INGENIERÍA INVERSA EN LA EMULACIÓN DE VIDEOJUEGOS

MIGUEL ANGEL HORNA

EMULACIÓN DE VIDEOJUEGOS CLÁSICOS

- Hacer funcionar en un hardware actual el programa que corría en un hardware antiguo.
- Sin código fuente
- Sin documentación
- Algunos chips de uso general
- Normalmente hardware gráfico propietario

EMULACIÓN DE VIDEOJUEGOS CLÁSICOS

- Capcom CPS 1 y 2 (Street Fighter 2, Captain Commando, Dungeon & Dragons)
- SNK Neogeo (King of fighters, Fatal Fury, Metal Slug)
- IGS Polygamemaster (Knights of Valour, Daemon Front)
- Capcom System 3 (Street Fighter 3, Jojo's Adventure)
- Sega Model 2 (Daytona USA, Sega Rally, ManxTT Superbike, House of the Dead)

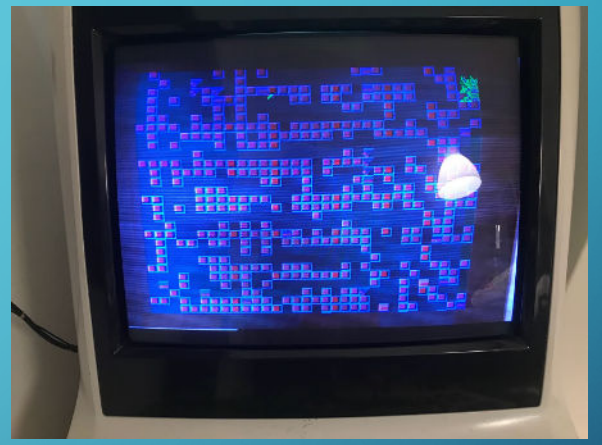
EL FIN DEL TIEMPO

- Fabricada por Niemer
- El eslabón perdido de los videojuegos españoles
- Solo había unos flyers y algún articulo en revistas
- Despues de 35 años
- Una placa apareció en Zaragoza
- ARPA, A.R.C.A.D.E. y Recreativas.org
- Averiada, pero se pudo arreglar

EL FIN DEL TIEMPO

justificar su presencia. Por su parte, Enrique Zarco y José María Arribas, de NIEMER, llevaron al A.T.E. dos vídeos de indudable éxito, como son "EL FIN DEL TIEMPO" y el juego de "LAS 4 EN RAYA", máquinas que tuvieron expuetas en el estand de ELECTROCOIN, firma con la que han establecido interesantes contactos para la venta en el extranjero de estos vídeos, si es que llegan con ella a un acuerdo, pues también se han interesado en ellos las famosas CENTURY y UNIVERSAL. Igualmen-

EL FIN DEL TIEMPO



EL FIN DEL TIEMPO

- Primero, Lectura de las memorias ROM y EEPROM

Nombre	Fecha de modificación	Tipo	Tamaño
12812-2532.bin	26/02/2018 9:55	Kodi	4 KB
12813-2716.bin	26/02/2018 9:55	Kodi	2 KB
12814-2532.bin	26/02/2018 9:55	Kodi	4 KB
12815-2532.bin	26/02/2018 9:55	Kodi	4 KB
12816-2532.bin	26/02/2018 9:55	Kodi	4 KB
12817-2532.bin	26/02/2018 9:55	Kodi	4 KB
12818-2532.bin	26/02/2018 9:55	Kodi	4 KB
12819-2532.bin	26/02/2018 9:55	Kodi	4 KB
12820-2532.bin	26/02/2018 9:55	Kodi	4 KB
12821-2532.bin	26/02/2018 9:55	Kodi	4 KB
12822-2532.bin	26/02/2018 9:55	Kodi	4 KB
22801-2532.bin	26/02/2018 9:55	Kodi	4 KB
22802-2532.bin	26/02/2018 9:55	Kodi	4 KB
22805-2532.bin	26/02/2018 9:55	Kodi	4 KB
22806-2532.bin	26/02/2018 9:55	Kodi	4 KB
22807-2532.bin	26/02/2018 9:55	Kodi	4 KB
22808-2532.bin	26/02/2018 9:55	Kodi	4 KB
FT1-tbp28l22n.bin	26/02/2018 9:55	Kodi	1 KB

- Pero algunas no se dejaban leer -> Viaje a Barcelona y vuelta

EL FIN DEL TIEMPO

- Identificar CPU y otros chips auxiliares

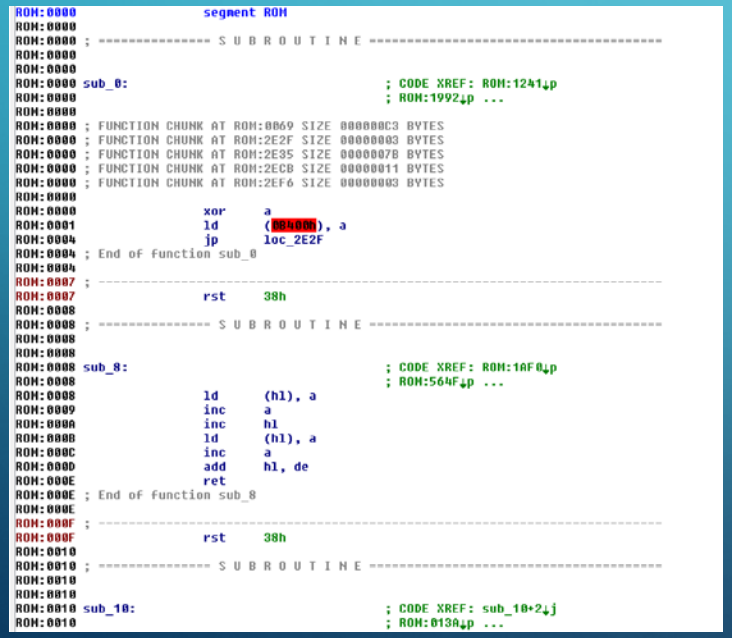
1x Zilog Z8400A PS (Z80A) CPU.
1x Fairchild F6802P CPU.
1x AMD 8255A PPI.
2x GI AY-3-8910.
8x 2114 SRAM (512 bytes).
1x 2516 EPROM (2KB)
17x 2532 EPROMs (4KB)
1x TBP28L22N (256 x 8-bits) bipolar PROM.
1x Xtal @ 14.318 MHz. (used for 6802 CPU & AY8910 sound devices).
1x Xtal @ 18.4320 MHz. (used for Z80 CPU).

EL FIN DEL TIEMPO

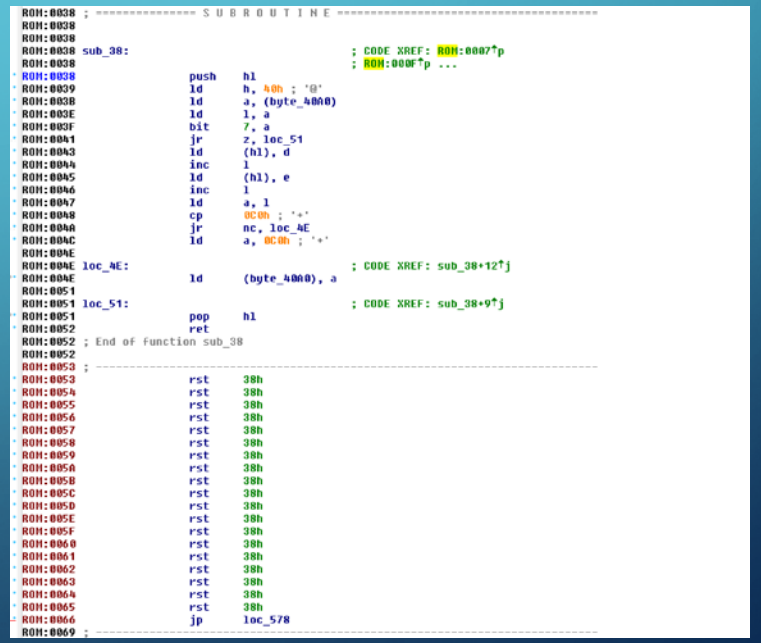
- Tenemos 18 ROMs, ¿dónde esta el código?
- 2 ROMS conectadas al 6802, y cerca de los AY (Sonido)
- No hay ROMs cerca del Z80
- Probar desensamblando ROMS hasta encontrar código válido
- IDA, Ghidra

EL FIN DEL TIEMPO

- Z80
  - Código válido en 00h, 38h y 66h
  - 22805.bin



```
ROM:0000                segment ROM
ROM:0000
ROM:0000 ; ---------------  S U B R O U T I N E  ---------------------------------------
ROM:0000
ROM:0000
ROM:0000 sub_0:                                 ; CODE XREF: ROM:1241↓p
ROM:0000                                        ; ROM:1992↓p ...
ROM:0000
ROM:0000 ; FUNCTION CHUNK AT ROM:0069 SIZE 000000C3 BYTES
ROM:0000 ; FUNCTION CHUNK AT ROM:2E2F SIZE 00000003 BYTES
ROM:0000 ; FUNCTION CHUNK AT ROM:2E35 SIZE 00000078 BYTES
ROM:0000 ; FUNCTION CHUNK AT ROM:2EC8 SIZE 00000011 BYTES
ROM:0000 ; FUNCTION CHUNK AT ROM:2EF6 SIZE 00000003 BYTES
ROM:0000
ROM:0000                xor     a
ROM:0001                ld      (8400h), a
ROM:0004                jp      loc_2E2F
ROM:0004 ; End of function sub_0
ROM:0004
ROM:0007 ; ---------------------------------------------------------------------------
ROM:0007                rst     38h
ROM:0008
ROM:0008 ; ---------------  S U B R O U T I N E  ---------------------------------------
ROM:0008
ROM:0008
ROM:0008 sub_8:                                 ; CODE XREF: ROM:1AF0↓p
ROM:0008                                        ; ROM:564F↓p ...
ROM:0008                ld      (hl), a
ROM:0009                inc     a
ROM:000A                inc     hl
ROM:000B                ld      (hl), a
ROM:000C                inc     a
ROM:000D                add     hl, de
ROM:000E                ret
ROM:000E ; End of function sub_8
ROM:000E
ROM:000F ; ---------------------------------------------------------------------------
ROM:000F                rst     38h
ROM:0010
ROM:0010 ; ---------------  S U B R O U T I N E  ---------------------------------------
ROM:0010
ROM:0010
ROM:0010 sub_10:                                ; CODE XREF: sub_10+2↓j
ROM:0010                                        ; ROM:013A↓p ...
```



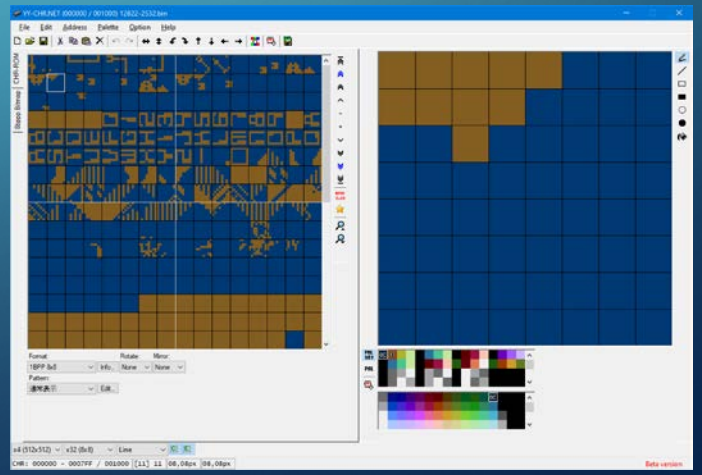
```
ROM:0038 ; ---------------  S U B R O U T I N E  ---------------------------------------
ROM:0038
ROM:0038
ROM:0038 sub_38:                                ; CODE XREF: ROM:0007↑p
ROM:0038                                        ; ROM:000F↑p ...
ROM:0038                push    hl
ROM:0039                ld      h, 40h ; '@'
ROM:003B                ld      a, (byte_40A0)
ROM:003E                ld      l, a
ROM:003F                bit     7, a
ROM:00A1                jr      z, loc_51
ROM:00A3                ld      (hl), d
ROM:00A4                inc     l
ROM:00A5                ld      (hl), e
ROM:00A6                inc     l
ROM:00A7                ld      a, l
ROM:00A8                cp      0C0h ; '+'
ROM:00AA                jr      nc, loc_4E
ROM:00AC                ld      a, 0C0h ; '+'
ROM:00AE
ROM:00AE loc_4E:                                ; CODE XREF: sub_38+12↑j
ROM:00AE                ld      (byte_40A0), a
ROM:0051
ROM:0051 loc_51:                                ; CODE XREF: sub_38+9↑j
ROM:0051                pop     hl
ROM:0052                ret
ROM:0052 ; End of function sub_38
ROM:0052
ROM:0053 ; ---------------------------------------------------------------------------
ROM:0053                rst     38h
ROM:0054                rst     38h
ROM:0055                rst     38h
ROM:0056                rst     38h
ROM:0057                rst     38h
ROM:0058                rst     38h
ROM:0059                rst     38h
ROM:005A                rst     38h
ROM:005B                rst     38h
ROM:005C                rst     38h
ROM:005D                rst     38h
ROM:005E                rst     38h
ROM:005F                rst     38h
ROM:0060                rst     38h
ROM:0061                rst     38h
ROM:0062                rst     38h
ROM:0063                rst     38h
ROM:0064                rst     38h
ROM:0065                rst     38h
ROM:0066                jp      loc_578
ROM:0069 ; ---------------------------------------------------------------------------
```

EL FIN DEL TIEMPO

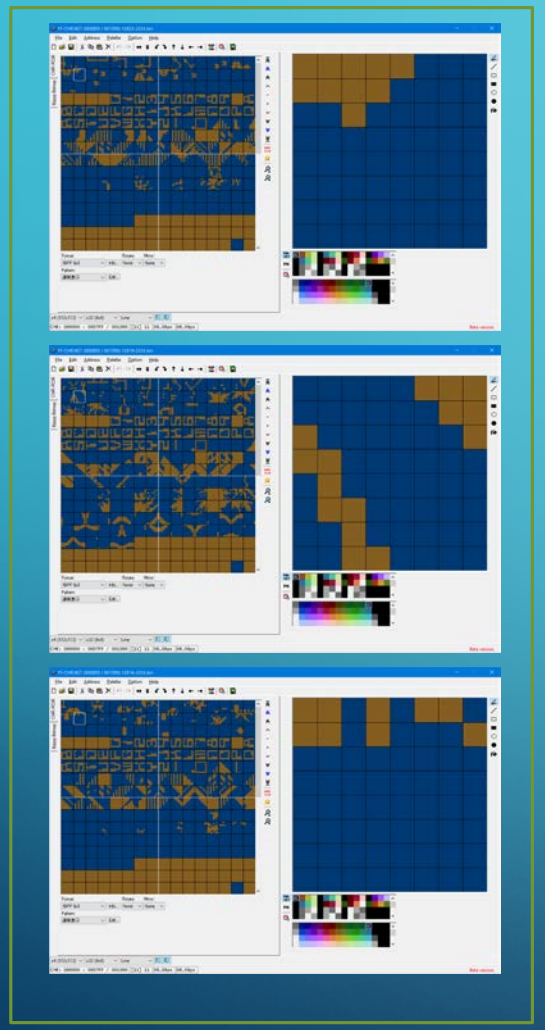
- Tenemos el primer bloque, ¿cual es el siguiente?
- Probamos ROM siguiente y anterior a ver si el código tiene sentido
- Y vamos así con todas las del mismo bloque en la placa
- 6 ROMS de Z80 y 2 roms de 6208
- Preparamos el driver de MAME cargando las ROMS

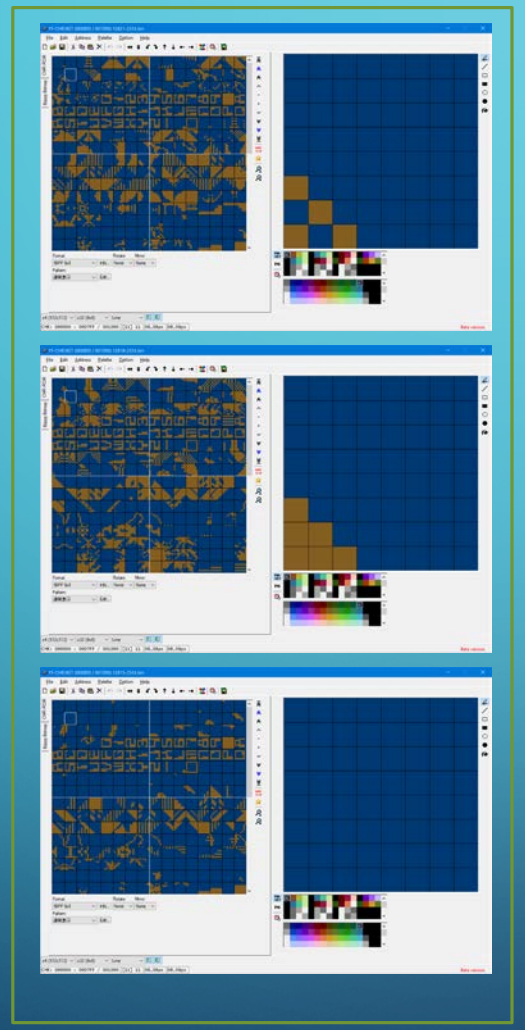
EL FIN DEL TIEMPO

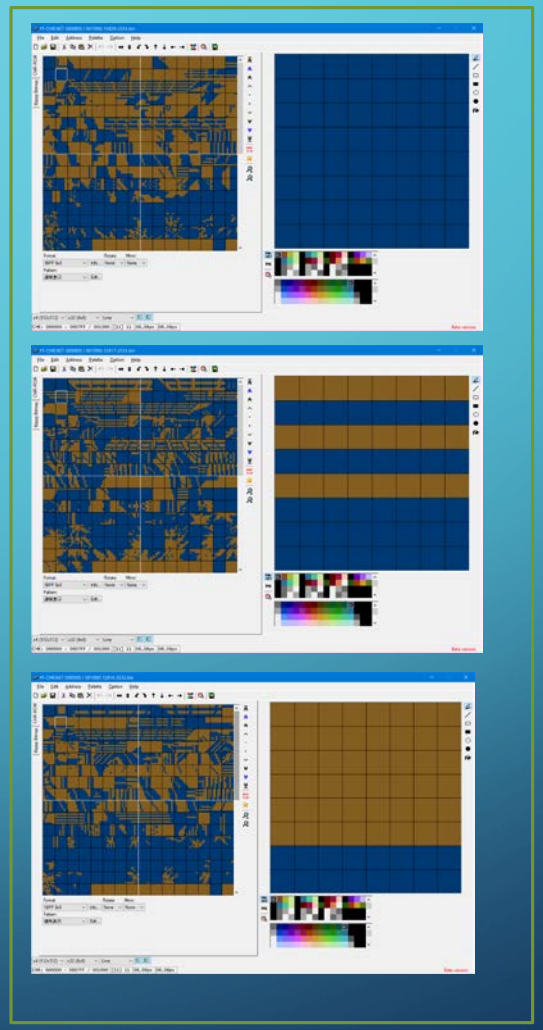
- Gráficos
- Ya sabemos qué ROMS son programa principal y cuales programa de audio.
- El resto serán gráficos. Hay 9 juntas y una aparte.
- Cogemos una de las 9. Vamos decodificando diferentes formatos
- ¡8x8 1bpp hay graficos!

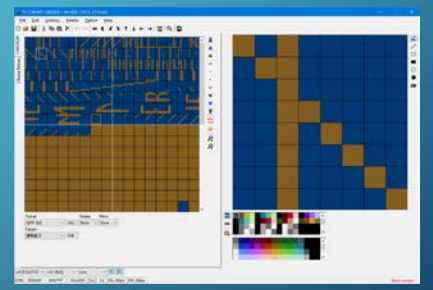


EL FIN DEL TIEMPO









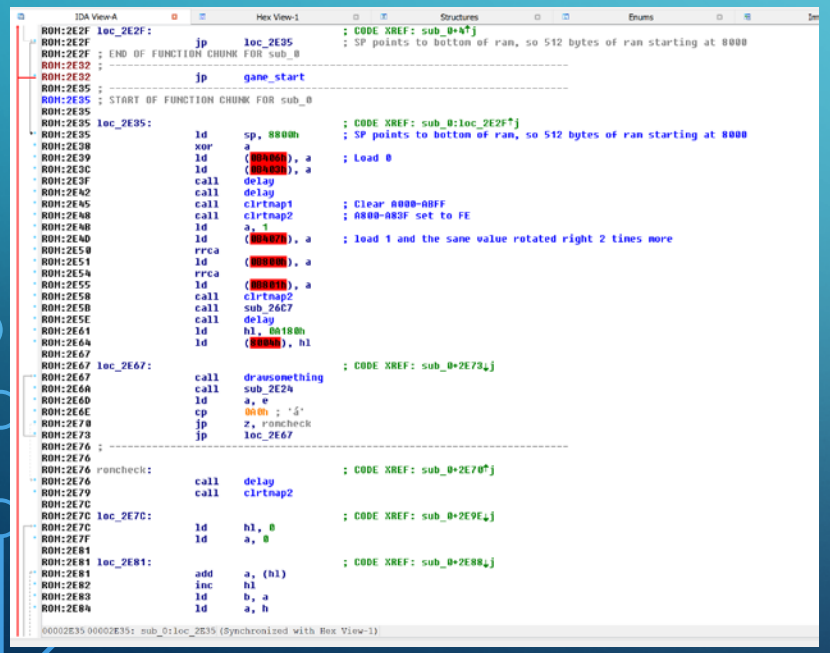
EL FIN DEL TIEMPO

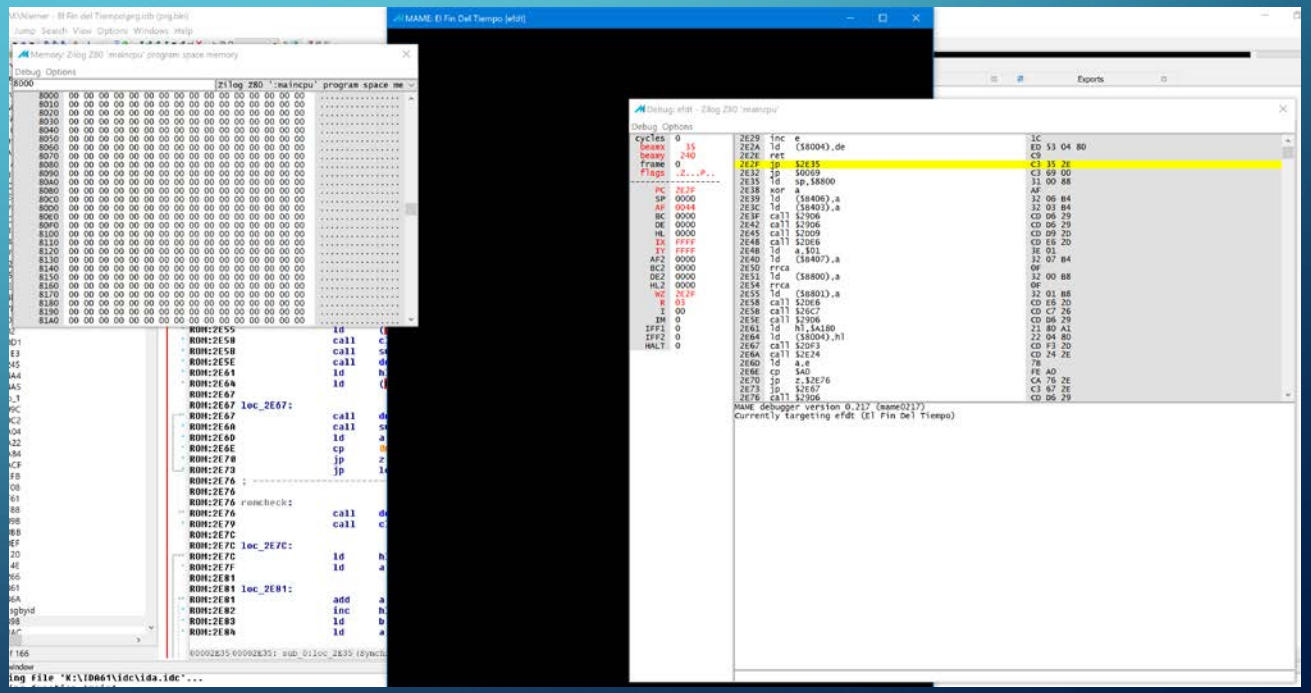
- 3 bancos de tiles de 8x8 de 3bpp
- 1 banco de tiles de 8x8 de 1bpp
- Podemos ya cargarlos en MAME

```
ROM_REGION( 0x8000, "maincpu", 0 )
          ROM_LOAD( "22805.b10", 0x0000, 0x1000, CRC(7e27df91) SHA1(0f2ef3563af5d6e0de77a6ac929dbd3802aea8f0))
          ROM_LOAD( "22806.b9",  0x1000, 0x1000, CRC(00cad810) SHA1(e93f9365227c1e5c2dea1325b379e78e37c0a953))
          ROM_LOAD( "22807.b8",  0x2000, 0x1000, CRC(8e51af2b) SHA1(ac496781fb599d26905aa28449eefc3959de0e9a))
          ROM_LOAD( "22808.b7",  0x3000, 0x1000, CRC(932bd16d) SHA1(3df0f222b0803da9021d3144ec7bc28453fdd947))
          ROM_LOAD( "22801.a10", 0x4000, 0x1000, CRC(ea646049) SHA1(bca30cb2dde8b5c78f6108cb9a43e0dce697f761))
          ROM_LOAD( "22802.a9",  0x5000, 0x1000, CRC(74457952) SHA1(f5f4ece564cbdb650204ccd5abdf39d0d3c595b3))

          ROM_REGION( 0x2000, "audiocpu", 0 )
          ROM_LOAD(   "1811.d8",   0x0000, 0x1000, CRC(0ff5d0c2) SHA1(93df487d3236284765dd3d690474c130464e3e27))
          ROM_LOAD(   "1812.d7",   0x1000, 0x1000, CRC(48e5a4ac) SHA1(9da4800215c91b2be9df3375f9601b19353c0ec0))

          ROM_REGION( 0x9000, "gfx1", 0 )
          ROM_LOAD( "12822.j3", 0x2000, 0x1000, CRC(f4d28a60) SHA1(bc1d7f4392805cd204ecfe9c3301990a7b710567) )
          ROM_LOAD( "12821.j4", 0x1000, 0x1000, CRC(b7ef75a6) SHA1(057f737fce63639879db95659de7d1d659058759) )
          ROM_LOAD( "12820.j5", 0x0000, 0x1000, CRC(70126c8d) SHA1(f380868f3afad2898c136b15210aaa6231f1d3c2) )
          ROM_LOAD( "12819.j6", 0x5000, 0x1000, CRC(2987b5b6) SHA1(0e57aae21e674155e407512f1edfb3d8b31d1fa3) )
          ROM_LOAD( "12818.j7", 0x4000, 0x1000, CRC(e0a61419) SHA1(65caa9e2700a0bec9e105dc763e9fe61dae8c3d6) )
          ROM_LOAD( "12817.j8", 0x3000, 0x1000, CRC(856a2537) SHA1(5e8f96239721a0dd64b37267bb3b343ac3034898) )
          ROM_LOAD( "12816.j9", 0x8000, 0x1000, CRC(69664044) SHA1(57465c4c37be2b4846b49a13dec9e354dabb155a) )
          ROM_LOAD( "12815.j10", 0x7000, 0x1000, CRC(abe7a7b6) SHA1(e3bc6aa3a741fcfa2eafb1464be3cb5437d5fd90) )
          ROM_LOAD( "12814.j11", 0x6000, 0x1000, CRC(6c06f746) SHA1(c7e80c5dde733e9ef520b9afa78bed902f04b74d) )

          ROM_REGION( 0x800, "gfx2", 0 )
          ROM_LOAD( "12813.h10", 0x0000, 0x0800, CRC(ea03c5a8) SHA1(7ce385b43a24cbbc780162ed89031d1cc1b0b9ef))
```

EL FIN DEL TIEMPO

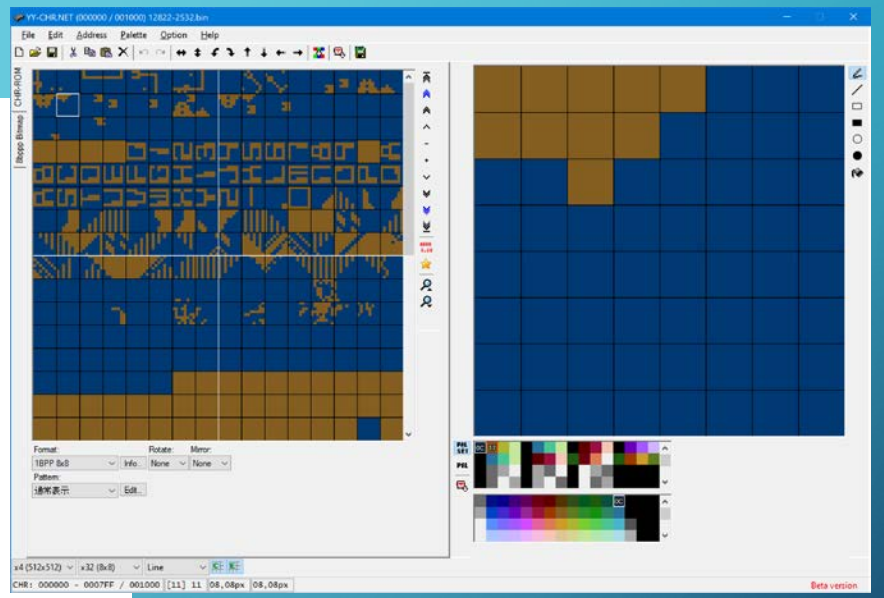
- Ponemos un mapa de memoria con el resto RAM, y vamos con el debugger de MAME analizando el código, y comentándolo en IDA

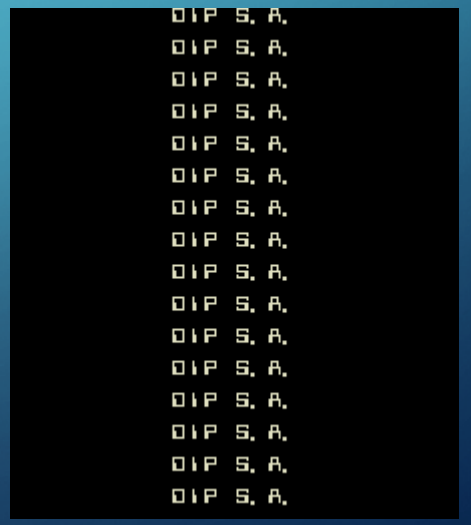




EL FIN DEL TIEMPO

```
ROM:2DF3
ROM:2DF3 drawsomething:                          ; CODE XREF: sub_0:loc_2E67↓p
ROM:2DF3                ld      hl, (8004h)
ROM:2DF6                ld      (hl), 5Ah ; 'Z' ; .   HL is A180 here
ROM:2DF8                ld      a, 20h ; ' '
ROM:2DFA                add     a, l
ROM:2DFB                ld      l, a
ROM:2DFC                ld      (hl), 3Fh ; '?' ; A
ROM:2DFE                ld      a, 20h ; ' '
ROM:2E00                add     a, l
ROM:2E01                ld      l, a
ROM:2E02                ld      (hl), 5Ah ; 'Z' ; .
ROM:2E04                ld      a, 20h ; ' '
ROM:2E06                add     a, l
ROM:2E07                ld      l, a
ROM:2E08                ld      (hl), 51h ; 'Q' ; Q + 2 -> S
ROM:2E0A                inc     h
ROM:2E0B                ld      a, 20h ; ' '
ROM:2E0D                add     a, l
ROM:2E0E                ld      l, a
ROM:2E0F                ld      (hl), 0FEh ; '¦' ; space
ROM:2E11                ld      a, 20h ; ' '
ROM:2E13                add     a, l
ROM:2E14                ld      l, a
ROM:2E15                ld      (hl), 4Eh ; 'N' ; N + 2 -> P
ROM:2E17                ld      a, 20h ; ' '
ROM:2E19                add     a, l
ROM:2E1A                ld      l, a
ROM:2E1B                ld      (hl), 47h ; 'G' ; G + 2 -> I
ROM:2E1D                ld      a, 20h ; ' '
ROM:2E1F                add     a, l
ROM:2E20                ld      l, a
ROM:2E21                ld      (hl), 42h ; 'B' ; B + 2 -> D
ROM:2E23                ret
ROM:2E23 ; End of function drawsomething
ROM:2E23
```





EL FIN DEL TIEMPO

- Con el tilemap principal ya podemos “ver” el juego y con el debugger ir buscando mas zonas

```
VIDEO RAM STRUCTURE
  A000-A3FF - 3bpp Tile layer tile code
  A400-A7FF - ?? doesn't seem used
  A800-A83F - Tile column scroll on even bytes. Tile column
palette on odd bytes.
  A840-A85F - 8 sprites. 4 bytes per sprite:
                 76543210
               0 YYYYYYYY   Y position of the sprite
               1 yxCCCCCC   x: xflip  y: yflip C: sprite code
(of a 4 sprites block)
               2 -----PPP   P: Palette
               3 XXXXXXXX   X position of the sprite
  A860-A87F - Bullets. 4 bytes per bullet
               0 --------  Unknown (X pos high byte?)
               1 XXXXXXXX  X position of the bullet
               2 --------  Unknown (Y pos high byte?)
               3 YYYYYYYY  Y position of the bullet
  AC00-AFFF - 1bpp Tile layer tile code
```

```
VIDEO REGISTERS
  Most video registers are unknown :(
  B400 - This is usually 1, but changes to 2 sometimes during scenes, also toggles 0x80 on and off in the interrupt
handler before changing some regs (disable interrupt while on interrupt?)
  B401 - Always 00
  B402 - Always 00
  B403 - Starfield on/off. Pretty sure it is.
  B404 - Always FF
  B405 - Tile column scroll on/off. Almost, but not, because it's set to 0 on the galaxian stage, and it has
scrolling on the top rows
  B406 - 1bpp tilemap on/off ?. Almost but not, because it's set to 0 on the title screen and the niemer text is on
the 1bpp layer and must appear
  B407 - Tile bank
  B800 -\-- these 2 values contain ror(Tilebank,1), ror(Tilebank,2). Sprites bank? always set to the same than Tile
bank reg
  B801 -/
  B802 -\   these 3 registers usually contain x, ror(x,1), ror(x,2) and are related to the 1bpp bitmap palette color.
2 is red, used for the "galaxian" level lines
  B803 -|-- during the initial scene of the attract, when the bomb explodes, they cycle 1,2,3,3,3,3,4,5,6 to cycle
several colors, yellow, blue and red
  B804 -/   in the logo screen, when it says "Fin del tiempo", the "Niemer" letters must be orange/brown, these are
set to 3. Set to 7 in the survival stage (red laser)
  B805 - Always 00
  B806 - Always 00
  B807 - Always 00
```

```
Some video register dumps (B400):
  01 00 00 01 FF 01 01 00 - first attract screen, starfield on, scroll not used, 1bpp layer used (for explosion)
  01 00 00 01 FF 01 00 02 - first level (scrolling) , starfield on, scroll used, 1bpp layer not used
  01 00 00 01 FF 01 01 03 - 2nd level (survival), starfield on, scroll not used, 1bpp layer used (for laser)
  01 00 00 00 FF 01 01 04 - 3rd level (rescue), starfield off, scroll not used, 1bpp layer used (for tentacles)
  01 00 00 01 FF 00 01 01 - 4rd level (galaxian), starfield on, scroll used, 1bpp layer used (red lines)
  01 00 00 01 FF 01 01 05 - last level, starfield on, scroll not used, 1bpp layer not used
  01 00 00 00 FF 01 00 01 - Logo screen, starfield off, scroll not used, 1bpp layer used (niemer logo)
  01 00 00 00 FF 01 00 01 - Scoring, starfield off, scroll used, 1bpp layer not used
```

EL FIN DEL TIEMPO

```
void efdt_state::efdt_map(address_map &map)
{
	map(0x0000, 0x7fff).rom().region("maincpu", 0);
	map(0x8000, 0x87ff).ram();

	map(0x8800, 0x8803).rw(FUNC(efdt_state::main_soundlatch_r), FUNC(efdt_state::main_soundlatch_w));

	map(0x9000, 0x93ff).portr("P1");
	map(0x9400, 0x97ff).portr("P2");

	map(0xa000, 0xafff).ram().share("videoram");
	map(0xb000, 0xb000).r("watchdog", FUNC(watchdog_timer_device::reset_r));
	map(0xb400, 0xb407).w(m_vlatch[0], FUNC(ls259_device::write_d0));
	map(0xb800, 0xb807).w(m_vlatch[1], FUNC(ls259_device::write_d0));
}

void efdt_state::efdt_snd_map(address_map &map)
{
	map(0x6000, 0x6000).nopw();
	map(0x7000, 0x7000).nopw();
	map(0x8000, 0x83ff).ram();

	map(0x9000, 0x9000).rw("ay1", FUNC(ay8910_device::data_r), FUNC(ay8910_device::data_w));
	map(0x9200, 0x9200).w("ay1", FUNC(ay8910_device::address_w));

	map(0x9400, 0x9400).rw("ay2", FUNC(ay8910_device::data_r), FUNC(ay8910_device::data_w));
	map(0x9600, 0x9600).w("ay2", FUNC(ay8910_device::address_w));

	map(0xe000, 0xffff).rom().region("audiocpu", 0);
}
```

EL FIN DEL TIEMPO

- Tras un par de semanas el juego funcionaba… casi
- La fase “galaxian” no se podía pasar
  - La nave no llegaba al borde
  - Los enemigos eran infinitos
- 1 mes depurando código y al final…
- Una EPROM tenia un bit “flojo”

EL FIN DEL TIEMPO

```
ROM:1029 loc_1029:                                       ; CODE XREF:
ROM:1029                ld      a, 1
ROM:102B                ld      (8147h), a
ROM:102E                ld      (1D4h), a
ROM:1031                call    55Ch
ROM:1034                call    2702h
ROM:1037                call    2702h
ROM:103A                call    26C7h
```

```
ROM:1029 ; ---------------------------------------------------------------
ROM:1029
ROM:1029 loc_1029:                                       ; CODE XREF: ROM:1014↑j
ROM:1029                ld      a, 1
ROM:102B                ld      (8147h), a
ROM:102E                ld      (81D4h), a
ROM:1031                call    55Ch
ROM:1034                call    2702h
ROM:1037                call    2702h
ROM:103A                call    26C7h
ROM:103D                ld      a, 40h ; '@'
```

EL FIN DEL TIEMPO

PROTECCIONES DE JUEGOS ARCADE

- “Caja Negra”
- Konami
  - Bucky O’Hare
    - Cálculo de intersección de 2 rectángulos
  - C.O.W. Boys of Moo Mesa
    - DMA con operaciones aritméticas
- NeoGeo
  - Metal Slug X
    - Scrambling de 1024 bits
  - Garou, King of fighters 99
    - Scrambling de datos, direcciones y bankswitch
  - King of Fighters 2003
    - Conversión de paletas RGB <> Neogeo

IGS POLYGAMEMASTER

- Hardware “estandar”: 68k, Z80, tilemaps, sprites, pero…
- Los sprites estaban en un formato desconocido
- Corriendo código en la placa ver cómo es el Sprite 0 y otros códigos de Sprite válidos
- En resumen, partidos en 2 ROMS
  - ROM B: puntero a ROM A y bitmask de si el pixel es transparente
  - ROM A: solo los pixels opacos, 5BPP empaquetando 6 en 32 bits (y 2 bits no usados)

IGS POLYGAMEMASTER

- Protección
- Martial Masters: Comandos sin bounds check. Dumpear la memoria “a fotos” con una BIOS custom.

CAPCOM SYSTEM 3

- Hardware peculiar, pero “simple”
- ROMS de programa encriptadas -> para los magos de la encriptación de MAME
- Como siempre gráficos custom
- Los patrones en la ROM eran muy raros
- Bloques de 256 bytes, y a continuación bloques grandes sin estructura de tiles, aparentemente comprimidos.
- No podíamos correr código en la placa

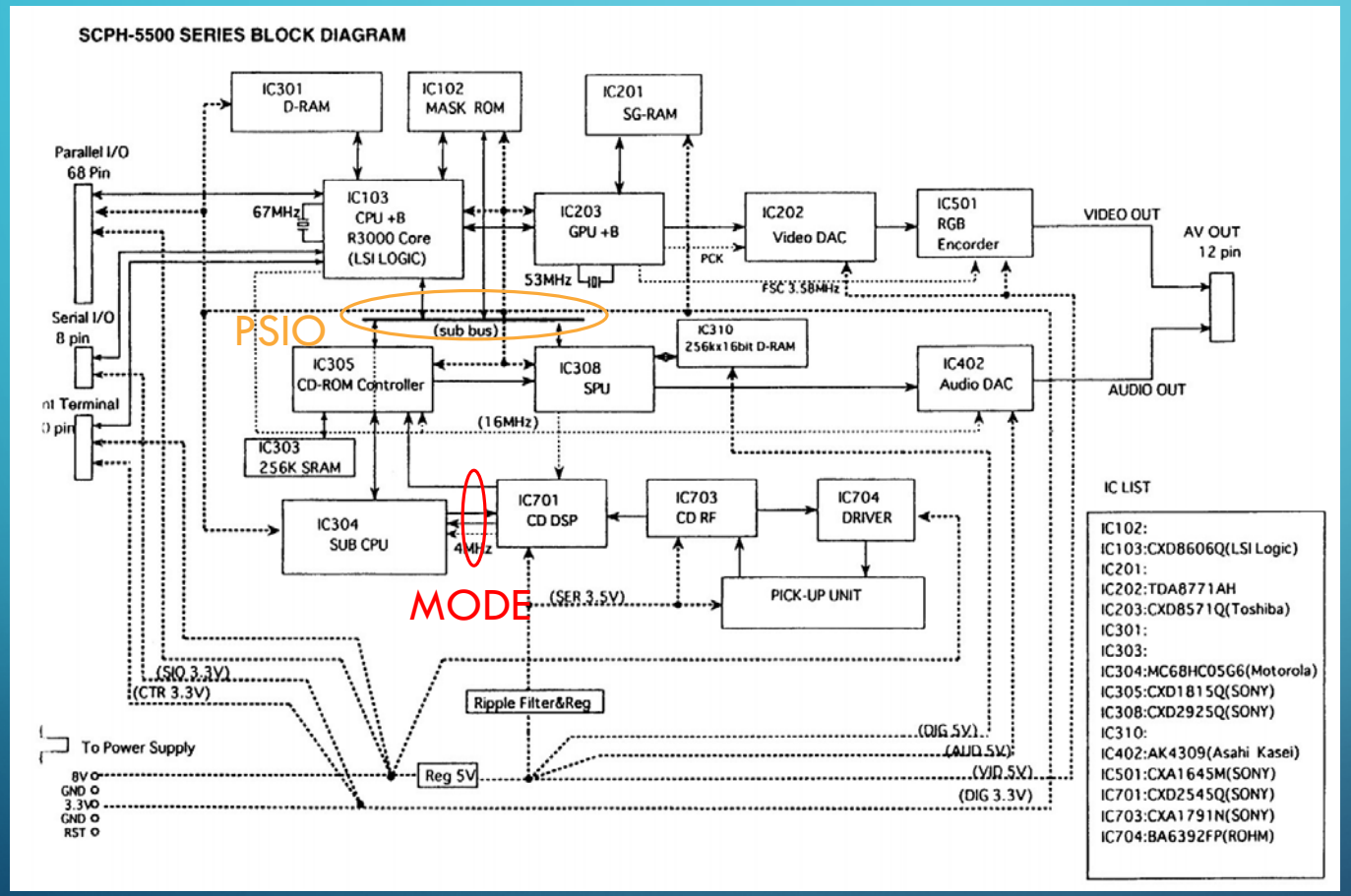
CAPCOM SYSTEM 3

- El hardware gráfico tenia un DMA que descomprimía al vuelo
- 2 punteros, uno para la tabla y otro para los datos
- La tabla eran 128 entradas de 2 bytes
- Los datos:
  - Primero, si el byte tenía el bit alto a 1, era un índice (128 elementos) en la tabla, y se sustituía ese byte por los 2 que había en la tabla
  - Tras el paso de sustitución, si el siguiente bit era 1, los demás bits (de 0 a 63) indicaban las veces que se repetía el siguiente byte (RLE)

PLAYSTATION 1 ODE

- Existe un Drive Emulator (PSIO)
- Se conecta al bus de memoria. Problemas de compatibilidad
- Lo ideal es emular el láser, pero es analógico y complicado
- Paso intermedio, conectarse a la salida del DSP

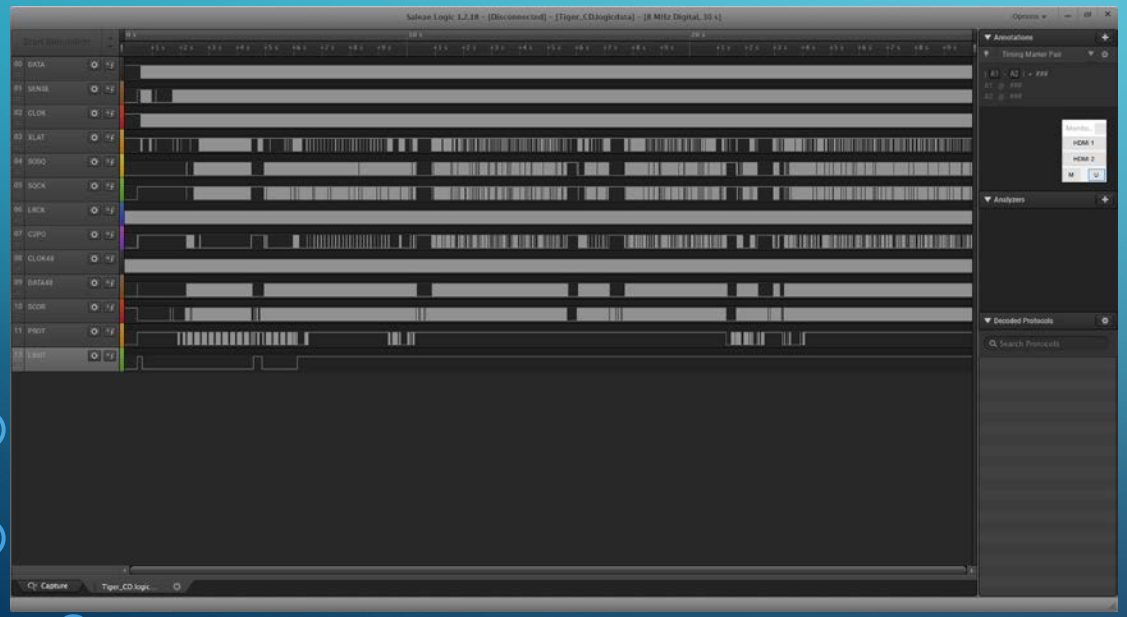
PLAYSTATION 1 ODE

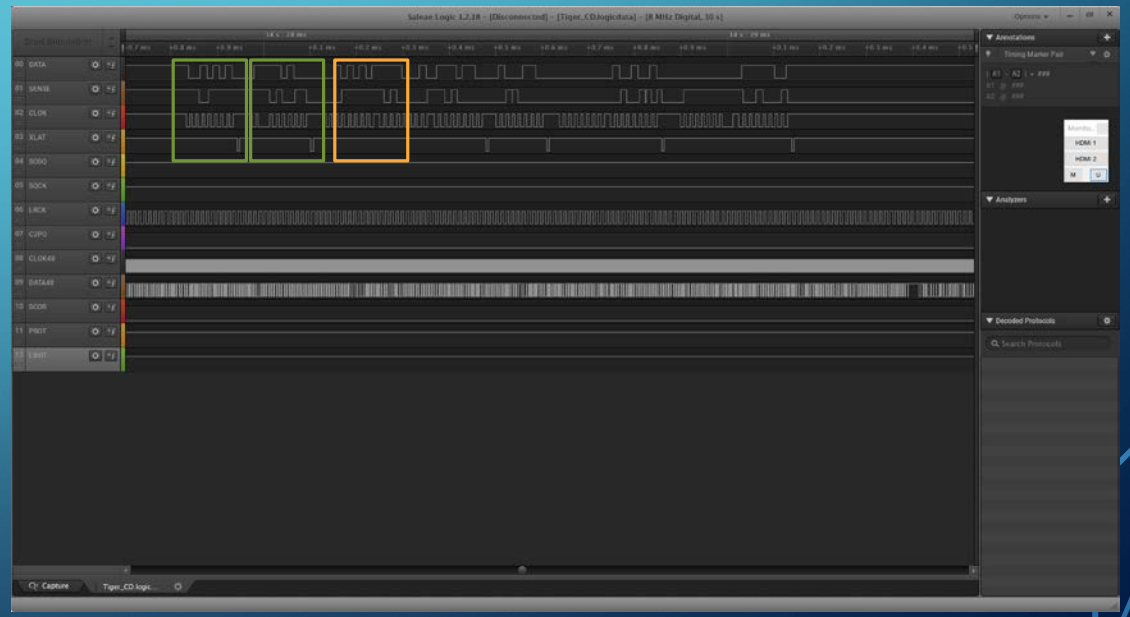


- Datos y audio en formato serie normal (Similar a I2S): DATA, CLOCK, LR (datasheet)
- Comandos: DATA, CLOCK, XLAT, SENSE (datasheet)
- Subcanal: SUBQ, SQCK, SCOR (datasheet)
- Protección (puerto serie): GATE, PDATA
- Otros: switch de tapa de cd, switch de fin de carrera

PLAYSTATION 1 ODE

- La CapturingStation
- Aprox 1GB de datos cada 30 segundos





PLAYSTATION 1 ODE

- Herramienta de proceso de capturas

```
39040370 4880.05ms XLAT 31 37 49 SELECT Kick +2Unknown
39040411 4880.05ms SENSEX 31 MANY 0
39041428 4880.18ms XLAT 17 31 37 Tracking BRAKE
39041470 4880.18ms SENSEX 17 AS 0
39042475 4880.31ms XLAT 25 17 31 Tracking MODE TRACKING SERVO ON. SLED SERVO ON
39042517 4880.31ms SENSEX 25 TZC 0
39043609 4880.45ms XLAT E6 25 17 Spindle CLV (Play)
39043650 4880.46ms SENSEX E6 OV64 1
39044633 4880.58ms XLAT 08 E6 25 Focus ON. Gain normal
39044674 4880.58ms SENSEX 08 FZC 0
39047642 4880.96ms SENSE 40 XBUSY 1
39047884 4880.99ms SENSE A0 GFS 0
39051500 4881.44ms SENSE A0 GFS 0
39052695 4881.59ms SENSE 50 FOK 1
39053779 4881.72ms SENSE A0 GFS 0
39056984 4882.12ms SENSE A0 GFS 0
39065446 4883.18ms SENSE A0 GFS 0
39066640 4883.33ms SENSE 50 FOK 1
39067725 4883.47ms SENSE A0 GFS 0
39073140 4884.14ms SENSE A0 GFS 0
```

PLAYSTATION 1 ODE

- Hay que simular las señales del CD
  - Focus
  - Sync
  - Cambio de reflectividad al atravesar una pista
  - …
- Y la protección
  - Señal serie modulada en la desviación (Tracking Error) de la pista en el Lead-in
  - Devuelve SCEA, SCEE o SCEI en función de la región, con su bit de start, de stop, …

PLAYSTATION 1 ODE

- Emulación a bajo nivel del posicionamiento
  - El MECHACON estima la distancia
  - Mueve el carro
  - Cuenta pistas por cambio de reflectividad
  - Frena poniendolo “en reversa”
  - Escucha subcanal, recalcula distancia y repite hasta estar suficientemente cerca.
  - Activa el stream de datos.

PLAYSTATION 1 ODE

- Prototipo

Final

PREGUNTAS