Gmod2 for Commodore 128

by Daniël Mantione

October 2021

In the past few years, Gmod2 has been a popular choice as a format to release games and other software for the Commodore 64. Besides its low cost, its EEPROM memory makes it attractive. Commodore 128 developers, even if they wish to develop for the Commodore 128, have few pre-existing cartridges to choose from and therefore have to invent the wheel.

This document describes a proposal for a Commodore 128 native version of the Gmod2 cartridge. The cartridge is only modified in a minimal way.

Required changes

Changes are required in the following fields:

- EXROM
- Bank size from 8K to 16K
- Flash programming

EXROM

In the C64 Gmod2, bit 6 of the bank switch register is connected to EXROM. The Commodore 128 activates cartridges through the MMU rather than EXROM/GAME and therefore in the C128 version of Gmod2, bit 6 will no longer be connected to EXROM. It will still be connected to the EEPROM.

Bank size from 8K to 16K

The Gmod2 makes the flash ROM visible via ROML, which on the Commodore 64, is an 8KB memory range $8000..$9fff. On the Commodore 128, ROML is mapped into the 16KB MIDROM region from $8000..$bfff when activated through the MMU. This means that a bank is twice as large as on the C64, thus a 512KB flash ROM will consist of 32 16KB pages rather than the 64KB pages on the C64.

Bit 0-5 are used for bank switching on the Commodore 64. On the Commodore 128, bit 0-4 will be used for bank switching. This means the cartridge can be expanded in the future to 1MB flash ROMs, or even 2MB if bit 6 is also used (but in that cause programmers would need to be more cautious to not disturb the EEPROM).

For now the Gmod2 C128 cartridge is 512KB just as the C64 version and only bits 0-4 are used for bank switching, so less lines are shared with the EEPROM as on the C64.

Flash programming

The Gmod2 C64 cartridge can be programmed from the C64. Cartridge ROM normally doesn't receive chip select signals for writes, but the Ultimax mode is exempt from this and will receive selects on write. Gmod2 C64 uses this and dynamically activates Ultimax for writes on addresses >= $8000 and uses ROMH to connect to the write line of the flash ROM.

In the Commodore 128 native mode, there exists no Ultimax mode and this removes the possibility to get write selects on the cartridge port. As we cannot get selects, when cannot depend on the ROML/ROMH lines for writing. Bus snooping has to be used instead.

When bit 7 of the bank switch register is active, any write by the CPU in the Commodore 128 MIDROM region (regardless of the MMU memory configuration) will go to the flash ROM, in addition to the chip selected by the PLA. The PLA normally selects memory in the current RAM bank for writes to the MIDROM region, thus the write will go both to flash and to RAM.

Code that writes to the flash will be less complex on the C128 than on the C64. On the C64, setting bit 6 is required, but this deactivates the cartridge for reading and read accesses are required during flash operations. Code needs to deal with different memory addresses, ROML for read and ROMH for write and continuously switch bit 6/7 to switch between reading and writing. This complexity is not necessary on the Commodore 128, only bit 7 needs to be set and can remain set during the entire flash procedure and code only needs to deal with the MIDROM region.

Bank switch register layout

The changes can be summarized in a description of the function of the bits the bank switch register:

bit	r/w	Gmod2 for Commodore 64	Gmod2 for Commodore 128
0	w	Bank selection bit 0	Bank selection bit 0
1	w	Bank selection bit 1	Bank selection bit 1
2	w	Bank selection bit 2	Bank selection bit 2
3	w	Bank selection bit 3	Bank selection bit 3
4	w	Bank selection bit 4 EEPROM data in	Bank selection bit 4 EEPROM data in
5	w	Bank selection bit 5 EEPROM clock	EEPROM clock
6	w	EXROM control EEPROM chip select	EEPROM chip select
7	rw	Flash ROM write enable (w) EEPROM data out (r)	Flash ROM write enable (w) EEPROM data out (r)

Gmod2 128 from the viewpoint of the developer

From the viewpoint of the developer, using the Gmod2 is very similar as on the C64. You can simply poke the bank number that you want to access into $DE00 and it will be mapped into

$8000..$bfff. Code for using the EEPROM can remain unchanged, it is exactly the same as on the C64.

When switching from C64 to C128, programmers mainly need to be aware of the different startup scheme. The Commodore 128 can transfer control to the cartridge either before the computer has been initialized, or during the boot sequence. This is controlled by a flag in the cartridge. The Commodore 128 maps the cartridge in memory, checks the signature and flag and transfers control to $8000.

Control is transfered to your cartridge with the RTI instruction. This enables interrupts, but the KERNAL is not visible, when booting your cartridge, the C128 has mapped the cartridge ROML to the MIDROM memory region and ROMH to the HIROM region. Therefore, an IRQ will crash the computer. Therefore the first instruction in your cartridge startup routine should be a SEI in order to disable interrupts.

The next thing you want to do is to switch the C128 to a more convenient memory configuration. In Commodore 128 memory bank 8..11, normally the ROM connected to ROMH is mapped to the HIROM region. As ROMH has no meaning for Gmod2, we can just as well map the KERNAL into HIROM. This can be done by writing $0a to MMU register $ff00. In this configuration $0000..$7fff will be mapped to RAM in RAM bank 0, $8000..$bfff will be the current Gmod2 bank, $c000..$cfff the screen editor, $d000.$dfff I/O and $e000..$ffff KERNAL.

It will now be safe to enable interrupts again and the machine is fully functional for your application, for example we can use the KERNAL to print a message on the screen.

Cartridge startup code can be as follows:

```
* = $8000
        sei                 ; C128 has enabled interrupts, disable!
        jmp cartstart       ; jump to cartridge start
        nop
        nop
        .byte $FF           ; We want control when the C128 has initialized and is booting
        .byte $43,$42,$4D   ; "CBM" string, C128 checks this for cartridge presence
cartstart:
        lda #$0A            ; Make the KERNAL visible
        sta $FF00
        cli                 ; Now it’s safe to enable interrupts
        ; Print some text….
        lda #>texttoprint
        sta $03
        lda #<texttoprint
        sta $04
        ldy #0
next:
        lda ($03),y
        beq ready
        jsr bsout
        iny
        bne next
ready:
       ; We are done, rts will make the C128 to continue its boot process
        rts
texttoprint:
        text $0d,"cartridge booted...",0
```

At this point, the machine is in a configuration that a Commodore 64 developer will feel comfortable, as the memory layout is not that different from the C64s 16K cartridge mode.

Programming is not that different from programming the C64, only you have much more RAM available. You can access the rest of the RAM by poking different configurations into the MMU register at $ff00, but note that the C128 makes your life easy with the FETCH and STASH routines. Using FETCH and STASH, even code that executes directly from cartridge ROM can read/write anywhere from/to the C128's 128KB of RAM.

Therefore a viable strategy is to execute your program directly from cartridge ROM, use the area from $0000..$7fff as "hot RAM", and use FETCH/STASH access the rest of the RAM as "cold RAM". You can also use the KERNAL's LOAD/SAVE routines to load files directly into the "cold RAM". This way you don't have to do any bank switching in your own code.

However, if this doesn't suit your needs, the MMU configuration at $ff00 is at your service. It is advisable to write to the $ff00 register directly instead of using the 16 preset banks in the C128, because none of the preset banks contains the ROML+KERNAL combination that is useful for the Gmod2.

The meaning of the bits in the $ff00 register are documented here:
http://www.oxyron.de/html/registers_mmu.html

The Gmod2 bank switch register contains 0 on reset, so the first flash ROM bank is mapped and you can switch to a different Gmod2 bank by writing the bank number to $de00.

The above cartridge startup routine is less complex than a cartridge startup routine for the C64 and when coding for the C128, quite often you will discover that the C128 code is less complex than the C64 code.

During development, you may want to disable the autostart flag. To do so replace the `.byte $FF` with `.byte $00` in the example above. In this case the Commodore 128 will boot to BASIC so you have the full machine at your disposal to do development work. To boot your cartridge from BASIC, type:

```
BANK 8
SYS 32768
```

Books to read to learn more about programming the Commodore 128

The text in this document is intended as a small introduction to get you started, there are obviously many small details that are out of scope of this document. Recommended books to consult if you want to learn more are:

- "Commodore 128 programmers reference guide" by Commodore
- "Commodore 128 intern" by Data Becker
- "Mapping the Commodore 128" by Compute! Publications