The “Seahorse Board” for Gigacart is in-system programmable, with a few caveats:

- **Has a hardware bug**:
    - Flash pin 16 (WE#) is tied to ground which write protects the lowest 128k (on this chip it’s lowest, anyway).
    - It’s tied directly so we can’t override it, and it’s tied under the flash chip so we can’t just cut the trace
    - The pin should be floating. It might be possible to very carefully cut the pin off, but it’s probably safer to change the software to work around it.
    - This bug exists on the “Mutant Centipede” board, but those flash were top protect so I didn’t see it in my testing.
    - We can lose 128k without much worry for Dragon’s Lair, at least.
- **Requires a different CPLD load** that enables the behaviour below
- **Requires a modification to the TI console**:
    - On the cartridge connector, tape off the GROM Select pin
    - Solder a wire from there to the 74LS138 pin that selects memory range >E000
        - an 8k range is required for full access to the chip
        - Other blocks could be used, but I’m assuming >E000 is easiest

The replacement CPLD load has the following setup:

- No GROM support (wouldn’t work anyway, and I needed the space)
- Three extra data bits are defined in the latch write to >6000:
    - >01 and >02 are still the most significant latch bits, as always
    - >04 enables LSB writes to the flash chip at >E000
    - >08 enables MSB writes to the flash chip at >E000
    - >10 directly controls the flash reset line
- When any writes to the flash chip are enabled, all reads at both addresses are disabled
    - This is necessary because if you need to change the latch or config bits by writing to >6000, the read-before-write will otherwise interrupt whatever process you are doing.

*Be careful in EasyBug - it’s very easy to type “E000” instead of “**M**E000” to access the >E000 range. ‘E’ is the ‘execute’ command, so this makes EasyBug jump into random code.*

Hardware notes

See CF7 analysis for CF7 notes, but at the basics:

- >5Exx = CF card read registers
- >5Fxx = CF card write registers
- Running in IDE mode
    - TI.Adr Read Write

    - 01 Even Data Even Data (has some overlap issues with 03, use dups if conflicting)
    - 03 Error Features
    - 05 SectorCnt SectorCnt (counts down to 0 after an access)
    - 07 Sector.No Sector.No
    - 09 Cyl.Low Cyl.Low
    - 0B Cyl.High Cyl.High
    - 0D Card/Head Card/Head
    - 0F Status Command
    - 11 Even Data Even Data (Duplicate, but not available on CF7?)
    - 13 Odd Data Odd Data (Duplicate, but not available on CF7?)
    - 15 n/a
    - 17 n/a
    - 19 n/a
    - 1B Error Features (Duplicate)
    - 1D Alt.Status Device Ctl
    - 1F Drive.Adr reserved
- Soft Reset:
    - Write >04 to >5F1D to set the reset bit
    - Write >00 to >5F1D to clear the reset bit
- Init:
    - Write >81 to >5F03 to request 8-bit mode
    - Write >EF to >5F0F to command ‘set features’
- Set sector address (LBA mode, flat addressing):
    - Write >E*x* (drive 1, LBA) to >5F0D, where *x* is bits 24-27
    - Write bits 16-23 to >5F0B
    - Write bits 8-15 to >5F09
    - Write bits 0-7 to >5F07
- Check status (before and after commands):
    - Read >5E0F
        - >80 = Busy
        - >40 = Ready (RDY)
        - >20 = write fault
        - >10 = Ready (DSC)
        - >08 = Data Request (DRQ - read or write)
        - >04 = ECC occurred
        - >02 = always 0
        - >01 = error occurred (see Error Register)
    - When Busy, ignore all other bits. You want both Ready lines to be set. DRQ means there is data ready to read or the flash is expecting data to be written.
- Read sector(s):
    - Set sector address (I did not check if it’s preserved or incremented)
    - Write desired sector count to >5F05 (this is NOT preserved)

        - Ensure flash is ready
        - Write >20 to >5F0F to command Read sector(s)
        - Wait min 400uS
        - Wait for flash status to go not busy
        - Check for error
        - Check that DRQ is set
        - Read 512 bytes per sector (in 8 bit mode, 256 in 16 bit mode) from >5E01
            - Repeat the status check after every sector
        - Note that only the MSB is hooked up if you're in 16 bit mode
- Write sector(s):
    - Set sector address (I did not check if it's preserved or incremented)
    - Write desired sector count to >5F05 (this is NOT preserved)
    - Ensure flash is ready
    - Write >30 to >5F0F to command Write sector(s)
    - Wait min 400uS
    - Wait for flash status to go not busy
    - Check for error
    - Check that DRQ is set
    - Write 512 bytes per sector (in 8 bit mode, 256 in 16 bit mode) to >5F01
        - Repeat the status check after every sector
        - Also check for write fault!
    - Note that only the MSB is hooked up if you're in 16 bit mode

TI Console Modifications:

The GROM port is modified by disabling the GROM Select line, and replacing it with an active low select from the >E000 address. There is a convenient 74LS138 on the motherboard with an unused >E000 output that can be tapped.

This is only possible on my new PCBs (or by manually moving CPLD pins 88 and 89 to ROM1 and ROM2, !WE to ROM4 and !OE to Spare).

A different CPLD load is also used, which recognizes the following, as well as can toggle OE and WE appropriately).

- Cartridge is read-only at >6000
- Writes to >6000 toggle the page latch as per normal, except there are extra data bits:
    - Bits >01 and >02 are part of the latch as normal
    - Bit >04 enables LSB writes to the flash chip
    - Bit >08 enables MSB writes to the flash chip
    - Bit >10 is mapped to flash reset
    - No writes are performed to the flash at this address.

- Writes to >E000 are routed to the GROM select pin and treated as access to the flash, gated by the LSB/MSB control writes as above. No read is performed to the flash at this address.
- Reads even to >6000 are blocked when any writes are enabled, to prevent read-before-write interference.
- It's unclear whether 16-bit writes to the flash will work correctly, Easy Bug didn't let me test that. But they are possible if bits >04 and >08 are both selected.

Gigacart flash access (based on above hardware mod):

- For all command writes, the actual address (excluding least significant bit) and two least significant data bits are mapped to the latch, and so may be preserved in all of the following.
- Likewise the three flash control bits are independent of each other and may be preserved when manipulating only one of them.
- Reset flash chip:
    - Write >00 to >6000 to set !reset low (note this will also zero the latch!)
    - Write >10 to >6000 raise !reset and release it (this will also zero the latch!)
- Read CFI data (both chips?)
    - (no unlock needed)
    - Write >18 to >6000 to enable MSB writes and clear latch
    - Write >98 to >E0AA to activate CFI mode
    - Write >10 to >6000 to enable reads
    - Read data from >6020 onward (even bytes if in 16-bit mode, or always?)
    - Write >F0 to >E0AA to exit CFI mode
- Unlock sequence (both flash chips):
    - Write >18 to >6000 to enable MSB writes and clear latch
    - Write >AA to >EAAA for step 1
    - Write >55 to >E554 for step 2
    - (Required for each command unless accelerate unlock bypass is on)
- Write buffer (both flash chips?)
    - I believe it's 64 bytes on the old chip, and 512 bytes on the new chip
        - A "page" is 32 bytes long
        - The write buffer line is 64 bytes (old) or 512 bytes (new)
        - A sector is 64k in size
        - However, you can only use word size writes to use the whole buffer, so it's 32 bytes on the old chip and 256 bytes on the new chip.
        - ECC is maintained on the new chip as long as you write 32 bytes at a time
    - Must write to a page boundary ('adr')
    - Perform Unlock Sequence
    - Write to >6xxx to set latch as needed
        - Include >18 bits to release reset and write to MSB
    - To the page address (at >Exxx), write >25 to command buffer write

        - To the same page address, write >1F to indicate 32 writes will follow
        - (not tested) change the latch bits to >1C to enable MSB and LSB
        - Write 64 bytes to the correct addresses (>Exxx), starting at the page.
            - If using 16 bit writes, this will be 32 instructions but the hardware will do 64 writes to the flash
            - If using 8 bit writes, you will have to change the latch bits to allow MSB and LSB writes. The change does not impact the count.
            - The documentation suggests that out of sequence writes are okay, so you may be able to do all MSBs and then all LSBs.
            - The chip counts down the NUMBER of writes, not where they are!
        - At the page address (at >Exxx), write >29 to start the process. An error will occur if the correct number of writes did not happen.
        - Write >10 to >6000 to enable reads
        - Read back the last loaded address (at >6xxx) until it matches, or until bit >20 is set (indicating error) or >02 is set (indicating aborted). The flowchart recommends double checking in case it was in the process of changing to the valid data. A reset can be used to clear the error.
        - It may be a little more reliable to read the status register, which will give an absolute status instead of the above ‘probably’.
- Single write (both chips: byte only I believe since we don’t have a 16-bit bus):
    - Using this mode will disable ECC on the newer flash
    - Perform Unlock Sequence
    - Write >A0 to >EAAA to initiate single write command
    - Write to >6xxx to set latch as needed
        - Include >10 bit for reset and LSB or MSB bit as desired
    - Write the desired byte to the desired address (>Exxx)
    - Write >10 to >6000 to enable reads
    - Read back the address (at >6xxx) until it matches, or until bit >20 is set (indicating error). A reset can be used to clear the error.
- Accelerate unlock bypass (both chips - allows to skip the unlock sequence):
    - Perform unlock sequence
    - Write >20 to >EAAA to enter unlock bypass state
    - This allows even a chip erase to occur in just two writes to ANY chip address, so maybe is not a great thing to enable!
- Reset bypass (both chips):
    - Write >18 to >6000 to clear latch and enable MSB (if needed)
    - Write >90 to >EAAA for step 1
    - Write >00 to >E554 to disable bypass mode
    - Actual addresses apparently don’t matter
- Chip Erase (both chips):
    - Perform unlock sequence
    - Write >80 to >EAAA to setup
    - Repeat unlock sequence

        - Write >10 to >EAAA to begin chip erase
        - Chip erase will take about 10 minutes(!)
        - Write >10 to >6000 to enable reads
        - When readback (>6000) returns >FF the erase is complete.
- Sector Erase (both chips)
    - Perform unlock sequence
    - Write >80 to >EAAA to setup
    - Repeat unlock sequence
    - Write >30 to sector address (>Exxx) to start
    - Write >10 to >6000 to enable reads
    - When readback (>6xxx) returns >FF the erase is complete.
- Read status register
    - Write >18 to >6000 to clear latch and enable MSB (if needed)
    - Write >70 to >EAAA to access status register (any mode)
    - Write >10 to >6000 to enable reads
    - Read anywhere
        - There are two bytes, the MSB is all reserved
        - Not clear from the docs if the LSB only can be retrieved, or if both are needed
        - 16-bit read may work as well