SF30th Hacking Edition : A journey into Moo
Table of Contents

1 Introduction

2 From Moo to Arcade

3 Play additional games

4 Netcode
About me

Pau Oliva Fora - @pof

**Position:** Senior Security Consultant

**Company:** IOActive

**Description:** I enjoy a diverse and challenging role performing penetration testing, reverse engineering and vulnerability discovery.

I only play (and care about) Super Street Fighter 2X.
Nico

Position: Reverse Engineer

Company: Synacktiv

Description: French offensive security company
3 teams: pentest, reverse engineering, development

RE team: focus on low level dev, reverse, vulnerability research / exploitation
SF30th Anniversary Collection

- released in May 2018 on every modern platform
- developed by Digital Eclipse and edited by Capcom
12 Street Fighter games playable offline
4 Street Fighter games playable online
Content

- Training mode
- Museum
- everything great but... only 12 games to play offline and fewer games with online mode :'(
Moo Emulator

- Digital Eclipse uses a custom emulator called “Moo” in some of their games:
  - SF30th Anniversary Collection
  - SNK 40th Anniversary Collection
  - Samurai Shodown Collection (not released yet)
- Arcade emulator written from scratch, proprietary
Goal

**MOAR Games**
- Instrumentalize the emulator in order to load additional games

**Netcode**
- Fix SSF2X turbo speed
- Play different games and enjoy Capcom netcode (:}
Why?

- Because “Moo” is really a great emulator and some games run better than in any other emulators (2x, 3.3, etc.)
- An online mode is provided natively and works smoothly.

---

SabreAZ 0 points · 1 year ago
30th anniversary has some of the strongest netcode I have ever seen. It was the fact that before filters, you were getting paired up with people from all around the world, regardless of connection quality. I can play people all over the US and Canada on delay zero, and get clean games. I don't know of any netcode that has such a high threshold. Not even GGPO can do this at zero delay settings
Moo ??

By looking at the classes names extracted from the RTTI information, the symbol “Moo” appears.
Let’s google it

- If we google it, there is only one accurate occurrence, a guy that talk about Moo in Arcade1up reddit

**MOO emulator and stock controls**: Arcade1Up - Reddit
https://www.reddit.com/r/comments/moo_emulator_a...

- **EMULATORS**: there are **two three four** different emulations systems in use in - **MAME**, the "MOO" commercial emulator, RetroArch + Libretro + FBA (for Gauntlet on Rampage v1.0.1, MAME for the rest), RetroArch + Libretro + MAME2003 (for Gauntlet on Rampage v1.0.4 & v1.0.5, MAME for the rest):
  - **MAME** is v0.139u1 on the 12-in-1, Centipede, Asteroids, & Rampage cabinets. Each cabinet has its own compiled build of MAME, configured to understand that cabinet's control panel layout. (See controls in the pin-out "spreadsheet" above.) If you **add a USB port**, a PC keyboard allows access all the standard MAME options - **including spinner sensitivity**. USB mouse functions as a trackball (at least on 12-in-1)
  - **MOO** is a commercial emulator, and appears to be built/licensed per cabinet; it's hard-coded to support **only a specific small number of ROMs** for the games that are in the cabinet. Used in SF2, Galaga, PacMan, and Space Invaders. All future cabinets are very likely to use this same emulator.
  - **RetroArch + Libretro + MAME 2003** I have near-zero familiarity with. Who can give me a quick rundown on how it works, how it differs from MAME, how it's similar, etc?
  - **RetroArch + FinalBurn Alpha** I have near-zero familiarity with. Who can give me a quick rundown on how it works, how it differs from MAME, how it's similar, etc?
Arcade1Up: Cheap Arcade Cabinet ($250)
Arcade1up PCB

STREET FIGHTER PCB

LA-815221026582

$29.99 $39.99

Arcade1Up Printed Circuit Board is the replacement brain for your cabinet.

Easily installed
nico@debian ~/WIP/r2con % file MOO-Capcom-ShipMusl-SF
MOO-Capcom-ShipMusl-SF: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter /lib/ld-musl-armhf.so.1, stripped

ROM strings

The Arcade1up cabinets use the “Moo” emulator

0x000ca364 MOO Emulation Copyright Daniel Filner (moo@tilekiller.com)
0x000caa10 Moo_Audio_Base
0x000caad4 MooUnZipImp::GetFileAsBuffer(%s) couldn't be matched
0x000cab0c MooUnZipImp[%p]::GoToFileByIndex(%d) invalid index
0x000cbf40 Moo_Audio_YM2151
0x000cbf54 Moo_Audio_OKIM6295
0x000e1d70 Moo_SYS_CPS2::ResetNVState() finds no Default Contents
0x000e1e80 Moo_CPU_68000
0x000e1e90 Moo_Audio_QSound
0x000e40f0 Moo_CPU_Z80
Moo author

- Experimented developer (+30 years of experience)

**Daniel Filner**

<table>
<thead>
<tr>
<th>Also Known As</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan Filner</td>
<td></td>
</tr>
</tbody>
</table>

**Game Credits**

<table>
<thead>
<tr>
<th>Programming/Engineering</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Street Fighter: 30th Anniversary Collection (2018)</td>
<td>Emulation Engineer</td>
</tr>
<tr>
<td>Yu-Gi-Oh! Legacy of the Duelist (2015)</td>
<td>Lead Engineer</td>
</tr>
<tr>
<td>Midway Arcade Origins (2012)</td>
<td>Lead Engineer</td>
</tr>
<tr>
<td>The Simpsons (2012)</td>
<td>Lead Engineer</td>
</tr>
<tr>
<td>Sonic: Generations (2011)</td>
<td>Lead Engineer</td>
</tr>
<tr>
<td>Build It Green: Back to the Beach (2010)</td>
<td>Playground Game Engine</td>
</tr>
</tbody>
</table>

**Related Sites**

- LinkedIn -- professional profile

[add portrait]
Table of Contents

1 Introduction

2 From Moo to Arcade

3 Play additional games

4 Netcode
Moo -> Mame -> EEPROM -> Arcade
Agenda

1. background
2. motivation
3. demotivation
4. results
Capcom (Digital Eclipse?) promised *arcade perfect* game play

**BUT**

fixing any glitch or bug causing the games to freeze or reset.
Freeze: Old Honda Throw

https://www.youtube.com/watch?v=06xuJSVJXeE
Reset: Sagat Tiger Knee vs CPU Gief

https://www.youtube.com/watch?v=_vPj8fwCLb4
ETC: 940223
JAPAN: 940223
30th: 940323
Motivation

Possible undumped rom!!? :)

Write a quick & dirty PoC to extract it!
#!/bin/bash

sf30th="./30th/
out="./out"
cps2key="0x94fa8002 0x4c77143F"

# audiocpu - sfx.01 & sfx.02
echo "Extracting audiocpu - sfx.01 & sfx.02"
dd if=$sf30th/SuperStreetFighterIIITurbo.Z80 of=$out/sfx.01 bs=131072 count=1 skip=0
dd if=$sf30th/SuperStreetFighterIIITurbo.Z80 of=$out/sfx.02 bs=131072 count=1 skip=1

# qsound - sfx.11m sfx.12m
echo "Extracting qsound - sfx.11m sfx.12m"
dd conv=swab <$sf30th/SuperStreetFighterIIITurbo.qs > /tmp/SuperStreetFighterIIITurbo.qs.$$
dd if=/tmp/SuperStreetFighterIIITurbo.qs.$$ of=$out/sfx.11m bs=2097152 count=1 skip=0
dd if=/tmp/SuperStreetFighterIIITurbo.qs.$$ of=$out/sfx.12m bs=2097152 count=1 skip=1
rm /tmp/SuperStreetFighterIIITurbo.qs.$$

# maincpu
echo "Extracting maincpu 

dd if=$sf30th/SuperStreetFighterIIITurbo.u.68y of=$out/d_sfxu.03e bs=0x80000 count=1 skip=0
rahash2 -E cps2 -S "$cps2key" "$out"/d_sfxu.03e > "$out"/sfxu.03e
dd if=$sf30th/SuperStreetFighterIIITurbo.u.68y of=$out/d_sfxu.04a bs=0x80000 count=1 skip=1
rahash2 -E cps2 -S "$cps2key" "$out"/d_sfxu.04a > "$out"/sfxu.04a
dd if=$sf30th/SuperStreetFighterIIITurbo.u.68y of=$out/d_sfxu.05 bs=0x80000 count=1 skip=2
rahash2 -E cps2 -S "$cps2key" "$out"/d_sfxu.05 > "$out"/sfxu.05
dd if=$sf30th/SuperStreetFighterIIITurbo.u.68y of=$out/d_sfxu.06b bs=0x80000 count=1 skip=3
rahash2 -E cps2 -S "$cps2key" "$out"/d_sfxu.06b > "$out"/sfxu.06b
dd if=$sf30th/SuperStreetFighterIIITurbo.u.68y of=$out/d_sfxu.07a bs=0x80000 count=1 skip=4
rahash2 -E cps2 -S "$cps2key" "$out"/d_sfxu.07a > "$out"/sfxu.07a
dd if=$sf30th/SuperStreetFighterIIITurbo.u.68y of=$out/d_sfxu.08 bs=0x80000 count=1 skip=5
rahash2 -E cps2 -S "$cps2key" "$out"/d_sfxu.08 > "$out"/sfxu.08
dd if=$sf30th/SuperStreetFighterIIITurbo.u.68y of=$out/d_sfxu.09 bs=0x80000 count=1 skip=6
rahash2 -E cps2 -S "$cps2key" "$out"/d_sfxu.09 > "$out"/sfxu.09

# gfx - sfx.13m sfx.15m sfx.17m sfx.19m sfx.14m sfx.16m sfx.18m sfx.20m sfx.21m sfx.23m sfx.25m sfx.27m
"extract.sh" 67L, 4669C
Demotivation 1

- Converting CPS2 Graphics from MAME <-- MOO was difficult
Demotivation 1

- Converting CPS2 Graphics from MAME <-> MOO was difficult
Demotivation 1

- Converting CPS2 Graphics from MAME \(\text{\textlangle}\text{--\textrangle}\) MOO was difficult
Demotivation 2

- The rom was already dumped :")(\n
ssf2t, "Super Street Fighter II Turbo (World 940223)"
ssf2ta, "Super Street Fighter II Turbo (Asia 940223)"
ssf2th, "Super Street Fighter II Turbo (Hispanic 940223)"

ssf2tu, "Super Street Fighter II Turbo (USA 940323)"

ssf2tur1, "Super Street Fighter II Turbo (USA 940223)"
ssf2xj, "Super Street Fighter II X: Grand Master Challenge (Japan 940311)"
ssf2xjr1, "Super Street Fighter II X: Grand Master Challenge (Japan 940223)"
ssf2xjr1r, "Super Street Fighter II X: Grand Master Challenge (Japan 940223 rent version)"
Changes
Results

Burn EEPROMs and play the game on real hardware
Donor B-board
CPS2 Crypto

Undamaged CPS2 InfiniKey used to inject the game's key on the B board
EEPROMS

- 27C1001
- 27C4096
- 27C322
Burn EEPROMs
oops...
TODO

Graphics conversion is a PITA
# Table of Contents

1. Introduction
2. From Moo to Arcade
3. Play additional games
4. Netcode
Workflow

Workflow when loading a game

- Init the Game object according to the chosen game. For S2HF, the following object is initialized:
  - `Game_StreetFighterII_HF : Moo_Sys_CPS1 : MooBase`
- Parse and retrieve game assets from the filesystem
- Map the GFXs using bank mappers
- Render graphics, run the 68k emulator with the maincpu rom
Game assets

The game assets are located into a kind of ordered dictionary files.

These resources are neither compressed nor encrypted.

Loïc WydD Petit wrote a script to extract these assets:

https://github.com/WydD/sf30ac-extractor
Game assets

Roms

- By extracting the game assets, we can get the roms data.
- SF30th emulator do not support Mame roms, it works only with plain rom.
Street Fighter II Hyper Fighting roms

MOO
- StreetFighterII_HF.oki
- StreetFighterII_HF.u.68k
- StreetFighterII_HF.u.vrom
- StreetFighterII_HF.z80

MAME
- s2tu_21.6f
- s2tu_22.7f
- s2tu_23.8f
- s92-1m.3a
- s92-2m.4a
- s92-3m.5a
- s92-4m.6a
- s92-5m.7a
- s92-6m.8a
- s92-7m.9a
- s92-8m.10a
- s92-10m.3c
- s92-11m.4c
- s92-12m.5c
- s92-13m.6c
- s92_18.11c
- s92_19.12c
- s92_09.11a

68000 code

GFX

AUDIO SAMPLES

AUDIOPC CPU
the differences are easily visible: swap each word (2 bytes)
Mame CPS1 driver source code

```c
/* B-Board 91635B-2 */
ROM_START( sf2hfu )
    ROM_REGION( CODE_SIZE, "maincpu", 0 )       /* 60000 code */
    ROM_LOAD16_WORD_SWAP( "s2tu_23.8f", 0x000000, 0x80000, CRC89a1fc38) SHAl(aa6b40fc311e31825973be8c6aa8d37f7902cb3c)
    ROM_LOAD16_WORD_SWAP( "s2tu_22.7f", 0x000000, 0x80000, CRC(acea8e035) SHAl(ce5fe961b21c95d231d1235bfc83b47de489f2a)
    ROM_LOAD16_WORD_SWAP( "s2tu_21.6f", 0x100000, 0x80000, CRC(fd200288) SHAl(3817b67ab77c7b3d4a573a53f18671bea6905e26) )
```

**Figure 1** – https://github.com/fesh0r/old-mame/blob/master/src/mame/drivers/cps1.c#L9618
Audio samples (oki files)

- Just a concatenation of the oki files (for the order: check mame CPS1 driver source code)
Audio CPU (z80 file)

- Identical

```
shasum StreetFighterII_HF.z80 s92_09.11a
8258fcaca4ac419312531ee67070b97f471179c StreetFighterII_HF.z80
8258fcaca4ac419312531ee67070b97f471179c s92_09.11a
```
GFX files

VROM?
- The VROM is a ROM chip inside the game board, it contains:
  - pixel patterns, the colors and the metadata for assembling the tiles into the background and sprites

Conversion
- Convert gfx from Mame to Moo:
  - merge each files into one and reorder bytes
  - decode gfx data
GFX files [1/2]
GFX files [2/2]

```c
void cps_state::cps1_gfx_decode()
{
    int size = memregion("gfx")->bytes();
    int i, j, gfxsize;
    UINT8 *cps1_gfx = memregion("gfx")->base();

    gfxsize = size / 4;

    for (i = 0; i < gfxsize; i++)
    {
        UINT32 src = cps1_gfx[4 * i] + (cps1_gfx[4 * i + 1] << 0) + (cps1_gfx[4 * i + 2] << 10) + (cps1_gfx[4 * i + 3] << 16);
        UINT32 dval = 0;

        for (j = 0; j < 8; j++)
        {
            int n = 0;
            UINT32 mask = (0x00000000 >> j) & src;

            if (mask & 0x000000ff) n |= 1;
            if (mask & 0x0000ff00) n |= 2;
            if (mask & 0x00ff0000) n |= 4;
            if (mask & 0xff000000) n |= 8;

            dval |= n << (j * 4);
        }
        cps1_gfx[4 * i] = dval >> 0;
        cps1_gfx[4 * i + 1] = dval >> 8;
        cps1_gfx[4 * i + 2] = dval >> 16;
        cps1_gfx[4 * i + 3] = dval >> 24;
    }
}
```

Figure 2 – https://github.com/fesh0r/old-mame/blob/master/src/mame/video/cps1.c#L1720
Mame to Moo conversion

- The Mame driver source code must be parsed
  - to know what are the audio / gfx / 68k files
  - to get the correct order when concatenating the oki files
  - to know how to reorder the 68k files
Mame to Moo conversion

Figure 3 –
https://github.com/angelkillah/MooHijack/blob/master/script/mame2moo.py
original GFX patched in sf30th
Hijack Moo roms loading

Now that we can convert any CPS1 roms from Mame to Moo, we need to force the game to load our freshly converted roms.

**Steps**

- Locate the assets loading function
- Hijack the execution flow
Moo assets loading
Moo assets loading

```assembly
lea rdx, str.size ; 0x14026c684 ; "size"
mov rcx, rbx
call GetField
mov r14d, eax
lea rdx, str.offset ; 0x14026c68c ; "offset"
mov rcx, rbx
call GetField
mov ebx, eax
mov ecx, r14d
call qword sym.imp.api_ms_win_crt_heap_11_1_0.dll_malloc ; 0x14023c5d0 ; "$\114" ; void *malloc...
mov r15, rax
mov r8d, ebx
mov rdx, rsi
mov rcx, rbp
call MBundle_fseek
mov r9d, r14d ; size
mov r8, r15 ; data
mov rdx, rsi ; MFString
mov rcx, rbp ; MFileSystem
call GetData
mov rdx, rsi
mov rcx, rbp
call fcn.140091180
lea ecx, [rdi + 0x28] ; '(' ; 40 ; size
call fcn.new
mov qword [var_80h], rax
test rax, rax
je 0x1401482b0
```
Hooking

To replace the loading of whatever resource, we hijack the execution flow at two different locations:
- the result of the first call to GetField() : to replace the original resource size
- the buffer filled by GetData() : to replace the resource data
Hooking

**VEH Hooking**

- We modify the first byte of the instruction to hijack by an opcode that will cause an exception
- We install a vectored exception handler to catch it
- cons: no need to calculate the instructions size
Hijack assets loading

```
le   sf30thanniversarycollection.7FF7CAF582B0  
lea  rdx,qword ptr ds:[7FF7CB07C684]  
mov  rcx,rbx  
call <sf30thanniversarycollection.GetField>  
    int3  
mov  esi,eax  
lea  rdx,qword ptr ds:[7FF7CB07C680]  
mov  rcx,rbx  
call <sf30thanniversarycollection.GetField>  
mov  ebx,eax  
mov  ecx,r14d  
call qword ptr ds:[<malloc>]  
mov  r15,rax  
mov  r8d,ebx  
mov  rdx,rsi  
mov  rcx,rbp  
call sf30thanniversarycollection.7FF7CAE91E00  
mov  r9d,r14d  
mov  r8,r15  
mov  rdx,rsi  
mov  rcx,rbp  
call <sf30thanniversarycollection.GetData>  
mov  rdx,rsi  
    int3  
mov  ecx,ebp
```

**VEH Handler**

- Set rax to the size of the resource we want to load.
- R10-size always points to the resource data, we patch the resource content with our resource.
Results...
Results...
Game over?

HANDSOME FIGHTERS NEVER LOSE BATTLES.
Almost no differences between both functions

Setup_CPS1_With_ROM_info() takes two arguments:

- the object of the chosen game (this)
- an address to a structure...
SF2CE_Config

.qword 0x000000001402de240 ; str.Street_Fighter_II:_Champion_Edition
.qword 0x000000001402de268 ; str.920313_USA
.qword 0x000000001402de278 ; str.StreetFighterII_CE
.qword 0x000000001402de290 ; str.Capcom_StreeetFighterII_CE
.qword 0x000000001402bc3e8
.qword 0x000000001402de2b0 ; str.StreetFighterII_CE.ua.68k
.qword 0x000000001402de2d0 ; str.StreetFighterII_CE.vrom
.qword 0x000000001402de2f0
.qword 0x000000001402ddf30 ; str.eagle_logo.vrom
.qword 0x000000001402de300 ; str.StreetFighterII_CE.oki
1402bc3e8 => Dipswitches
Test Mode
1402bd090 => ??

```
.qword 0x0000000000000002
.dword 0x00571b00
.dword 0x0000005b
.dword 0x00000032
.dword 0x00000001
.dword 0x00000026
.dword 0x00000006
.dword 0x0000003e
.dword 0x00000002
.dword 0x00000028
.dword 0x00000003
.dword 0x0000002a
.dword 0x00000004
.dword 0x0000002c
.dword 0x00000005
.dword 0x0000002a
.dword 0x00000007
.dword 0x00000000
.dword 0x00000008
.dword 0x00000002
.dword 0x00000009
.dword 0x00000004
.dword 0x0000000a
.dword 0x00000006
.dword 0x0000000c
.dword 0x00000006
.dword 0x0000000c
.dword 0x0000000f
.qword 0x0000000000000000
.qword 0x0000000000000000
.dword 0x00000000
.dword 0x00000000
.dword 0x00000000
.dword 0x00000000
.dword 0x00000000
.dword 0x00000000
.dword 0x00000000
.dword 0x00000000
.qword 0x0000000000000000
```

SYNACKTIV

69
A set of these data are copied to the “StreetFighterII_CE” object attributes.

The first 4-bytes data value (0xb71b00) is used in a method of the class “Moo_Sys_CPS1”.

This method is used to execute the 68000 code ROM through an emulator.
A set of these data are copied to the “StreetFighterII_CE” object attributes.
The first 4-bytes data value (0xb71b00) is used in a method of the class “Moo_Sys_CPS1”.
This method is used to execute the 68000 code ROM through an emulator.

Clock frequency
- 0xb71b00 == 12000000 = 12Mhz
- The processor Motorola 68000 used for SF2CE runs at 12Mhz
CPS-B Registers

- The original arcade board of CPS1 games contains several registers:
  - priority mask: used to set the tiles priority levels
  - palette control register: indicates which palette pages to copy from gfxram to dedicated ram
  - test register: used for self test checks
  - etc.
CPS-B Registers

Figure 4 –

Luckily, the values to set in CPS-B registers for each game are listed in mame cps1 video source code
CPS-B Registers for SF2CE

{"sf2ceua", CPS_B_21_DEF, mapper_S9263B, 0x36 }

Figure 5 – name, CPSB, gfx mapper, in2
CPSB-21-DEF

/*
   CPSB ID            multiply protection unknown        ctrl    priority masks palctrl    layer enable masks */
#define CPS_B 01   -1, 0x0000,             __not_applicable__,   0x26,{0x28,0x2a,0x2c,0x2e},0x30, {0x02,0x04,0x06,0x30,0x30}
#define CPS_B 02   0x20,0x0002,             __not_applicable__,   0x2c,{0x2a,0x28,0x26,0x24},0x22, {0x02,0x04,0x06,0x00,0x00}
#define CPS_B 03   -1, 0x0000,             __not_applicable__,   0x26,{0x2e,0x2a,0x2a,0x28},0x26, {0x28,0x10,0x08,0x00,0x00}
#define CPS_B 04   0x20,0x0004,             __not_applicable__,   0x2e,{0x26,0x30,0x28,0x32},0x2a, {0x02,0x04,0x08,0x00,0x00}
#define CPS_B 05   0x20,0x0005,             __not_applicable__,   0x28,{0x2a,0x2c,0x2e,0x30},0x32, {0x02,0x08,0x26,0x14,0x14}
#define CPS_B 11   0x32,0x0401,             __not_applicable__,   0x26,{0x28,0x2a,0x2c,0x2e},0x30, {0x08,0x10,0x26,0x00,0x00}
#define CPS_B 12   0x26,0x0402,             __not_applicable__,   0x2c,{0x2a,0x28,0x26,0x24},0x22, {0x02,0x04,0x08,0x00,0x00}
#define CPS_B 13   0x2e,0x0403,             __not_applicable__,   0x22,{0x24,0x26,0x28,0x2a},0x2c, {0x20,0x02,0x04,0x00,0x00}
#define CPS_B 14   0x1e,0x0404,             __not_applicable__,   0x12,{0x14,0x16,0x18,0x1a},0x1c, {0x08,0x20,0x10,0x00,0x00}
#define CPS_B 15   0x0e,0x0405,             __not_applicable__,   0x02,{0x04,0x06,0x08,0x0a},0x0c, {0x04,0x02,0x26,0x00,0x00}
#define CPS_B 16   0x00,0x0406,             __not_applicable__,   0x0c,{0x0a,0x08,0x06,0x04},0x02, {0x10,0x0a,0x0a,0x00,0x00}
#define CPS_B 17   0x08,0x0407,             __not_applicable__,   0x14,{0x12,0x10,0x0e,0x0c},0x0a, {0x08,0x14,0x02,0x00,0x00}
conversion needs 0x04 for the 2nd layer enable on one level, gfx confirmed to appear on the PCB, register at the time is 0x8e, so 0
#define CPS_B 18   0x10,0x0408,             __not_applicable__,   0x1c,{0x1a,0x18,0x16,0x14},0x12, {0x10,0x08,0x02,0x00,0x00}
#define CPS_B 21 DEF 0x32,  -1, 0x00,0x02,0x04,0x06,  0x08,  -1,  -1, 0x26,{0x28,0x2a,0x2c,0x2e},0x30, {0x02,0x04,0x08,0x30,0x30}*/
Makes more sense!
GFX Mapper
GFX Mapper

```c
#define mapper_S9263B  { 0x8000, 0x8000, 0x8000, 0 }, mapper_S9263B_table
static const struct gfx_range mapper_S9263B_table[] =
{
    // verified from PAL dump:
    // FIXME there is some problem with this dump since pin 14 is never enabled
    // instead of being the same as pin 15 as expected
    // bank0 = pin 19 (ROMs 1,3) & pin 1B (ROMs 2,4)
    // bank1 = pin 17 (ROMs 5,7) & pin 16 (ROMs 6,8)
    // bank2 = pin 15 (ROMs 10,12) & pin 14 (ROMs 11,13)
    // pins 12 and 13 are the same as 14 and 15

    /* type    start    end    bank */
    { GFXTYPE_SPRITES, 0x00000, 0x07fff, 0 },

    { GFXTYPE_SPRITES, 0x08000, 0x0fff, 1 },

    { GFXTYPE_SPRITES, 0x10000, 0x11fff, 2 },
    { GFXTYPE_SCROLL3, 0x02000, 0x03fff, 2 },
    { GFXTYPE_SCROLL1, 0x04000, 0x04fff, 2 },
    { GFXTYPE_SCROLL2, 0x05000, 0x07fff, 2 },
    { 0 }
};
```

Figure 6 –
GFX Mapper

/* type      start   end    bank */
{ GFXTYPE_SPRITES, 0x000000, 0x7fff, 0 },

{ GFXTYPE_SPRITES, 0x08000, 0x0fff, 1 },
{ GFXTYPE_SPRITES, 0x10000, 0x11fff, 2 },
{ GFXTYPE_SCROLL3, 0x02000, 0x03fff, 2 },
{ GFXTYPE_SCROLL1, 0x04000, 0x04fff, 2 },
{ GFXTYPE_SCROLL2, 0x05000, 0x07fff, 2 },
{ 0 }

.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff
.dword 0x00000000
.dword 0x00007fff

Summarize

How to load an additional game?
- Convert the rom to Moo compatible one
- Hijack the roms loading with the converted ones
- Patch the CPSB data with the ones from the new game
- Patch the GFX mapper
Demo
I wish I could play Ghouls’n’Ghost :( 

- Some games can be set to freeplay through their dipswitches (no coins needed)
- What about the games that do not have freeplay available?
How to fix in a “generic” way

Patch emulator game memory

- Get the address of the coins through the cheat engine included in Mame debugger
- Hijack the handler of an opcode that is used to read a word value from the game VRAM to set some coins
- Enjoy moar games (:}
Enjoy moar games

Figure 7 – before VRAM patching
Enjoy moar games

Figure 8 – after VRAM patching
How to load an additional game?

- Convert the rom to Moo compatible one
- Hijack the roms loading with the converted ones
- Patch the CPSB data with the ones from the new game
- Patch the GFX mapper
- **Either patch dipswitches to set freeplay game mode or patch game VRAM if freeplay not available**
Table of Contents

1. Introduction
2. From Moo to Arcade
3. Play additional games
4. Netcode
SSF2X speed problem

What’s the problem?

- The online version of ssf2x is not running at the correct speed
- The problem exists since launch day and hasn’t been fixed until now
Workflow when running ssf2x online

- Init the following object:
  - Game_SuperStreetFighterII_Turbo : Moo_Sys_CPS2 : MooBase
- Parse and retrieve game assets from the filesystem
- **Load save state from assets to avoid desynch**
- Map the GFXs using bank mappers
- Render graphics, run the 68k emulator with the maincpu rom
Save state

Save state?

- Moo supports memory save state (emulator snapshot memory)
- When playing offline mode, it is used to save game progress
- For online mode, it is used for both players to start the game at the same state
- For the four games available to play online, there are four saved state files embedded in the mbundle files
Solution: patch and hijack save state

Steps

- RE the save state format and patch the turbo value with the correct one
- Hijack the save state loading with the patched one
- Enjoy
SSF2X online speed FIX
Play a different game online

Netplay

- When reversing the netplay code to fix the ssf2x speed problem, we noticed something interesting . . .
- the roms are loaded LOCALLY for both players !!!!
Enable netplay for MOAR games

How to play additional games online

- Convert the rom to Moo compatible one
- Hijack the roms loading with the converted ones
- Patch the CPSB data with the ones from the new game
- Patch the GFX mapper
- Play the additional game offline, at the menu, select two players and save a memory snapshot
- Take out the new save state from the memory and write it to a file
- Hijack the save state loading with the new one
- Either patch dipswitches to set freeplay game mode or patch game VRAM if freeplay not available
Demo
Source code

https://github.com/angelkillah/MooHijack
QUESTIONS?

Thank you for your attention