to patch in some "hints" for MuMove4K:

The kickstart checksum at offset "ROM end - 0x18" must be replaced by the offset of the first longword relative to the kickstart base ("Rom start") that must be patched to the upper chip memory limit. The longword to be patched must be either set to NULL, or must be patched by the offset of the next longword to be patched. Hence, you need to fill in a singly-linked list of relative pointers of the chip memory upper bound whose root is given by the ROM checksum field.

MuMapRom will dereference the nodes of the list and will patch in the proper chip memory upper bound during bootstrap, and will finally re-compute the ROM checksum.

- The initialization of the transparent translation registers in the CPU test code of exec must be stripped away as this would disable the ROM remapping as well.

- For v33 only, the non-working "boot point" logic of the v33 expansion.library ConfigBoard() function must be disabled as it will crash on boards offering "boot point" booting for v34 and above. This is simply a v33 Os bug that would prevent booting with auto-booting host adapter expansion boards otherwise.

How does MuMapRom work?

Hacky, in general. We skip some of the less gory details here and go only for the real tricky parts.

- Step 1: Argument parsing, Library opening, etc.

- Step 2: Check whether a remapped ROM is already available. MuMapRom therefore computes the checksum for both the ROM area with, and the ROM area without the MMU. If both are identically, the origial ROM is supposed to be active. We follow this branch from now on.

- Step 3: The ROM image gets loaded from disk. MuMapRom performs some basic checks here and loads the image into chip memory.

- Step 4: MuMapRom builds a new MMU context. The context is cloned from the default public context, with the MMU tables allocated from chip memory. The following modifications are made:

- All pages marked as "blank" or "invalid" get remapped to 0xfeff0000, a "supposed to be free" address. This is a must since firmware RAM tests may assume to hit "blank space" on a out-of-bounds condition. Remapping these memory areas to a bogus-page could make memory tests think of "infinite" amounts of memory.

- The full zorro-II area except for the ROM area is marked as "cacheinhibited". This might be required again for memory tests, and because the expansion board mapping is not yet clear at the time the MMU configuration gets loaded.

- The Z-III configuration area of 0xff000000 and up is marked as cacheinhibited.

- The ROM area is marked as "remapped". For 512K ROMs, the full ROM area is mapped as a single chunk, for 256K the ROM is mapped in two chunks with the lower chunk as a mirror of the upper. Caching mode is set to cacheinhibited imprecise nonserial as the ROM will end up in chip memory. Some boards will not tolerate the burst accesses into the chip memory that would result otherwise.

- All other mappings remain the same hoping that the current MMU configuration, especially for the Z-III area, will remain valid.

- Step 5: MuMapRom either patches in the data from the patch file, or uses an ad-hoc algorithm to perform the patches by hand. At this stage, the lower end of the MMU tables and the kick image is determined and patched into the ROM. The patches are described above.

- Step 6: MuMapRom builds its resident node, copies the general code and the CPU specific code together. This code is described in detail below.

- Step 7: MuMapRom tries to shutdown the system safely. All filing systems are flushed and inhibited, the corresponding devices are updated and stopped.

- Step 8: MuMapRom removes all resident programs in the kicktags and capture vectors as it is unclear wether they will continue to work with the new ROM. The MuMapRom resident code get installed and the machine get reset.

- Step 9: Exec runs as part of its bootstrap process (of the native ROM) the resident code of MuMapRom. This code checks whether MuMapRom is active already by testing for the MMU. This resident code is run very early in the bootstrap (at priority 104) such that board-resident CPU libraries are (hopefully) not yet running. For the following, we assume MuMapRom found an inactive MMU.

- Step 10:CPU caches are disabled and MuMapRom continues to run in supervisor mode. Interrupts and native DMA are disabled. MuMapRom copies now the first part of the native(!) ROM into the first chip memory page, up to the point where the CIA toggles the overlay bit. The delay loop at the start of the code gets bypassed as well. For the exec bootstrap code, this looks "as if" this code runs in the ROM mirrored to the lower memory area due to the toggled CIA overlay bit, even though it is only a copy.

Following the bootstrap code, MuMapRom patches its own code in which will be described below. Finally, two(!) RESET instructions get patched in front of the bootstrap code as I found that one reset is not enough for some boards to re-set the expansion bus due to the tight timing of high-clocked CPUs. The code continues execution now in the zero-page at the two resets, following the native exec bootstrap.

- Step 11:Exec bootstrap code initializes the F-space and hence supposed-to-be debug-ROMs (in reality, CPU board ROMs ignoring autoconfig, but so what), then toggles CIA overlay. This may remap the real ROM shortly over the copied code in the zero-page, but since the code then disables the overlay bit again, execution will continue at the copy and will run into the copied MuMapRom code.

- Step 12:MuMapRom code starts with a "line" of NOPs to flush the instruction pipeline that might be filled with data from the overlayed ROM. Custom chip DMA and IRQ get now disabled similar to what the native ROM would have done at this time. Since the supervisor stack might have been set to 0x400 for the bootstrap code and we might want to reserve this room for more, we place now the stack over the already executed code which gets stumped upon. It is no longer needed.

- Step 13:The MMU gets activated and hence the native ROM gets magically replaced by the ROM image from disk. Note that we are currently still executing in low chip memory. For the 68060, the FPU gets disabled since exec cannot handle the 060 FPU "frestore" instruction stackframe.

- Step 14:The code checks the ROM revision of the disk based image. For v33 and v34, somewhat more "cheats" must be run as otherwise these ROM revisions wouldn't accept an "execbase" that got build by a different release than its own. Furthermore, execbase might now be in expansion memory and hence "out of sight". v37 and above doesn't care and installs its resident tags after expansion initialization, but v33/v34 does. Hence, we build up a "faked" execbase for v33/v34 at 0x4400, place kicktags and captures in the fake and checksum it. Finally, this mess get installed at AbsExecBase.

- Step 15:MuMapRom starts a delay loop to allow expansion boards to map in properly. This delay loop has been removed from the ROM image. Then, the code jumps into the mirrored disk image in the ROM area.

- Step 16:Since delay loop and F-Space init have been removed from the disk mirror, the exec bootstrap code continues with its checksumming. Since we adjusted version, revision and checksum correctly for v33/v34, resident tags are accepted. Depending on the kickstart version, a chip mem test would be performed which got bypassed by the patch file. Instead, the upper chip memory limit gets accepted we patched in in step 5. The CPU test gets run, but since we removed the TTx register setup, the ROM remains active.

- Step 17:MuMapRom resident code is found due to kicktags, and gets executed again at priority 104. Unlike step 9, we will find the MMU active this time.

- Step 18:MuMapRom replaces the "ColdReboot" vector of exec by code that disables the MMU before resuming. This is required as otherwise we would reset with the wrong (disk based) ROM and would hang. Unfortunately, replacement of the "Guru" reset cannot happen by a simple patch.

- Step 19:MuMapRom locates and removes resident modules called 68040.library, 68060.library and ppc.library to avoid that these modules re-install a different MMU mapping.

- Step 20:The system (hopefully) boots up. SetPatch gets loaded by the startup-sequence.

- Step 21:SetPatch opens the 680x0.library, the 680x0.library opens the proper CPU library, the CPU library opens the mmu.library. The mmu.library scans the already active MMU setup of MuMapRom and builds a new MMU table with the ROM remapped to the disk based image. At this point, it becomes important that the MuLib based 680x0.libraries are used as any other 68040/68060.library would ignore an active MMU configuration. Note that now the MMU is still sub-optimally configured as the ROM in the chip memory area is in use.

- Step 22:MuMapRom is called in the startup-sequence (hopefully!). Step 2) of above now finds a remapped ROM active and hence follows a different control flow than step 3) from there on.

- Step 23:MuMapRom checks wether a resident module of type « MuMapRom » is installed. If not, someone else hacked the ROM in.

- Step 24:MuMapRom allocates memory for a second Os ROM image and copies the image from chip memory over to fast memory.

- Step 25:Since the chip mem upper bound is incorrect and was setup for the needs of MuMapRom and its MMU tables, the missing chip RAM test of the exec bootstrap code is now performed. The MMU table is here already in fast memory due to step 21, as the mmu.library build its own MMU table there and not in chip.

- Step 26:MuMapRom corrects the MMU setup now. All pages remapped to 0xfeff0000 are now marked as "blank" again to bypass illegal accesses. The ROM area gets marked as "copyback", and remapped to the ROM image in fast memory build in step 24. Chip memory setup is corrected to "cacheinhibited, imprecise, nonserial".

- Step 27:In case the "RELEASECHIP" option was found active, MuMapRom checks which chip memory MemHeader could be extended over the incorrect chip memory upper limit to include the memory detected in step 25. This memory is then hacked in and the exec free memory pool gets extended by this memory.

- Step 28:MuMapRom installs the currently active MMU configuration as "old" MMU configuration such that we may run into a reboot safely without using the now released memory of the MuMapRom MMU tables in chip memory. The mmu.library will then keep the active MMU tables up to the point where ColdReboot() finally enters the code of the 68040/68060.library which will disable the MMU before resetting, allowing a clean reboot from the "native" Kickstart ROM.

You should have got the general idea: 1) What happens here is meta- magical, breaks with every good design issue and depends very much on kickstart internals that have never been documented. 2) It requires all components of the MuLib package to work hand in hand. The CPU libraries must load the mmu.library, and the mmu.library must not ignore an active MMU.

## 1.11 History

Release 40.0:

Aminet beta release.

Release 40.1:

This is the first official release.