Gameboy Bootstrap ROM

From GbdevWiki

Contents

The DMG bootstrap

On July 17, 2003, neviksti published that he had been able to extract the contents of the Gameboy boot ROM from a DMG-01 on the Cherryroms.com forums. The boot ROM is a bootstrap program which is a 256 bytes big piece of code which checks the cartridge header is correct, scrolls the Nintendo bootup graphics and plays the "po-ling" sound.

When the Gameboy is turned on, the bootstrap ROM is situated in a memory page at positions $0-$FF (0-255). The CPU enters at $0 at startup, and the last two instructions of the code writes to a special register which disables the internal ROM page, thus making the lower 256 bytes of the cartridge ROM readable. The last instruction is situated at position $FE and is two bytes big, which means that right after that instruction has finished, the CPU executes the instruction at $100, which is the entry point code on a cartridge.

Neviksti managed to read out this memory area by opening the CPU of a Gameboy he got from Duo, and looking at it with a microscope. That way he managed to read the code bit by bit.

The SGB bootstrap

On September 16th, 2009, Costis Sideris was able to extract the Super Gameboy bootrom using a form of clock glitching involving an FPGA. See Costis' page describing the dumping (http://www.its.caltech.edu/~costis/sgb_hack/). The clock crystal for the SGB was disconnected and instead controlled by the FPGA. After viewing an address bus trace (which shows the address as the bootrom is reading/writing to the $FFxx i/o space, but not the data), he found which exact clock cycle the write to the $FF50 register (which disables the bootrom) was. He then caused the FPGA to clock the SGB CPU at 4 times the normal speed for that write cycle only. This caused the CPU to glitch, the disable write to fail to properly occur, and the program counter to continue past there to $100 and onward, into cartridge rom space. A program was placed in that area which wrote the bootrom out byte by byte to the FPGA (using a bogus cartridge-address-space address which the FPGA recognized).

When the Super Gameboy is turned on, the first part of the bootrom is not very different from the DMG one; it sets up sound registers and clears vram, but also writes 0x30 to the $ff00 keypad port (which the sgb uses as a bit-banged serial output port in addition to its keypad reading function). After that however, it clears WRAM bytes $c05f to $c058, and then copies the cartridge header ($104 to $14f) to WRAM at $c000-$c057, placing count and sum bytes at $c000-$c001, $c010-$c011, $c020-$c021, $c030-$c031, $c040-$c041 and $c050-$c051. This data is then bit-banged as a giant packet over the $ff00 port to the snes. See Just Dessert's disassembly at the Bannister MAME subforum (https://forums.bannister.org/ubbthreads.php?ubb=showflat&Number=54179#Post54179). Unlike the DMG and CGB bootroms, the bootrom does NOT lock out the cartridge if the header sum or logo is wrong; its the SNES which does that!

The CGB bootstrap

Neviksti has also tried to extract the bootstrap from a Gameboy Color (CGB-01) CPU. However, because that CPU uses NAND ROM and is laid out in a different way, he had no success in extracting that ROM. Based on some limited preliminary decapsulation work done by Dr. Decapitator, it was determined that the CGB CPU die has three roms on it: one 256 bytes, one 512 bytes, and one 1792 bytes.

On September 21st, 2009, Costis Sideris was able to extract the Gameboy Color bootrom using a combination of clock and power glitching involving an FPGA. See Costis' page describing the dumping (http://www.fpgb.org/?p=17). The clock crystal for the CGB was disconnected and instead controlled by the FPGA, as well as the 3.3v power pin for the CGB CPU. After viewing an address bus trace (which shows the address as the bootrom is reading/writing to the $FFxx i/o space, but not the data), he found which exact clock cycle the write to the $FF50 register (which disables the bootrom) was, but attempting a similar clock glitch attack as the SGB didn't work. Instead, he used a much more 'brute force' attack after observing that unlike the DMG and SGB, the CGB cpu uses dynamic logic and loses its state when not clocked for a few seconds. He HALTED the cpu clock before the write, and in addition dropped the 3.3v line down to near 0v (to help randomize the internal register contents). This caused both the disable write to fail to properly occur, and the CPU's program counter and other registers to be filled with random values. After doing this several times, the program counter ended up pointing into external cartridge rom space, which contained a long chain of NOPS and a dumping program. The dumping program wrote the bootrom out byte by byte to the FPGA (using a bogus cartridge-address-space address which the FPGA recognized). The rom dump includes the 256 byte rom (0x0000-0x00FF) and the 1792 byte rom (0x0200-0x08FF) which Dr. Decapitator observed, but not the 512 byte rom, which may be cpu microcode or lcd color lookup related.

The 'Pokemon' CGB bootstrap

An interesting 'prototype' or alternate version of the CGB bootrom can be found included in the "Pokemon Stadium" N64 cartridge rom. This might possibly have been a leftover from an earlier prototype "Pokemon Stadium" cartridge which actually had a variant CGB CPU on it which would retrieve its rom from the n64 rom. The final n64 cartridge does not have a CGB CPU on it, but it does emulate the CGB hardware using N64 software, but is locked to only running the pokemon CGB games, which are copied, ram and rom, out of the cart on startup. The pokemon stadium 'emulator' code probably does use the bootstrap when starting up.

Impact

Apart from amazement, the dumping of the DMG bootrom led to the inclusion of a feature to emulate the bootstrap ROM in the emulators KiGB and BGB. The dumping of the SGB bootrom led to the inclusion of support for it in the MESS emulator.

Other findings

As a result of the process, neviksti also published pictures of the rest of the chip. All material published in conjunction with the hack can be found here: [1] (http://www.neviksti.com/DMG/)

Patents

The following invention is claimed for the bootstrap:

- US Patent #5,134,391 (http://www.google.com/search?q=patent+%235134391) - System for preventing the use of an unauthorized external memory

Contents of the ROM

Below is the disassembled code of the bootstrap ROM, together with Neviksti's comments. A binary file of the 256 byte area can be downloaded here: [2] (http://www.neviksti.com/DMG/DMG_ROM.bin). The disassembled ROM file can also be found here: [3] (http://www.neviksti.com/DMG/DMG_ROM.asm).

```
        LD SP,$fffe             ; $0000  Setup Stack

        XOR A                   ; $0003  Zero the memory from $8000-$9FFF (VRAM)
        LD HL,$9fff             ; $0004
Addr_0007:
        LD (HL-),A              ; $0007
        BIT 7,H         ; $0008
        JR NZ, Addr_0007        ; $000a

        LD HL,$ff26             ; $000c  Setup Audio
        LD C,$11                ; $000f
        LD A,$80                ; $0011
        LD (HL-),A              ; $0013
        LD ($FF00+C),A  ; $0014
        INC C                   ; $0015
        LD A,$f3                ; $0016
        LD ($FF00+C),A  ; $0018
        LD (HL-),A              ; $0019
        LD A,$77                ; $001a
        LD (HL),A               ; $001c

        LD A,$fc                ; $001d  Setup BG palette
        LD ($FF00+$47),A        ; $001f

        LD DE,$0104             ; $0021  Convert and load logo data from cart into Video RAM
        LD HL,$8010             ; $0024
Addr_0027:
        LD A,(DE)               ; $0027
        CALL $0095              ; $0028
        CALL $0096              ; $002b
        INC DE          ; $002e
        LD A,E          ; $002f
        CP $34          ; $0030
        JR NZ, Addr_0027        ; $0032

        LD DE,$00d8             ; $0034  Load 8 additional bytes into Video RAM (the tile for ®)
        LD B,$08                ; $0037
Addr_0039:
        LD A,(DE)               ; $0039
        INC DE          ; $003a
        LD (HL+),A              ; $003b
        INC HL          ; $003c
        DEC B                   ; $003d
        JR NZ, Addr_0039        ; $003e

        LD A,$19                ; $0040  Setup background tilemap
        LD ($9910),A    ; $0042
        LD HL,$992f             ; $0045
```

```
Addr_0048:
        LD C,$0c                ; $0048
Addr_004A:
        DEC A                   ; $004a
        JR Z, Addr_0055 ; $004b
        LD (HL-),A              ; $004d
        DEC C                   ; $004e
        JR NZ, Addr_004A        ; $004f
        LD L,$0f                ; $0051
        JR Addr_0048    ; $0053

        ; === Scroll logo on screen, and play logo sound===

Addr_0055:
        LD H,A          ; $0055  Initialize scroll count, H=0
        LD A,$64                ; $0056
        LD D,A          ; $0058  set loop count, D=$64
        LD ($FF00+$42),A        ; $0059  Set vertical scroll register
        LD A,$91                ; $005b
        LD ($FF00+$40),A        ; $005d  Turn on LCD, showing Background
        INC B                   ; $005f  Set B=1
Addr_0060:
        LD E,$02                ; $0060
Addr_0062:
        LD C,$0c                ; $0062
Addr_0064:
        LD A,($FF00+$44)        ; $0064  wait for screen frame
        CP $90          ; $0066
        JR NZ, Addr_0064        ; $0068
        DEC C                   ; $006a
        JR NZ, Addr_0064        ; $006b
        DEC E                   ; $006d
        JR NZ, Addr_0062        ; $006e

        LD C,$13                ; $0070
        INC H                   ; $0072  increment scroll count
        LD A,H          ; $0073
        LD E,$83                ; $0074
        CP $62          ; $0076  $62 counts in, play sound #1
        JR Z, Addr_0080 ; $0078
        LD E,$c1                ; $007a
        CP $64          ; $007c
        JR NZ, Addr_0086        ; $007e  $64 counts in, play sound #2
Addr_0080:
        LD A,E          ; $0080  play sound
        LD ($FF00+C),A  ; $0081
        INC C                   ; $0082
        LD A,$87                ; $0083
        LD ($FF00+C),A  ; $0085
Addr_0086:
        LD A,($FF00+$42)        ; $0086
        SUB B                   ; $0088
        LD ($FF00+$42),A        ; $0089  scroll logo up if B=1
        DEC D                   ; $008b
        JR NZ, Addr_0060        ; $008c

        DEC B                   ; $008e  set B=0 first time
        JR NZ, Addr_00E0        ; $008f    ... next time, cause jump to "Nintendo Logo check"

        LD D,$20                ; $0091  use scrolling loop to pause
        JR Addr_0060    ; $0093

        ; ==== Graphic routine ====

        LD C,A          ; $0095  "Double up" all the bits of the graphics data
        LD B,$04                ; $0096     and store in Video RAM
Addr_0098:
        PUSH BC         ; $0098
        RL C                    ; $0099
        RLA                     ; $009b
        POP BC          ; $009c
        RL C                    ; $009d
        RLA                     ; $009f
        DEC B                   ; $00a0
        JR NZ, Addr_0098        ; $00a1
        LD (HL+),A              ; $00a3
        INC HL          ; $00a4
        LD (HL+),A              ; $00a5
        INC HL          ; $00a6
        RET                     ; $00a7
```

```
Addr_00A8:
        ;Nintendo Logo
        .DB $CE,$ED,$66,$66,$CC,$0D,$00,$0B,$03,$73,$00,$83,$00,$0C,$00,$0D
        .DB $00,$08,$11,$1F,$88,$89,$00,$0E,$DC,$CC,$6E,$E6,$DD,$DD,$D9,$99
        .DB $BB,$BB,$67,$63,$6E,$0E,$EC,$CC,$DD,$DC,$99,$9F,$BB,$B9,$33,$3E

Addr_00D8:
        ;More video data (the tile data for ®)
        .DB $3C,$42,$B9,$A5,$B9,$A5,$42,$3C

        ; ===== Nintendo logo comparison routine =====

Addr_00E0:
        LD HL,$0104             ; $00e0 ; point HL to Nintendo logo in cart
        LD DE,$00a8             ; $00e3 ; point DE to Nintendo logo in DMG rom

Addr_00E6:
        LD A,(DE)               ; $00e6
        INC DE          ; $00e7
        CP (HL)         ; $00e8 ;compare logo data in cart to DMG rom
        JR NZ,$fe               ; $00e9 ;if not a match, lock up here
        INC HL          ; $00eb
        LD A,L          ; $00ec
        CP $34          ; $00ed ;do this for $30 bytes
        JR NZ, Addr_00E6        ; $00ef

        LD B,$19                ; $00f1
        LD A,B          ; $00f3
Addr_00F4:
        ADD (HL)                ; $00f4
        INC HL          ; $00f5
        DEC B                   ; $00f6
        JR NZ, Addr_00F4        ; $00f7
        ADD (HL)                ; $00f9
        JR NZ,$fe               ; $00fa ; if $19 + bytes from $0134-$014D  don't add to $00
                                        ;  ... lock up

        LD A,$01                ; $00fc
        LD ($FF00+$50),A        ; $00fe ;turn off DMG rom
```

References

Mirror of the original Cherryroms thread on archive.org (http://web.archive.org/web/20060507053755/http://forums.cherryroms.com/viewtopic.php?t=3848&start=75)

Retrieved from "https://gbdev.gg8.se/wiki/index.php?title=Gameboy_Bootstrap_ROM&oldid=874"

- This page was last modified on 11 May 2019, at 12:35.