Technical Reference Documentation

Commodore RAM Expansion Unit Controller — 8726R1

Title:	Technical Reference Documentation
Subtitle:	Commodore RAM Expansion Unit Controller — 8726R1
Date:	2008-08-02
Version:	1.0
Author:	Wolfgang Moser
Reviewed:	Gideon Zweijtzer, Spiro Trikaliotis (intermediate reviews done for versions 0.7 and 0.8)
Copyright:	Copyright © 2008, Wolfgang Moser
License:	Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the appendix section entitled "GNU Free Documentation License".
Changelog:	2008-05-12, W. Moser: Version 0.1 created, basic document structure, references 2008-05-18, W. Moser: Version 0.1, introduction, more references 2008-05-19, W. Moser: Version 0.2, document structure, basic information blocks, markers 2008-05-22, W. Moser: Version 0.3, register descriptions 2008-05-22, W. Moser: Version 0.3, errata of existing documents 2008-05-22, W. Moser: Version 0.3, operation of the REU controller 2008-05-25, W. Moser: Version 0.4, SWAP hardware implementations 2008-05-25, W. Moser: Version 0.4, document licensing term 2008-05-25, W. Moser: Version 0.4, completing REU controller operation 2008-05-25, W. Moser: Version 0.5, half autoload bug 2008-05-25, W. Moser: Version 0.6, implementation variants of the SWAP operation 2008-05-25, W. Moser: Version 0.7, typographical corrections 2008-05-31, W. Moser: Version 0.8, bit 4 of register 0xDF01 is reset to default state 1 after a transfer 2008-06-01, W. Moser: Version 0.8, jumper J1 setting does influence bank register wrap around 2008-06-07, W. Moser: Version 0.8, comments about the 1541-UltimatePlus implementation 2008-06-21, W. Moser: Version 0.9, description of bit 4 of register 0xDF00 corrected in section 2.2.1 2008-06-28, W. Moser: Version 0.9, chip naming: 8726 vs. 8726R1, applying GNU FDL 2008-07-06, W. Moser: Version 0.9, adding results from explicit verify operation tests 2008-07-09, W. Moser: Version 0.9, adding results from status flag behavior tests 2008-07-21, W. Moser: Version 0.9, I/O area DMA transfer unreliabilities 2008-08-02, W. Moser: Version 1.0, typographical error corrections, public release

Inhaltsverzeichnis

1 Introduction

This document describes the hardware register interface of the Commodore RAM Expansion Unit (REU). This technical documentation tries to describe the hardware implementation aspects of the 8726R1 controller chip of the Commodore REU [1] as they show up to a programmer of this hardware. This should help to get a much better understanding of what this chip does internally.

The intended audience are three groups of people:

- software developers that want to get out more features of the REU and its controller chips,
- software emulation authors that are interested in making the REU's behavior as authentic as possible and
- hardware emulation authors that are going to clone the 8726R1 controller with programmable logic.

There already do exist many technically oriented documents and articles in the Internet. Even the original user manuals from Commodore that came along with the REU all contained a technical section that gave important information to software developers [2], [3], [4]. Further documents were created by programmers who got additional experiences, when working with the REU hardware and programming it. They got a different view on how the hardware should be used efficiently, others concentrated onto the register interface only [5], [6]. Other REU related documents as well as tools and programs are referenced through the C64-Wiki web project [7]. With emulations becoming more precise and authentic and first attempts to emulate the REU in hardware, more hardware level oriented documents were created [8].

This technical reference was mostly composed from collecting fragments from all the other documents and validating their facts through own tests. Other facts presented here were collected from programming experience with the REU, especially regarding some misbehavior that could be called a hardware bug of the 8726R1 controller chip. And then, as a preparation step to this document, a test suite has been created to validate assumptions and to deduce to some of the inner working steps of the REU.

2 8726R1 register interface

This section is mainly a recomposition of all the existing documents. Some details are not described everywhere else. Due to comparisons of emulations (software as well as hardware) of the REU with authentic hardware, such details need to be clarified so that emulations can be made more precise. All these new details were verified with automated tests against authentic original Commodore hardware, if not mentioned otherwise.

2.1 Address decoding

The Commodore REU gets enabled by chip select line /IO2 from the C64's or C128's expansion port. From a programmers view looking at memory addresses, it is mapped to base address 0xDF00. Since the REU controller consists of only eleven directly used registers, most addresses in the 0xDF00 area are not used.

But this does not mean, that all addresses beside the first eleven are so named “open” address space, where other cartridges could map in their resources. The REU controller in fact occupies each and every address in the 0xDF00 area. The first 32 addresses are segmented as following[1]:

1 See also [8], Ruud's description is the first one known to me that explicitly mentions the 32-addresses mirroring

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0xDF00	R,C	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	0xFF	0xFF	0xFF	0xFF	0xFF
0xDF10	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF

Legend:

R,C – The register is read-only, (some) flags are cleared on reading

R/W – The register is a read-write register

0xFF – The address is not mapped to a register, when read, the value 0xFF is put onto the bus

The 8726R1 controller chip does only decode the lowest 5 bits of the address bus, therefore these 32 addresses are mirrored over the whole 0xDF00 area. It is no difference, if the REU gets programmed via accessing registers at base address 0xDF00 or at base address 0xDFA0 for example. Since the REU may be used in conjunction with expansion port multiplexer units that allow to drive several cartridges in parallel, it is wise to only make use of the first eleven registers at base address 0xDF00.

2.2 Register description

This section is mainly a recapitulation and a summary of the technical sections of the Commodore manuals [2], [3] as well as most of the documents available through third party authors [4] and the Internet [5], [6], [8].

2.2.1 Status and command registers:

Address offset	Bits used	Read write	Bit	Function	Mode
0x00	7-0			Status register, read-only, bits 7-5 are automatically cleared on reading, 0x00/0x10 after reset	
		R,C	7	Interrupt pending[2]	1 – if any interrupt is pending (bits 6-5), this requires configuration of an interrupt in the interrupt mask register 0 – otherwise
		R,C	6	End of block	1 – if a transfer is completed[3] 0 – otherwise
		R,C	5	Verify error	1 – if a verify operation stopped with an error 0 – otherwise
		R	4	Size (DRAM), J1 jumper setting	1 – if the REU board is populated with 256ki×1 DRAM chips 0 – if the REU board is populated with 64ki×1 DRAM chips
		R	3-0	Chip version	Current versions of the 8726R1 REU controller chip all carry version number 0. For both package variants, the rectangular DIP as well as the quadratic PLCC, the chip version number is 0.
0x01	7-0			Command register, read/write, 0x10 after reset	
		R/W	7	Execute	1 – a transfer is started either immediately or deferred (using 0xFF00 trigger option) 0 – transfer disabled. The bit is automatically set back to 0, when a transfer took place[4].
		R/W	6	Reserved	The bit is backed by a register. The value last written to it is given back on reading again. Any REU operation does not change its state.
		R/W	5	Autoload	1 – if autoloading registers (addresses, length) should take place after a transfer 0 – if the registers should remain on their last counting state
		R/W	4	FF00 trigger	1 – 0xFF00 trigger option is disabled, transfers are started immediately by setting bit 7 0 – 0xFF00 trigger option is enabled, transfers are started, when bit 7 is

2 The exact behavior of this flag was not backed by a dedicated test yet

3 With verify operation this bit gets also set, when a verify error occurred on the last or next to last byte

4 It is not investigated yet, if the bit is set back to 0 just before the end of the transfer or directly after it was started

Address offset	Bits used	Read write	Bit	Function	Mode
					set also and a write access to address 0xFF00 occurs. The bit is automatically set back to 1, when a transfer took place[5].
		R/W	3-2	Reserved	The bits are backed by a register. The values last written to is given back on reading again. Any REU operation does not change their state.
		R/W	1-0	Transfer type	00 – data is transferred from C64/C128 to REU 01 – data is transferred from REU to C64/C128 10 – data is exchanged between C64/C128 and REU 11 – data is verified between C64/C128 and REU

2.2.2 Addressing registers

Address offset	Bits used	Read write	Bit	Function
0x02	7-0	R/W	7-0	C64 base address LSB, lower 8 bits of C64 base address, read/write, 0x00 after reset
0x03	7-0	R/W	7-0	C64 base address MSB, upper 8 bits of C64 base address, read/write, 0x00 after reset
0x04	7-0	R/W	7-0	REU base address LSB, lower 8 bits of REU bank base address, read/write, 0x00 after reset
0x05	7-0	R/W	7-0	REU base address MSB, upper 8 bits of REU bank base address, read/write, 0x00 after reset
0x06	2-0			REU bank pointer (selecting 64KiB memory pages), read/write, 0xF8 after reset
		R	7-3	Unused bits, they are always set to 1 on read accesses[6]
		R/W	2-0	Select pages 0...7
0x07	7-0	R/W	7-0	Transfer length LSB, lower 8 bits of the byte counter, read/write, 0xFF after reset
0x08	7-0	R/W	7-0	Transfer length MSB, upper 8 bits of the byte counter, read/write, 0xFF after reset

2.2.3 Control registers

Address offset	Bits used	Read write	Bit	Function	Mode
0x09	7-5	R/W		Interrupt mask register[7], read/write, 0x1F after reset	
			7	Interrupt enable	1 – if REU interrupts are enabled 0 – otherwise
			6	End of block mask	1 – if end of block interrupts is enabled 0 – otherwise
			5	Verify error mask	1 – if verify error interrupt is enabled 0 – otherwise
			4-0	Unused bits, they are always set to 1 on read accesses	
0x0A	7-6	R/W		Address control register, read/write, 0x3F after reset	
			7-6	Addressing type	00 – both addresses are incremented on transfers (default) 01 – the REU base address is fixed 10 – the C64/C128 base address is fixed 11 – both addresses are fixed
			5-0	Unused bits, they are always set to 1 on read accesses	

2.3 Operation of the REU controller chip

This section starts with explaining the default behavior upon a data transfer between the C64/C128 and the REU and then progresses with telling something about the more exotic features and options.

5 It is not investigated yet, if the bit is set back to 1 just before the end of the transfer or directly after it was started

6 This holds true for original 8726 REU controller chips and known extensions to it (when only write accesses are intercepted). Clone chips and REU emulations may have extended this register for either write-only or read and write accesses.

7 The behavior of the REU controller upon different interrupt mask settings was not tested in detail yet

2.3.1 Simple transfers

After some startup initializations, where the programmer determines the REU's size and preinitializes registers, a minimal effort transfer can be established as following (it is assumed that registers 0xDF09/0A are set to default values, all interrupts off, addressing mode 00).

1. Write the C64/C128 start address into registers 0xDF02/03
2. Write the REU start address into registers 0xDF04/05/06
3. Write the transfer length into registers 0xDF07/08
4. Issue an immediate C64/C128-to-REU transfer without register autoload by writing the value 0x90 into register 0xDF01
5. The REU controller now takes over the bus via a DMA request,
6. On each system clock cycle the REU controller transfers one data byte from the currently addressed C64/C128 memory location into the addressed REU memory location
7. The REU controller then increments the two-byte C64/C128 address and the three-byte (two bytes and 3 bits) REU address. Both registers roll over to 0, when appropriate
8. If the transfer length register contains the value 0x0001, the REU stops transferring data, otherwise it decrements the transfer length register and repeats with the 6. step
9. The REU controller chip clears bit 7 and sets bit 4 of register 0xDF01 and sets bit 6 of the status register 0xDF00[8].
10. Read back status register 0xDF00 to clear bit 6 which gets always set after a normal transfer ended. This does not depend on the actual generation of interrupts

2.3.2 Data transfer via Direct Memory Access (DMA)

Whenever a transfer is initiated by programming register 0xDF01 accordingly, the REU controller takes over the processor bus as soon as possible. That way it can access each address of the C64/C128 on its own without needing to cooperate with the main CPU. But there is another DMA device within each C64/C128 computer that has got higher priority on doing DMA transfers, the VIC-II video chip. Whenever it needs to access the bus (so named “badlines”) other devices immediately have to stop accessing the bus, being it the 6510 main processor or the Commodore REU.

When the REU has got access to the bus and it is not intercepted on its own by the VIC-II, it does work down the transfer as programmed before and by the procedure described above. Depending on the transfer mode for each clock cycle either one byte is transferred from C64/C128 memory to REU memory or vice versa resulting in a data transfer rate of nearly 1MiB/s. This data rate also holds true for verify operation[9], but when the swap operation is programmed, the data rate is degraded to only around 500KiB/s.

2.3.3 Immediate command execution vs. 0xFF00 trigger option

As show in section 2.3.1, when the 0xFF00 trigger options is not configured (bit 4 set to 1), the DMA operation immediately starts after the 6510 main processor completed that last write operation. It then does work out the transfer, clears execution bit 7 of register 0xDF01 and sets the trigger option bit 4 back to default state 1.

---

8 It is not tested yet, if the command execution bit is reset just after the beginning of a new transfer or really at the end of a just finished transfer. Special equipment may be required to read out register 0xDF01 while a transfer is in progress. The same goes for the point in time, when the status register bits are set

9 This is not explicitly tested yet

If a transfer is programmed for 0xFF00 trigger option, bit 7 of register 0xDF01 remains set to 1 and no DMA transfer takes place, until the trigger event happens. For this reason command execution could be taken back by clearing bit 7 of register 0xDF01, as long as the trigger event did not happen.

The trigger gets activated by writing some arbitrary value into C64/C128 address location 0xFF00. For the C128 at this address location there sits the Memory Mapper Unit (MMU) which organizes different memory banks, ROMs and the I/O area within this computer. By using write to this memory location as trigger event, with one write access a decent memory configuration can be programmed and a transfer started just after that. With the next cycle available to the main CPU the former memory configuration can be restored. On the C64, the MMU (actually a simple 6-bit processor port) sits at address locations 0x00 and 0x01 which makes MMU programming and DMA execution not so smoothly combinable.

The deferred transfer execution with the 0xFF00 trigger options is of crucial use, when RAM or ROM "below" the I/O area should be accessed by the REU and its DMA controller. When the I/O area is banked out and replaced by RAM it is not possible to execute a transfer by writing to register 0xDF01, therefore you need some sort of redirection.

After a transfer got executed via a 0xFF00 trigger event, bit 7 of register 0xDF01 is cleared again and bit 4 (0xFF00 trigger option) is set back to its default state of 1.

2.3.4 Normal operation, transfer length

On normal transfers with the autoload option not set, the counter registers are left "as-is" upon transfer completion. Following the procedure described in section 2.3.1, the transfer length counter register always stops with value 0x0001 upon completion of a transfer; except when a verify error occurred or the autoload option is enabled. Even when this register was programmed with value 0x0001 (transfer a block with a length of only 1 byte), it stops with value $0x0001_{10}$.

The other two address counter registers were left at addresses that are one greater than the address where the last value was transferred from or to.

A programmed transfer length value of 0x0000 means to transfer a block with a size of 0x10000 bytes.

2.3.5 Autoload vs. normal operation

To reduce the need for reprogramming the REU's registers on repeating block transfers, the REU supports autoloading registers 0xDF02...0xDF08 with arbitrary values programmed once. Such repeated block transfers for example occur, when someone wants to fill the video buffer with a fast exchanging sequence of different pictures. The C64/C128 video buffer base address as well as the transfer length are the same for each single transfer. Only the REU's base address would change from transfer to transfer. With the autoload option enabled, the REU gets back the formerly programmed values into registers 0xDF02...0xDF08, after a transfer was finished. That way the programmer only needs to set up the REU's base address (registers 0xDF04...0xDF06) properly and can then start another transfer.

To accomplish the autoload feature, the REU contains an additional set of shadow registers for the range of registers from 0xDF02 to 0xDF08. Whenever to any of these registers a value is written to, it is stored into the shadow register first and transferred into the counter register from there. The exact implementation is discussed later again along with a side effect of the autoload option implementation (see section 3.2).

When the autoload option is enabled, the REU controller always reloads the values from the shadow

10 This was covered by a special test, probing different transfer lengths, including 0x0001 as start value

registers upon transfer completion. The shadow registers are not manipulated from the REU except, when a programmed register write operation from the C64/C128 occurs. So if the programmer executes several transfers with the autoload option disabled and he furthermore does not reprogram one or all registers, then the value in the appropriate shadow register do not change. For the next transfer with the autoload option enabled that old unchanged value gets restored upon transfer completion.

2.3.6 Verify operation

It is not useful to use the autoload option in conjunction with a verify transfer operation. Normally, when a verify error occurs, the both address counter registers stop with addresses that are one greater than the addresses where the different values were detected. Under special circumstances, when the exact mismatch addresses are of no interest and many blocks need to be checked against a given pattern, the autoload option may also be useful in conjunction with the verify option and bit 5 of register 0xDF00.

Whether a verify operation failed or not should always be checked via bit 5 of register 0xDF00. It is in no way sufficient to check the transfer length register 0xDF07/08, if it counted down to the value 0x0001. It may also contain the value 0x0001, when the REU detected a difference on the last or next to last verified value pair[11].

When a verify operation aborts with a verify error on the last or on the next to last compared value or if it completes without a verify error at all, then in all three cases the end-of-transfer flag is set. In the first two cases, this flag is combined with the verify error flag.

A pseudocode implementation of the verify operation may look like as following (it is assumed that registers 0xDF09/0A are set to default values, all interrupts off, addressing mode 00). In comparison to the simple transfer pseudocode, the description needs to become much more complex, so that the obeyed behavior is met[12]:

1. Program C64/C128, REU address and block length registers
2. Issue an immediate verify operation without register autoload by writing the value 0x93 into register 0xDF01
3. The REU controller now takes over the bus via a DMA request,
4. On each system clock cycle the REU controller compares one data byte from the currently addressed C64/C128 memory location with the addressed REU memory location
5. If the compared data is not equal to each other, the verify error flag is set (bit 5 of the status register 0xDF00)
6. The REU controller then increments the two-byte C64/C128 address and the three-byte (two bytes and 3 bits) REU address. Both registers roll over to 0, when appropriate
7. If the transfer length register contains the value 0x0001, the REU stops comparing data and proceeds with step 10
8. The REU controller decrements the transfer length register
9. If the verify error flag is set, the REU stops comparing data, otherwise it proceeds with jumping back to step 4
10. If the transfer length register contains the value 0x0001, the end-of-transfer flag is set (bit 6 of the status register 0xDF00)
11. The REU controller chip clears bit 7 and sets bit 4 of register 0xDF01

11 This is a side effect of the length counter register implementation

12 REU behavior is discovered and verified with systematic tests

2.3.7 Address counting modes

Under normal operation, when register 0xDF0A is set to value 0x00 respectively 0x3F, both address pointers are incremented on each DMA transfer cycle. Via configuring the address control register it is possible to not let the REU controller increment either the C64 address pointer or the REU address pointer or both. This can be used for special operations like:

- Filling C64/C128 memory with an arbitrary byte value that was stored into a decent memory location in the REU. By fixing the REU address pointer, always that byte is transferred into each C64/C128 address
- Filling REU memory with up to 65536 Bytes per one DMA transfer. A byte value is stored somewhere in the C64/C128 memory map, and then a transfer issued with this C64/C128 address programmed and configured to be fixed
- Implementing a very fast right-shift byte shifting engine with the help of the SWAP operation and either the C64/C128 (shifting queue contained within REU) or the REU (shifting queue contained within C64/C128 memory) address being fixed. An initial value is swapped with the first byte from the queue. That swapped byte is then swapped with the second byte from the queue and so on, until the byte from the last operation can be placed into the beginning of the queue manually
- Fixed C64/C128 address configurations may also be used to either do sampling certain hardware registers (CIA port, SID, VIC) with a sample rate of ~1MiB/s[13]. Using the other transfer direction may be of some use for writing data into register locations of the VIC or fixed address locations that are otherwise indirectly loaded into the VIC (sprite pointer, 0x3FFF/0x7FFF/0xBFFF/0xFFFF)
- Using a configuration with both address pointers fixed can be used to create handshaking flag signals for the CIA ports with a trigger rate of nearly 1Mi events per second[14]
- In combination with the SWAP operation, fixing both addresses could result in a fast alternating transfer pattern. The addressed C64/C128 register is written with a value, that was read in the previous SWAP cycle. If this is not the same as the value read for the current SWAP cycle, the value alternates for each SWAP cycle
- In combination with the VERIFY operation, fixing both addresses may be used to construct some sort of low latency wait loop with programmable timeout. Under the assumption that a decent C64/C128 register contains some value that is expected to change some time in the future, then the REU may be loaded with exactly this current value. It is then programmed to execute a verify operation with both addresses fixed and a transfer length value that determines the timeout. As soon as the verified register changes its value, the REU aborts the operation with a verify error. Since DMA stops immediately, the intended action can take place with a latency of less than two clock cycles[15].

2.3.8 Address counter wrap around

Address counter wrap arounds occur, if the desired counter is not configured to fixed mode. When the C64/C128 address counter reaches value 0xFFFF inmid a transfer and there are still more bytes to transfer as programmed via the transfer length register, the C64/C128 address counter wraps around to address 0x0000 and increments further from there.

---

13 As pointed out in section 3.3, transfers to or from the I/O area are not reliable on every C64/C128 model

14 See footnote above and section 3.3

15 This heavily depends on the number of opcodes that are needed to actually program that action. At least active waiting with the help of the REU produces less jitter than typical “L1: CMP xxxx/BEQ L1” loops
Since transfers to or from the I/O area are extremely unreliable for verify operation, such a low jitter wait routine is practically of very limited use, see also section 3.3

The REU address counter is constructed from a 16-bit portion ("behind" registers 0xDF04 and 0xDF05), plus the 3 bit of the banking register register 0xDF06, forming an 19 bit counter. This counter can count the whole span of 512kiB, neglecting some special behaviour of the 1700 REU, as below. Thus, there is a wrap-around from value 0x7FFFF to 0x00000 (from 0xFFFFFF to 0xF80000, when the unused stuck 1 bits are taken into account also).

For the Commodore 1700 REU, where jumper J1 is closed (see [1]), there is some special behavior that cannot be watched for any other REU type. Additionally to the rule above another wrap around occours from REU address 0x1FFFF to 0x00000 (0xF9FFFF to 0xF80000). No other REU address border is affected, so if e.g. banks 0x02, 0x03, ..., 0x06 (0xFA...0xFE) are programmed and a 16-bit wrap around occurs, it is going into the next following bank and does not get redirected to bank 0x00 (0xF8)[16].

Take note that the REU address register wrap around is independent from the installed memory size and populated DRAM memory banks – if not taking jumper J1 configuration into account as described above. There may occur another sort of wrap arounds:

1700: For a decent REU address not equal to bank 0x00000 RAM at address 0x00000 is accessed
1764: REU bank register locations 0x04 to 0x07 (0xFC to 0xFF) are not backed by DRAM
1750 expanded to 1MiB or 2MiB: The REU counter wrap around 512kiB does not wrap from one of the extended 512kiB banks into another one. Exchanging banks can only be done by writing into register 0xDF06, but is not automatically done through wrap-around

2.3.9 Status flag behavior

With bits 5 and 6 of the REU status register 0xDF00 the controller is able to signal, if a verify error occurred or if a transfer did finish. On most operations of the REU, the end-of-transfer flag bit 6 is set after a transfer was initiated and finished. When doing a verify operation and a verify error occurred, the end-of-transfer flag is not set, except if the verify error occurred on the last or second last compared byte of a block to verify. As a rule of thumb it can be said that the end-of-transfer flag is set, when the transfer length counter reached value 0x0001, regardless if a verify error occurred or not.

When a verify error occurred, then bit 5 of the status register is set. This is independent from the fact, if the verify operation did end also and the end-of-transfer flag is set and the transfer length counter reached value 0x0001. Therefore always the status flags should be used to decide if a verify operation ended without a verify error instead of checking if transfer length counter reached value 0x0001. Moreover, when doing a verify operation, the verify error flag has got higher priority than the end-of-transfer flag. If both flags are set, then a verify error occurred even if the last value of the block of bytes to compare was reached.

It should be noted that the flags have to be cleared before each transfer operation. The REU controller does not clear the flags on its own, when a transfer operation is executed. When the flags were set on a previous operation, they remain set after the following operation. This may lead to confusing results, if a first verify operation did end with a verify error and the next one did end without error. The verify error is set nevetherless and the end-of-transfer flag also. Interestingly this feature could be used to verify several blocks one after one but without checking the status flags

16 See also [3], section 5.4: "If the selected bank is equal to the maximum for a particular memory configuration, then the wrap is to bank 0."

each time. At the end it could be checked, if any of the previous verify operations did fail.

The very same goes for the end-of-transfer flag. If the first operation was a standard transfer or a verify operation that did not fail and the second operation did abort with a verify error, then the end-of-transfer flag remains set even after the second operation.

In summary the both flags are set with a boolean OR operation from the REU controller. The REU controller is not able to clear the flags. This can only be done from the C64/C128's register interface to the REU controller.

2.3.10 Using interrupts

The REU may be programmed to generate a hardware interrupt on two possible events: When a DMA transfer ended for any programmed operation and when a verify error occurred while the desired operation is executed. To enable any interrupt, bit 7 of the interrupt mask register has to be set. Along with bit 7 either bit 5 or 6 or both have to be set also to enable any of the two possible interrupt sources[17].

When any interrupt source is enabled and the desired event happened, bit 7 of the status register signals that this interrupt needs to be serviced and acknowledged[18]. Bit 5 and 6 of the status register are normal flags that are set independently from the interrupt mask register configuration. To clear the interrupt flag in an interrupt service routine, the status register is just read. This read access does not only clear interrupt flag bit 7, but the other two status flags also.

The interrupt mask register is not changed by the REU controller in any way, its programmed value remains stable, if an interrupt/clear sequence took place. It is strongly recommended to clear the status register, before the mask register is configured to enable interrupts and a new transfer initiated via the command register[19].

Normally the interrupt flags are of limited use to the “average programmer” since the CPU always has to wait until a DMA transfer is finished. The next processor command after a transfer was initiated is executed, when the transfer finished. The verify error flag signals, when the REU stopped comparing data, because a pair was different. From reading the address registers one can find out where exactly the difference is located. The end-of-transfer flag is of limited use, because there is no other possibility for the REU, but to end a transfer after it got executed.

From a more software architecture oriented point of view, it may be good design practice to implement a common handler for all end-of-transfer conditions. This could be a true interrupt handler. The handler decides, if a transfer exited normally or if a verify error condition needs to be handled. With this the main program only needs to initiate the transfer, but is not required to explicitly call an end-of-transfer handler after. This is just done automatically by letting the REU signal /IRQ and do the actual post-transfer actions within the interrupt service routine.

2.4 Errata of existing documentations

When comparing manufacturer supplied manuals which each other and documents that were created by developers as well as users, some differences can be determined. Often these differences arise from features and register settings of the REU that are rarely used. Additionally there are sentences contained within that may lead to misunderstandings.

---

17 If both interrupt sources are enabled, the interrupt service routine (ISR) should always check for a verify error first, since if a verify failed on the last or second last comparison, then also the end-of-transfer flag is set

18 Enabling only bit 7 of 0xDF0A does *not* enable an interrupt source

19 This is recommended by the manuals [2], [3], [4], section 2.3.9 gives a hint about unexpected behavior, if flags are not properly cleared. Actual interrupt (mis-) behavior is not backed by an explicit test yet

2.4.1 Register 0xDF00

There is some sort of general confusion about the meaning of bit 4. Is is often said to determine the size of the RAM available to the REU. The manual for the 1700/1750 REU [3] explains that the bit determines, if 128K (bit 4 = 0) or 512K (bit 4 = 1) of RAM are installed. The manual for the 1764 REU [2] tells that when this bit is set, this means that 256K of RAM are installed. Document [6] refers to the 1764 manual while [5] summarizes the both manuals that on a 1700 RAM expansion the bit is cleared and set otherwise (1764 or 1750). [8] tells that this bit simply reflects the setting of a jumper that is located at position J1 on the REU mainboard. The schematics for the REU [1] contain a note that explicitly tells to open the jumper for the 256K and 512K models only, but let it closed for the 128K variant.

In fact it is the REU controller chip that needs the size information configurable by jumper J1, see section 3.5 for more. The controller needs to know how to drive the equipped DRAM chips correctly. From a programmers point of view, bit 4 of register 0xDF00 is of not much use. It does not tell, how much RAM is installed in a REU, but reflects the state of the hardware configuration option J1 only.

Bits 3 to 0 determine a version identifier of the REU. Since all consulted documents state that these bits are always set to 0 which is also verified by testing different REU models, it can be concluded that the version bits don't identify the REU model (1700, 1750, 1764). Rather it is more likely that the version bits are configured directly in the REU controller chip to identify its mask revision or new variants with bug fixes or extensions to the existing 8726R1 design. Probably the version 0,0,0,0 correlates to the mask revision denotation 'R1'.

2.4.2 Register 0xDF01

Bit 4, 0xFF00 trigger option is explained in the opposite way for the German manual of the 1764 RAM expansion. There is written: “4 – FF00 1 = Enable FF00 decode” which is wrong. Maybe this mix up happened on translating the correct English manual with some sort of copy'n'paste error with the Autoload option a line above. Further the Commodore manuals tell that the FF00 decode option is “cleared” after a 0xFF00 triggered transfer took place. This in fact means that bit 4 is set back to default state 1.

Bit 6, 3, 2 are denoted as “reserved” with no further explanations in the official Commodore manuals. [6] mentions that these bits were always left to 1 judging from a sample program from the official Commodore manuals. [5] tells that these register bits are normally cleared (0), [8] similarly tells that for these bits *always* a 0 would be given back on read accesses.

Explicit tests on the behavior of these bits showed that the REU controller saves any state programmed into the bits. Their states even don't change, when the REU does some operations on its own (transferring data). In sum it looks like as if these 3 register bits could have been used for additional transfer options like with the Autoload feature and the 0xFF00 trigger option. These bits are backed with a register and read/write logic within the REU controller chips. It's only that their output lines are not used for anything useful within the chip.

3 8726R1 re-implementation

This chapter is thought as an additional help to emulation authors, be it hardware developers programming some sort of programmable logic or software developers that are addicted to Commodore computer software emulation.

This chapter discusses further hardware variants of the Commodore RAM expansion units, which

consists of clone hardware and extensions to the original hardware. It details out some unexpected behavior of the REU register interface that can be deduced to a decent internal construction of logic building blocks. Further, some implementation alternatives are discussed on how to reconstruct the operational behavior of the 8726R1 with minimum effort.

3.1 Hardware variants and extensions to the Commodore REU

Although always called the “8726R1” chip in this document, there do exist packages that don't contain the “R1” suffix. One without the R1 suffix was produced in week 27 of the year 1987 (2787). Another PLCC chip with the R1 suffix was produced in 5286, half a year before. Another R1 suffixed chip in a DIL package was produced in 4386. Therefore it is assumed that all produced silicon dice are of the very same R1 revision.

3.1.1 Chip packaging

The 8726R1 REU controller chips come in two chip packaging variants: a rectangular 64-pin ceramic DIL package with somewhat reduced pin distance and a quadratic 68-pin plastic PLCC package. Both variants behave absolutely identical as found by programming and testing the REU controller register interface. It is said that the DIL package variants can often be found in the Commodore 1700 model and maybe in early 1750 models [9]. They also appear in shrink wrapped clones of the Commodore 1750 REU from manufacturer Chip Level Designs (CLD, see [4]), as well as a similar product from Creative Micro Devices (CMD), called the 1750XL.

The REU controller is able to drive two independent DRAM banks with a data bus width of 8 bits. To implement this, the controller uses two pairs of /RAS and /CAS signals to control address selection for each DRAM bank separately [1]. Each DRAM bank can be populated with either 64ki×1 or 256ki×1 DRAM chips which has to be configured with jumper J1 (see also 3.5).

3.1.2 REU modifications and expansions

Early in the history of using the Commodore RAM expansion units there appeared instructions on how to extend the 1700 and 1764 models to appear as a 1750 REU [10]. For the 1700 model all existing DRAM chips had to be unsoldered and replaced by 256ki×1 types, also Jumper J1 had to be cut. For the 1764, on the empty bank solder had to be removed from the wholes and the bank then equipped with eight more 256ki×1 DRAM chips.

Some time after a circuit appeared that was used to expand an existing 1750 or an already 512kiB expanded 1700/1764 to 1MiB or 2MiB of DRAM [11]. The circuit “watched” write accesses onto the REU controller bank register 0xDF06. For such a write access it redirects the unused bits 3 and 4 into own registers. Each DRAM chip was stacked up with another 3 DRAM chips (piggybacking), but separating the /CAS signal. The two latched bits 3 and 4 were used to drive a 2-to-4 decoder that selected one of the four DRAM stack layers with the help of the /CAS signal line.

As mentioned by the author of the 2MiB expansion, the circuit comes with two little drawbacks. REU counter wrap around does occur for each DRAM stack layer separately (512kiB each) since the REU internal bank registers bits 0 to 2 and the externally added bits 3 and 4 don't know of each other. So for each 512kiB wrap around the preselected DRAM stack layer remains the same. The other drawback from a programmers point of view is that the extension bits 3 and 4 to the bank register 0xDF06 are write only. One cannot determine which bank is selected, therefore some bookkeeping has to be done by software.

3.1.3 REU clone hardware with 8726R1

Later on professional manufacturers went to create shrink wrapped designs of the Commodore 1750

REU and its expanded 2MiB version. Instead of using 1-bit DRAM chips they used so named VRAMs (Video-RAM due to its intended use) that got internal banks to store several bits per address. Additional logic was added (one programmable logic chip instead of a heap of TTL chips) to recreate the /CAS and /RAS signals to accommodate for the VRAMs[4]. Another clone also integrated the 2MiB extension circuit into the programmable logic chip [9]. Since both hardware clones used original and factory new 8726R1 controller chips, these have been 100% compatible to the 1750 REU or the expanded 2MiB version.

3.1.4 REU emulations

Later software emulations did overcome the drawbacks of the 2MiB expansion. They extended the REU bank register to make use of all 8 bits and all the bits became writable and readable. Furthermore wrap around was implemented in a way so that the full 16MiB address room is stepped through without each 512kiB "stack layer" being caught in itself.

3.1.5 Recommendations to emulation programmers

To support existing software that relies on the exact behavior on any of the existing REU models 1700, 1764, 1750, 1MiB or 2MiB extended 1750, the following points may be obeyed:

- REU emulations of well known sizes should be emulated as precise as possible to support existing "unforgiving" software
- Emulation of the 1700/1764/1750 REU should limit register 0xDF06 to bits 0 to 2 and also support the desired wrap around behavior
- Emulation of the 128kiB 1700 REU should have bit 4 of register 0xDF00 be cleared and the very special wrap around implemented as described in section 2.3.8
- Emulation of the 1764 and 1750 REU and any expanded model should have bit 4 of register 0xDF00 be set
- For the 1764 REU the non populated second DRAM bank should be emulated well[20]
- The 1MiB and 2MiB expanded versions of the 1750 REU should be emulated precisely regarding the non-readability of bits 3 and 4 for the extended bank register 0xDF06 as well as the wrap around being limited to 512kiB DRAM stack layers
- Specialties of the shrink-wrapped VRAM equipped versions of the REU should be emulated with respect to the /CAS and /RAS recreation[21]
- If a REU emulation is expanded by further registers and extended operation modes, the version information bits of register 0xDF00 may be used to reflect a new REU controller chip revision. It may make such an extension explicitly incompatible to existing software

Current software REU emulations are missing special behavior, when configured to sizes of 128kiB, 1MiB or 2MiB[22]. It may be a matter of reducing software complexity and performance, if all the little quirks described above are implemented or not. Software written for the extended 1MiB and 2MiB versions of the REU with the implicit internal 512kiB wrap around should mostly run fine on a 16MiB REU version without that implicit 512kiB wrap around. The same holds true for the additional wrap-around of the 1700.

---

20 It was not investigated or tested until now, how correct emulation of an empty DRAM bank should look like. From some quick manual tests it looks like if normally 0x00 is transfer from an unpopulated DRAM bank. But if some values were transferred to unpopulated banks first, then values 0xFF can be read back for a short time. This behavior may have something to do with internal buffers or latches of the the 8726 controller chip

21 If such special behavior can be detected through programming the register interface or doing specially crafted RAM test patterns. A test program that comes along with the 1750XL clone has got some dedicated DRAM columns test

22 These observations were done in June 2008 with C64 software emulators CCS64 and VICE

3.2 Half-Autoload bug of the 8726R1

As explained in section 2.3.5, to implement the autoload feature of the Commodore REU controller, shadow registers are needed for all the registers from 0xDF02 to 0xDF08. The shadow register's bit width for 0xDF06 should correlate to the emulated register 0xDF06. The shadow registers are programmed with new values by writes from the C64/C128 side only. The shadow registers cannot be written from the REU controller itself or by any operational side effect (e.g. counter wrap around).

On every write to the shadow register or when the autoload option is configured and a transfer ends normally or with a verify error, then the counter registers are loaded with the values from the shadow registers. When values are read from the C64/C128, then contents are delivered directly from the counter registers, not the shadow registers.

After a hardware reset, the shadow registers are initialized the same as their accompanying counter registers on a hardware reset:

Shadow register	Bits used	Read write	Function
0x02	7-0	W	Shadow register for the C64 base address LSB, 0x00 after reset
0x03	7-0	W	Shadow register for the C64 base address MSB, 0x00 after reset
0x04	7-0	W	Shadow register for the REU base address LSB, 0x00 after reset
0x05	7-0	W	Shadow register for the REU base address MSB, 0x00 after reset
0x06	2-0	W	Shadow register for the REU bank pointer, 0xF8 after reset
0x07	7-0	W	Shadow register for the Transfer length LSB, 0xFF after reset
0x08	7-0	W	Shadow register for the Transfer length MSB, 0xFF after reset

The Half-Autoload bug occurs when the registers 0xDF02 to 0xDF05, 0xDF07 or 0xDF08 are written after a transfer was done with the autoload option *disabled*. When register 0xDF02 is programmed with a new value, then at the same time register 0xDF03 is reloaded with the value from its shadow register and vice versa. The very same goes for register pairs 0xDF04/0xDF05 and 0xDF07/0xDF08. The is no such coupling detected between the REU bank register 0xDF06 and 0xDF04/0xDF05.[23]

Looking at this register behavior from a more hardware oriented point of view, then this leads to the following conclusion. The register pairs from above (in fact the counters behind) as well as their accompanying shadow registers are constructed as *16 bit registers*. Although each 8 bit half can be loaded on its own, the counting and the shadow-to-counter register transfer operate on a 16 bit basis. That way an old remaining value from the other half of the shadow register gets transported into the counter register and appears as (one byte) half autoloaded value. Judging from the effect that there is no coupling between the bank register and the other REU address counters, bits 0 to 2 from 0xDF06 look as if they were added on a single-bit basis.

The observations that led to the conclusion above are also the reason why write accesses onto registers 0xDF02 to 0xDF08 are transferred to the shadow registers only, but not directly to the counters itself. Only with such an implementation and the intermediate 16-bit transfer between a shadow register and the desired counter, the Half-Autoload bug can be emulated correctly with minimal effort.

3.3 Unreliable DMA transfers for I/O area exchanges with the REU

The german manual for the Commodore 1764 Ram Expansion Units explains that with having the C64 address counter held fixed it would be possible to exchange data with external devices via port

23 The whole behavior of the Half-Autoload bug register behavior as described here is backed by an explicit test suite

chips in the I/O area[24]. The manual for the 1700/1750 REU that was built for the C128 computer gives a note that it would be possible to DMA to any I/O device except for the 8563 video display controller (VDC) chip[25]. The manual for Chip Level Designs' clone hardware goes further by mentioning that the 8563's internal RAM, the 6510/8502 main processor's I/O ports at addresses 0x0000 and 0x0001, the 8722 memory management unit chip (MMU) and the REU itself cannot be used for DMA transfers[26].

From observations with testing the REU controller for all the features and behavior described in this document it needs to be said that the REU does not operate fully stable when transferring to or from the C64/C128 I/O area in the range from 0xD000 to 0xDFFF. The grade of transfer instability depends on the actual C64 or C128 mainboard revision. When doing data transfers from the main computer into the REU, then for older models some byte every 200 to 1000 bytes is copied wrong.

Actually not the byte value in its own gets corrupted, but the REU controller is not able to *address* the correct byte. It just takes another one from the C64/C128's I/O area instead.

The transfer reliability with the I/O area also seems to depend on the operation mode done. Verify operations were a lot more unreliable than simple transfers. Verify operation with both addresses being fixed could be used to watch a decent I/O area register to change its value. If the register does not change like with the sprite X register for example, then the REU should count down until the transfer length register reaches value 0x0001. But only, when the value within the REU was programmed to the same value before.

With the REU reliability bug, the REU controller does not count down to 0x0001 in most cases and on all tested C64/C128 mainboard revisions[27]. Often only 20 verify operation cycles were done until a non existing verify error was detected.

3.4 8726R1 register hideout

The 8726R1 REU controller does hide its register programming interface from its own DMA transfer engine. This was concluded for the tests done in the previous section. Even with the partial DMA transfer unreliabilities for transfers from the C64/C128's I/O area, it could clearly watched that for transfers from I/O area 0xDF00 the transferred contents looked like “open address space” data; more or less random data patterns.

For comparison the same random patterns can be watched, when sampling data from I/O area 0xDE00 to 0xDEFF. The REU controller does either actively disable the register interface, when a DMA transfer is in progress or the register hideout is the result of some other timing interference in the complex C64/C128 bus protocol.

3.5 Bank register vs. DRAM banks (jumper J1 and size bit)

An instruction on how to expand a 1764 REU to 512K [10] explains that jumper J1 tells the 8726R1 REU controller chip that the size of the single RAM *chips* is either 64ki×1 or 256ki×1. This makes sense, because with the bigger RAM chips, addresses need to be managed in a different way for row select (/RAS) and column select (/CAS) signals. And then even the original Commodore manual for the 1700/1750 REU [3] has got a section 5.6 that tells something about a bank configuration (the DRAM banks)[28], the BS signal mentioned there is also found in the schematics [1]. For BS=0 both banks have to be equipped with 64ki×1 chips and for BS=1 256ki×1 should be used.

---

24 See [2], section 4.2 on page 4-7, third paragraph

25 See [3], section 5.3, third paragraph

26 See [4], section “Special Features”, third paragraph

27 The three different C64 mainboard models tested were assembly numbers 250407, 250425 and 250466

28 Citation: “This bit in the status register reflects the state of an external jumper which was preinstalled in the factory”. But the manual doesn't mention which bit in the status register it refers to.

Since the 1764 and 1750 REUs both use the same type of DRAM chips, jumper J1 is cut for both. For the 1764 there is only one of the two 8-bit DRAM banks (this is different to the 64KiB pages selectable by REU banking register 0xDF06) populated with chips.

Note that when jumper J1 is not cut (1700, 64ki×1 chips), then a special wrap around occurs for the banking register 0xDF06, see section 2.3.8.

3.6 How to implement the SWAP transfer option

Since the SWAP operation mode of the REU has been rarely used in the past and is not backed by some tests yet, it is difficult to tell how this mode is implemented at hardware level. It is known that the transfer rate is degraded to ~500kiB/s for the SWAP operation since double the amount of data needs to be moved between C64/C128 and REU (see [3], section 5.5).

A straight-forward hardware design would be to use two temporary 8-bit registers within the REU controller chip. Within the first transfer cycle the registers are filled with contents from the addressed C64/C128 location and the addressed REU location respectively. Within the second transfer cycle, these internal register contents then are transferred cross-wise to the REU and the C64/C128 respectively. The SWAP operation therefore needs an additional counter bit to decide between the first SWAP cycle and the second one.

Since registers cost a lot of gates (die size, complexity), Commodore hardware engineers may have tried to find an optimized design that only needs one internal temporary register. There are some facts to consider when thinking about this. In the C64, half of the bus time for each cycle is always used by the VIC-II. When a so named badline occurs, both half of a bus cycle are taken by the VIC-II. When the REU communicates with the C64 it also has to obey the VIC-II bus accesses. It can then re-use these half-cycles for REU internal only accesses. Most probably DRAM refresh cycles would take place here. For the SWAP operation, reordering these REU internal DRAM accesses may lead to a 1-temp-register optimization for the SWAP operation:

1. Within cycle 1, transfer contents for a decent address from C64 into the REU temp register The REU has no internal DRAM transfer to do here, this can then be used to do two DRAM refresh cycles (within the two cycle half)
2. Within the first half of cycle 2, transfer contents from REU for a decent address to C64
3. Within the second half of cycle 2, transfer contents from the REU internal temp register into a decent address of the REU

For step 3, the C64's bus is occupied by the VIC-II in the second half, but the REU can use that time to transfer the saved temp register value into its DRAM. The normally happening DRAM refresh half cycle (assumption) has to move into the 1st cycle of the SWAP operation.

The exact implementation for the above 1-temp-register-SWAP operation depends on which half of a cycle is occupied by the VIC-II. Maybe it is more useful to implement a “cycle inverted” hardware:

1. Within the first half of cycle 1, transfer contents from a decent address of the REU into the internal temp register
2. Within the second half of cycle 1, transfer contents from C64 for a decent address to REU
3. Within cycle 2, transfer contents from the REU temp register into C64 for a decent address. The REU has no internal DRAM transfer to do here, this can then be used to do two DRAM refresh cycles (within the two cycle half)

4 Further notes

The test suite and tests mentioned in this document don't replace any of the existing Commodore RAM Expansion Unit (REU) tests. The tests behind this document mostly check out the register behavior of the REU controller chip. Existing REU test suites in general check if the REU does work as expected and if the RAM does not lose stored data. Find two examples described below.

Each Commodore Ram Expansion Unit comes with a test/demo diskette that contains a basic test program. See http://www.zimmers.net/anonftp/pub/cbm/demodisks/other/index.html for these two files:

1764-utility.d64.gz contains a 1764 REU (256kiB) test program for the C64 (1764 ramtest.bas/.bin) that does pattern based tests as well as perhaps some addressing mode tests (column/row)

1700-demo.d64.gz contains 1700 and 1750 REU test program for the C128 (ramtest and ramtest.bin) that does some very simple, but time consuming RAM test. There is not much user feedback except some flickering pixel bar in the upper left corner

Jens-Michael Gross wrote a third party Ram Expansion Unit tester for the 1700, 1764, 1750, 1MiB expanded 1750 and 2MiB expanded 1750. His tool can be found on public FTP servers, see http://ftp.giga.or.at/pub/c64/tools/c64/cartridge/index.html for a file named:

RAMTEST.SFX This file contains a self extracting archive for the C128 and C64 as well (load with ",8" only). Contained within is the program "ram-test 17xx/64", Jens-Michael Gross' "17xx Ram Expansion Testprogramm", © 1990. It autodetects the type and size of original and expanded REUs and tests each bank with byte patterns.

A GNU Free Documentation License

Version 1.2, November 2002

Copyright (C) 2000,2001,2002 Free Software Foundation, Inc.
51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

A.0 PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

A.1 APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to

text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

A.2 VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

A.3 COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or

with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

A.4 MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor

acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

A.5 COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

A.6 COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this

License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

A.7 AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

A.8 TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

A.9 TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

A.10 FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

A.11 How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (c)  YEAR  YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
```

```
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled "GNU
Free Documentation License".
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

References

[1] Commodore: Service Manual 1750/1764, , http://www.itsm.uni-stuttgart.de/staff/schuldes/online/serv17xx.pdf.

[2] Commodore Büromaschinen GmbH: 1764 RAM Expansion Module - 256K Erweiterung, 1987, .

[3] Commodore Electronics Limited: 1700/1750 RAM Expansion Module - User's Guide, 1985, http://project64.ath.cx/hw/1700re10.zip.

[4] Chip Level Designs: The Super 1750 Clone User's Guide, 1991, http://project64.ath.cx/hw/1750cl10.zip.

[5] Richard Hable: REU Programming, 1993, http://ftp.giga.or.at/pub/c64/library/reu_programming.html.

[6] Marko Mäkelä: REU Registers, , http://ftp.giga.or.at/pub/c64/library/reu_registers.html.

[7] C64-Wiki.de: REU, 2008 , http://www.c64-wiki.de/index.php/REU.

[8] Ruud Baltissen: E-REU, , http://www.baltissen.org/newhtm/e_reu.htm.

[9] Maurice Randall: 1750XL Question....., 2005, http://forum.cmdrkey.com/viewtopic.php?t=227#840.

[10] Scott A. Boydman: Expanding your 1764 Ram Expansion Unit from 256 to 512 Kilobytes of Memory, , http://ftp.giga.or.at/pub/c64/library/exp-1764to512kB.html.

[11] Andrew E. Mileski: Beyond 512 kb - The Two Megabyte REU, 1989, http://ftp.giga.or.at/pub/c64/library/exp-1750to2MB.html.