MuFastZero

Thomas Richter

COLLABORATORS			
	TITLE : MuFastZero		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Thomas Richter	January 13, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

MuFastZero

1.1 MuFastZero Guide

´### ,#. ,#### ,### ####’ #### ,#### ,###’ ####’ #### _____ ,#### ,###’ || ####’ #### | ___ ___ | ___ ,#### ,###’ ---- | | | | | | |___ ####’ #### | | | | | | | ,#####. ,###’ . | |___| |___| |_ ___| ####’##. ,#### ,## ,#### ########’ # ,##’ ####’ `####’ `###’ ,#### ##### © 1999-2002 THOR - Software, ####’ Thomas Richter `##’

MuFastZero Guide

Guide Version 1.14 MuFastZero Version 40.20

The Licence : Legal restrictions

MuTools : What is this all about, and what’s the MMU library?

What is it : Overview

Installation : How to install MuFastZero

Synopsis : The command line options and tool types

History : What happened before

© 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 "MuFastZero" and the "MuFastZero.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 MuFastZero?

MuFastZero is a mmu.library compatible autovector remapper. These vectors are read frequently by the CPU, but are by default placed in the rather slow chip memory. While it is possible to re-map these vectors to faster memory without making use of the MMU, it is more compatible to leave the autovectors at its original place and use the MMU to mirror them to faster memory. This might help - IMHO broken - software to run fine even with the faster system.

The MuFastZero program can also be used to re-map the most important system library to fast memory and hence to speedup your system even more. This requires, though, running an additional tiny program "MuMove4K". Hence, MuFastZero is able to replace certain functions of the Oxypatcher. If used in conjunction with the "ShapeShifter" Macintosh® Emulator and the "PREPAREEMUL" option of the "MuMove4K" program, "MuFastZero" can even speed up the macintosh emulation.

More technical details

1.5 Technical Details

What is precisely the job of MuMove4K and MuFastZero?

Some TurboBoards come with non-auto-configurable memory, i.e. memory that is not located in the Zorro-II, or Zorro-III configuration area. Since this kind of memory is not located at the time the expansion.library is build, the exec and expansion library will end up in Chip mem. To be precise, the following happens on a system startup:

o) The startup code looks for "native" Amiga memory. This is chip mem for first, and the so-called "ranger memory" of the early A2000's in the memory region 0x00c00000 and above, as well as the native motherboard memory of the A4000 and A3000. The system library, exec.library, is then build in this memory as a first step, and this native memory is added to the memory free list of exec.

o) Exec initialized the expansion library, used to mount the hardware expansions. Since only native memory is available at that time, expansion will end up in "slow" memory as well.

o) The expansion library will scan the hardware for auto-configuring boards. Every board found is added to the expansion library ConfigDev's, and memory boards are added to the system memory list.

o) In a fourth step, exec checks if now fast memory is available. If it is, it removes its copy from the slow memory and rebuilds itself in fast memory. Hence, in systems with auto-configuring memory available, the exec library is actually *build twice*.

o) The exec library runs the diag.init. This is the place where additional expansions might "hook in", as for example non-auto-configuring memory. However, since execbase is already constructed at this stage of the startup process, it will be no longer relocated to faster memory if that is made available now.

Hence, on systems without auto-configuring memory, execbase will end up in slow memory, in worst case in ChipMem.

MuMove4K hooks now in at reset time, right below the diag init. It scans the library and device list for entries that reside in chip memory, and builds a private memory pool keeping these memory areas. This is to avoid the graphics library allocating more memory from that area and using it for the Amiga custom chips. However, this memory still remains marked as ChipMem, so that the 68040 or 68060.library marks this area correctly as "non-cacheable". Setting this domain to cacheable won't cause conflicts with DMA operations, though, because of the MuMove4K program, but the CPU would try burst accesses into that memory domain the chip memory can't handle correctly.

If MuFastZero is loaded, it detects the private memory pool build by MuMove4K and relocates this memory to faster memory, using the MMU. Since no graphics buffers have been allocated there, this can be done safely without any risk. The memory attributes get now "private, non-chip", to avoid allocations from this pool.

1.6 Installation of MuFastZero

Installation is pretty simple:

- First, install the "mmu.library": Copy this library to your LIBS: drawer if you haven't installed it yet. It's contained in this archive.

- Copy "MuFastZero" wherever you want.

- In case your system doesn't have any auto-configurable fast mem and you want to re-map parts of the Os from chip ram to fast, you'd keep "MuMove4K" as well.

- Remove all other zero page remappers from your startup-sequence and add the following line:

MuFastZero ON

to enable re-mapping. In case you run the library on top of a third-party 68040 or 68060.library and it MuFastZero complains on startup by a warning message like "The zero page is already remapped", then please add the following option:

MuFastZero FORCENATIVE ON

The program will inform you anyways whether this option is required or not. If you get a warning message like "FORCENATIVE not required", just drop the option. This might happen after installation of a different processor library.

- If you want to use MuFastZero's "FastExec" option to re-map even more parts of the system, "MuMove4K" must be installed and run in the startup-sequence as well. The best place for this program is *IN FRONT OF* the SetPatch command, as one of the very few exceptions. It doesn't take any arguments.

MuMove4K

In case you want to run the Macintosh emulator "ShapeShifter", use the following line instead:

MuMove4K PREPAREEMUL

In case the above call does not work, please run "MuMove4K" like this:

MuMove4K PREPAREEMUL A1200

This implements a "magic of a different color" to provide the same feature.

More technical details

That's all.

1.7 Command line options and tooltypes

MuFastZero 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 "MuFastZero" 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.

MuFastZero ON=FASTZERO/S,OFF=NOFASTZERO/S,FASTEXEC/S,FORCENATIVE/S, MOVESSP=FASTSSP/S,STACKSIZE/K/
MOVEVBR=FASTVBR/S,CLEARVBR/S,IGNORE/S:

ON=FASTZERO

A simple switch. If present, re-mapping is enabled. MuFastZero will complain if it is already installed, or the zero page is already re-mapped or made inaccessible by MuForce.

OFF=NOFASTZERO

Another switch. If given, re-mapping is disabled again and the mirror image is released.

FASTEXEC

Another switch. If this switch is set, MuFastZero will try to re-map parts of the system libraries, most likely the "exec.library" and the "expansion.library" to fast memory. This is only required if you own a board without auto-configuring memory, hence with "exec.library" in chip mem. To enable this command line option, you *MUST* have run the "MuMove4K" program in the startup sequence.

WARNING: This option is not without its quirks! It will, for example, de-activate all reset-proof programs installed after MuFastZero was run. This is because these programs will obviously modify the re-mapped copy of execbase and will leave the original execbase unmodified. Since a reset will de-activate the re-mapping, the modifications will be never seen. Install reset-proof programs before running MuFastZero, and everything will be fine.

If you want to run the ShapeShifer, you've to run the "MuMove4K" program with the "PREPAREEMUL" option instead without any option at all.

More technical details

FORCENATIVE

Disables any other zero page remapper by "brute force". In case MuFastZero is installed, this option will fall back to "OFF" and will un-do the re-mapping in the "nice" way. This option is sometimes useful if other tools or system libraries enabled re-mapping before the mmu.library was loaded, and the re-mapped zero page is "in your way".

This command line option is unfortunately requried for certain third-party processor libraries.

MOVESSP=FASTSSP

Re-maps the supervisor stack to fast memory and helps to increase the performance even more. The old supervisor stack IS NOT released, hence use this keyword not more than once or you waste memory unnecessary.

STACKSIZE

Specifies the size of the supervisor stack in bytes to be allocated, if MOVESSP is used. The default is to use the size of the stack MuFastZero was invoked with, as for example set by the "Stack" shell command.

MOVEVBR=FASTVBR

This option relocates the "exception vector base" from location zero to some other place in fast memory. However, since MuFastZero relocates the zero page to fast memory anyhow, this makes no difference at all except introducing compatibility problems. This option is therefore usually not required unless you do not want to remap the zero page, but still require faster interrupt detection. Leave this option alone, speed won't increase further under normal MuFastZero usage by relocating the VBR.

CLEARVBR

Restores the "exception vector base" back to its default, namely the zero page. This option will fail and exit gracefully if the zeropage is marked as unavailable by debugging tools, e.g. by MuForce.

This option might be useful to play some games which expect the vector bases at address 0L in case you're using a third-party 68040/68060.library which relocates the autovectors without asking you.

IGNORE

Makes MuFastZero silent in case it finds the zero page already remapped. Otherwise, it would complain and result a failure code if this condition is detected.

This option might prove useful in case you want to run a system where you want to choose between two different 68040 or 68060 libraries of which one automatically remaps the zero page whereas the other doesn't.

When started from the workbench, MuFastZero 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/MuFastZero

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

1.8 History

Release 40.2:

This is the first official release. "MuFastZero" does what it is supposed to do, even though the "FASTEXEC" keyword does not yet work. This will be fixed as soon as the "mmu.library" reaches a final state.

Release 40.3:

Fixed the FASTEXEC keyword, added the MuMove4K program to allow re-mapping of the lower memory parts.

Release 40.4:

Internal release: Added support for the 0.30 mmu.library.

Release 40.5:

Fixed a fatal bug when disabling the re-mapping, forgot to copy the zero page back correctly.

Release 40.6:

Added the FORCENATIVE flag.

Release 40.7:

The FORCENATIVE flag was broken, and the OFF option was improved a bit again.

Release 40.8:

The remapper sets now the cache flags of the zero page image according to the cache flags of the mirror RAM. This is of importance in case the zero page gets re-mapped to non-cacheable Z-II memory.

Release 40.9:

Made "ON" the default option if no other option is found. Added the "MOVESSP" and the "STACKSIZE" option for re-mapping of the supervisor stack to fast RAM.

Release 40.10:

Forgot that the "FORCENATIVE" option could case a warning result code in case the zero page wasn't re-mapped at. All other options have been ignored in this case. Fixed.

Release 40.11:

MuFastZero did not unload if run from the workbench. Fixed. The MuFastZero code is now a bit more error tolerant.

Release 40.12:

Added the FASTVBR and CLEARVBR options because it was easy to do and possibly the right program to introduce them. NOTE THAT YOU DON'T NEED THESE OPTIONS IN CASE THE ZERO-PAGE IS REMAPPED ANYHOW.

Release 40.13:

Marking now the remapped memory as MEMF_FAST explicitly to avoid a 0 return value of TypeOfMem() for this memory. Lowered the priority of the rendezvous-port.

Release 40.15:

In case MuFastZero is removed, the unmapped chip memory is now set to IMPRECISE and NONSERIALIZED to gain at least a partial speedup.

Release 40.18:

Added the IGNORE command line option. Added a patch for SumKickData() that will keep resident programs alive even if installed after MuFastZero.

Release 40.20:

Returns now a separate error code in case FASTEXEC has been specified but exec is already in fast memory. Tries now to detect the lower limit of the chip memory automatically and will remap all memory below this boundary. Should help when using the ShapeShifter and "MuMove4K".