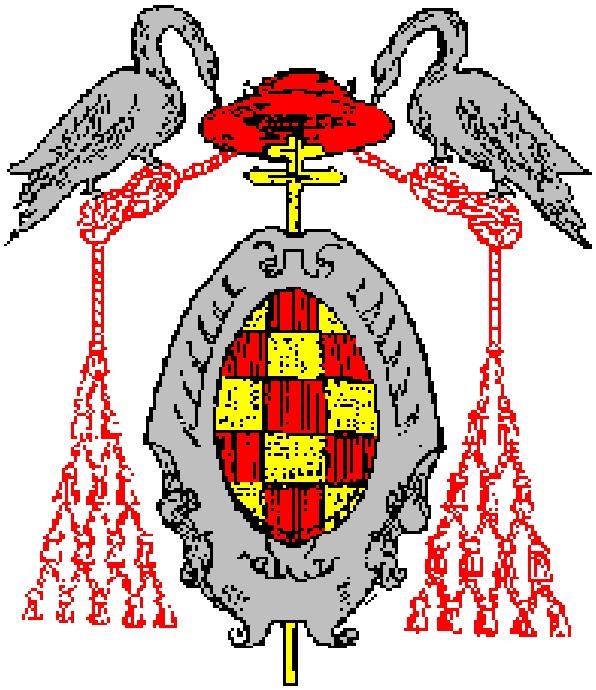
**UNIVERSIDAD DE ALCALÁ**

Escuela Politécnica

INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN
(ESPECIALIDAD TELEMÁTICA)



Trabajo Fin de Carrera

# EMULADOR DE RECREATIVAS "MULTIPLE ARCADE MACHINE EMULATOR" PARA LA CONSOLA GP32

**Francisco Javier Martínez Romo**

**Junio de 2005**

**UNIVERSIDAD DE ALCALÁ**
Escuela Politécnica

INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN

(ESPECIALIDAD TELEMÁTICA)

Trabajo Fin de Carrera

# EMULADOR DE RECREATIVAS "MULTIPLE ARCADE MACHINE EMULATOR" PARA LA CONSOLA GP32

**Autor:** Francisco Javier Martínez Romo

**Director:** José Luis Martín Sánchez

**TRIBUNAL:**

**Presidente:** D. Ignacio Bravo Muñoz

**Vocal 1º:** D. Claudio Fernández González

**Vocal 2º:** D. José Luis Martín Sánchez

**CALIFICACIÓN:**.....

**FECHA:**.....

# ÍNDICE

# LISTA DE FIGURAS

# 1. RESUMEN

La GP32 es una consola portátil desarrollada por la empresa *GamePark.*

El objetivo del proyecto ha sido portar el emulador de recreativas MAME (*Multiple Arcade Machine Emulator*) a la consola, logrando que este funcione con un rendimiento similar al de un PC de similares prestaciones.

Para el desarrollo del proyecto se ha hecho uso de un entorno de desarrollo libre basado en GNU GCC y un emulador para Windows.

Todo es fruto de un reto personal, un ejercicio de análisis y programación en un gran proyecto en C, y una experimentación de las capacidades de procesamiento de la GP32.

# 2. MEMORIA

## 2.1 INTRODUCCIÓN

La consola GP32 modelo BLU es una máquina portátil desarrollada por la empresa coreana *Gamepark*. Sus características principales son:

- Procesador RISC ARM9 de 166 MHz
- Memoria interna de 8 Mb SDRAM
- Pantalla TFT LCD de 3.5’’ con retro-iluminación y paleta de 65.536 colores.
- Sonido 44.1 KHz estereo de 16 bit.
- Entrada para tarjetas de memoria *Smart Media Card* (SMC).
- Conexión USB.
- Conexión para módulo RF.
- Entorno de desarrollo libre y gratuito, por lo que hay una gran cantidad de desarrolladores amateurs por todo el mundo realizando desde cero ó portando software para la consola, desde videojuegos, emuladores de distintos ordenadores, consolas y recreativas, mapas de ciudades, GPS, *Linux*, etc.

El proyecto ha consistido en portar el emulador de recreativas MAME *(Multiple Arcade Machine Emulator)* a la consola portátil, intentando lograr que este funcione al 100% con un rendimiento similar al de un PC Pentium de similares características ó superiores.

En principio es fruto de un reto personal, ya que se ha intentado realizar esto mismo por gran cantidad de personas y todos los proyectos han sido abandonados con el tiempo, debido a la gran complejidad del código fuente del emulador y a distintos problemas de portabilidad.

También es un ejercicio de análisis de código de terceros y desarrollo de nuevo código fuente para un gran proyecto en C, lo cual se podrá aplicar como experiencia en otros proyectos en lenguaje C.

Por último es una experimentación de las capacidades reales de procesamiento de estos dispositivos con procesadores RISC ARM, lo cual se puede aplicar a otros dispositivos con parecido hardware, como *Nokia nGage*, *Gameboy Advance*, etc.

Los trabajos realizados se pueden reducir a grandes rasgos a los siguientes:

- Elección de la versión del MAME a portar a la consola portátil. Se ha elegido la versión 0.34 (con fecha de diciembre de 1998) por ser pensada para funcionar en un Pentium con una capacidad de procesamiento similar a la de la GP32. De entre las versiones para las distintas plataformas, se ha decidido utilizar como base la versión de MS-DOS, aunque se ha estudiado y reutilizado código de otras versiones.
- Comprensión del código fuente del MAME: El tamaño del código fuente de la versión 0.34 del MAME es de aproximadamente 14 Mb de código en C. Por tanto es necesario realizar un análisis inteligente de tal cantidad de información, para planificar la mejor forma de ir portando cada uno de los módulos fuentes.
- Hay código fuente genérico que no tiene en cuenta el interfaz a bajo nivel con el hardware. Este código es relativamente fácil de portar, aunque ha habido que sustituir diversas funciones estándar de ANSI C por sus correspondientes funciones específicas del entorno de desarrollo de la consola (GPSDK). Por ejemplo: las funciones de gestión de memoria.
- Hay otro código fuente que ha sido necesario re-escribir desde cero: salida de video, salida de audio y controles (joystick y botones) de la consola, entrada / salida en la tarjeta de memoria SMC, etc.
- Hay que eliminar código fuente no imprescindible (grabación de partidas por ejemplo) ó no aplicable (partes de código sólo aplicable a otras arquitecturas).
- Cambios en el código fuente en aras de compatibilizar el código fuente para MS-DOS x86 con las particularidades del procesador RISC ARM9 que tiene la consola. Por ejemplo: la alineación estricta de la memoria, el direccionamiento, la *MMU (Memory Management Unit)*, etc.
- Por último explotar las características del procesador RISC ARM para optimizar en la medida de lo posible el código fuente. Esto implica por ejemplo cambiar el emulador interno del microprocesador Z80 por otro distinto que ofrezca mejor rendimiento en esta arquitectura.

Se ha utilizado un entorno de desarrollo libre basado en GNU GCC para Windows, más un emulador de la consola GP32 para Windows y una utilidad de generación de ficheros de imágenes de tarjetas SMC. Con este entorno no ha sido necesario acudir al hardware real (la consola) constantemente, sino sólo para probar el rendimiento en el hardware real ó apreciar características no emuladas por el emulador para Windows (por ejemplo el sonido).

## 2.2. LA CONSOLA PORTATIL GP32

### 2.2.1. HARDWARE DE LA GP32

La consola GP32 modelo BLU es una máquina portátil desarrollada por la empresa coreana *Gamepark*. La página web es la siguiente: http://www.gamepark.com/

**Figura 1: Logotipo de GamePark**

Apareció en el mercado asiático hace ya unos años, aunque en España no fue distribuida hasta el pasado mes de junio de 2004 por la empresa *Virgin Play*. Su página web es: http://www.virginplay.es/

**Figura 2: Logotipo de Virgin Play**

Los primeros modelos de la consola GP32 no tenían iluminación propia, posteriormente aparecieron las GP32 modelo FLU (*Front Light Unit*) con iluminación realizada desde los bordes de la pantalla. El modelo finalmente distribuido en España es el BLU (*Back Light Unit*) que tiene mejor calidad en la iluminación ya que esta se realiza desde detrás de la propia pantalla en el interior de la consola. En cualquier caso salvando estos detalles en la iluminación de la pantalla, las características técnicas de todos los modelos de la consola GP32 son idénticas.


**Figura 3: Foto de la consola GP32**

El precio de salida de la consola en España fue de 200 euros, aunque el precio actual es de unos 115 euros (sin tarjeta de memoria *SmartMediaCard* imprescindible para su uso). El precio aproximado de una tarjeta SMC de 128 Mb es de aproximadamente 30 euros.

Las características principales de la consola se resumen a continuación:

- Dimensiones: 147 mm x 88 mm x 34 mm.
- Peso: 163 gramos (sin pilas).
- Pantalla TFT LCD de 3.5’’.
- Resolución de pantalla 320 píxeles de ancho por 240 píxeles de alto (estándar VGA).
- Resolución de color de 8 bit (256 colores simultáneos) ó de 16 bit (65.536 colores simultáneos). La paleta de colores se forma con enteros de 16 bit con el siguiente mapeo de bits: `rrrrrgggggbbbbbi` (5 bits de rojo, 5 bits de verde, 5 bits de azul y un bit de intensidad de color).
- Procesador RISC de 32 bit Samsung ARM9 de velocidad variable y configurable por software: desde 40 MHz hasta 133 MHz. Además se puede realizar un *overclocking* seguro hasta 166 MHz (el micro ARM9 que tiene la consola es de esa velocidad, aunque la duración de las pilas se acorta lógicamente) y configurable hasta un máximo de 256 MHz.
- Memoria RAM de 8 Mb SDRAM. La velocidad del bus es configurable por software hasta un máximo de 166 MHz.
- Memoria ROM de 512 Kb con firmware actualizable.
- Sonido de 4 canales de 44.1 KHz estereo (16 bit), aunque por software se puede mezclar cualquier número superior de canales. La consola tiene físicamente dos altavoces y una conexión jack estereo para auriculares.
- Entrada para tarjetas de memoria *Smart Media Card* (SMC) desde 16 Mb hasta 128 Mb.
- Conexión para módulo RF 2.4 GHz de 4 canales (para juegos multi-jugador, intercambio de datos con otros dispositivos inalámbricos, etc.)
- Conexión con el PC mediante USB (también se puede conectar directamente la consola a otros dispositivos: discos duros portátiles, reproductores MP3, GPS, etc.)
- Alimentación: 2 pilas alcalinas de tipo AA. Se pueden usar pilas recargables (de hecho es recomendable), también se puede alimentar mediante un transformador de corriente continua de 3 voltios ó mediante una batería de litio recargable.
- Control: Joystick direccional de 8 posiciones más 6 botones: A, B, L, R, START y SELECT.

A continuación se muestra una figura con las características técnicas de la consola:

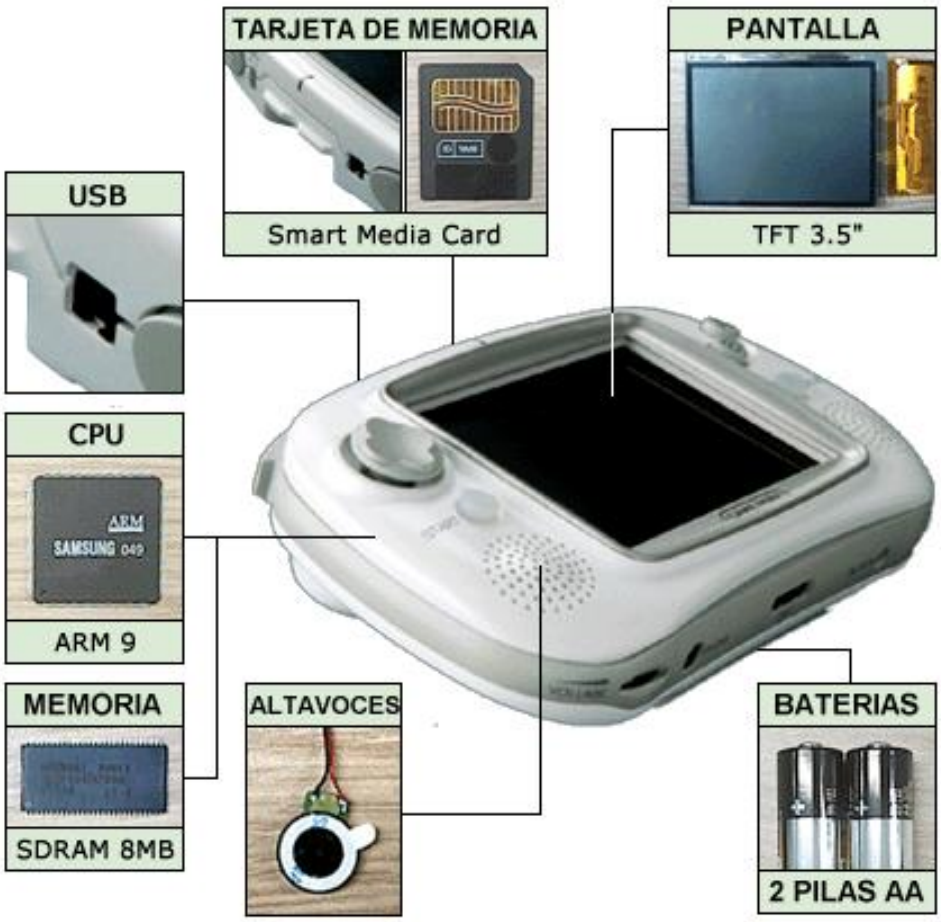



**Figura 4: Resumen de las características técnicas de la GP32**

A continuación se muestra un gráfico con el diseño de la consola:

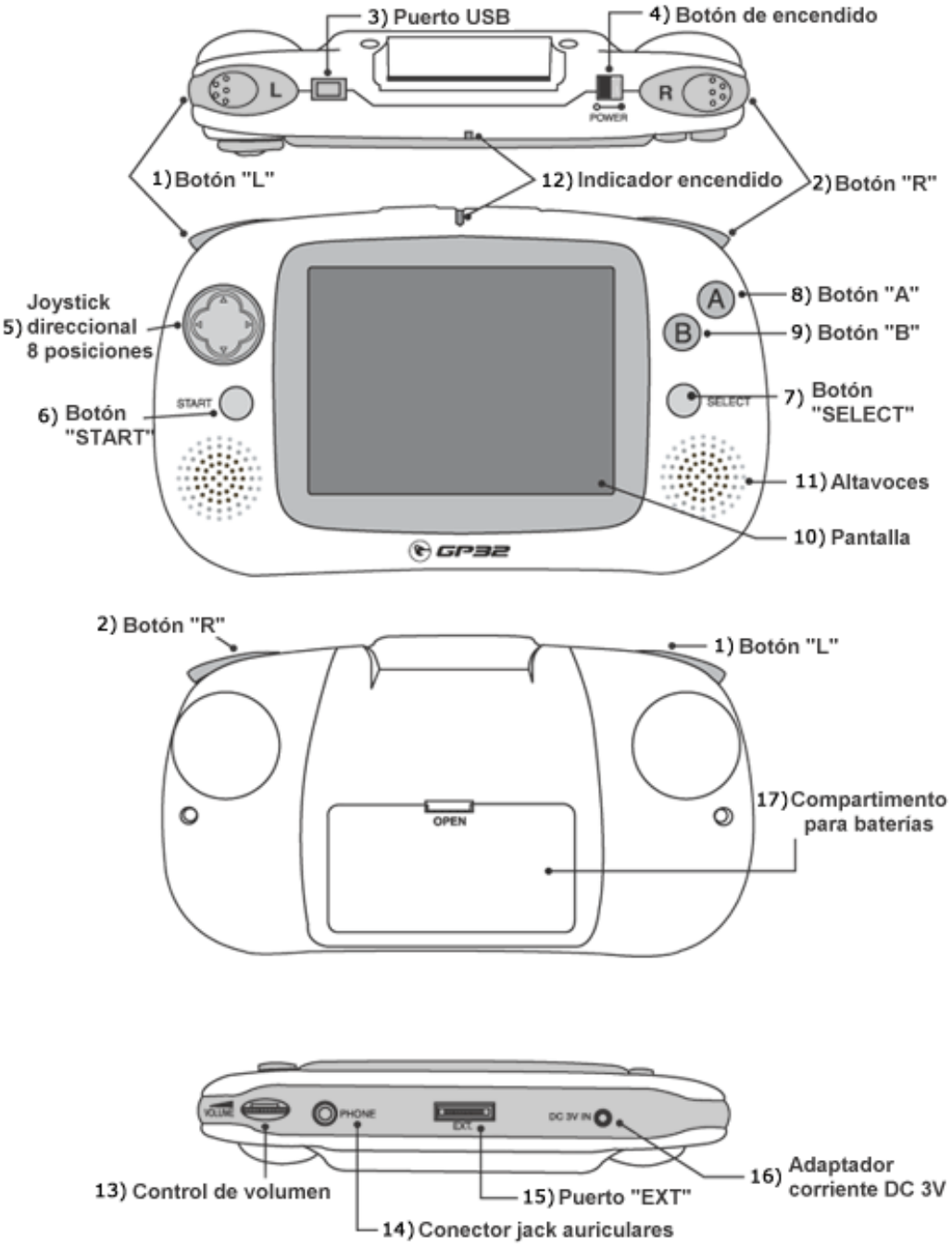



**Figura 5: Diseño de la consola GP32**

## 2.2.2. SOFTWARE DISPONIBLE

La consola viene de serie con el siguiente software:

- **Reproductor de audio MP3**: Permite escuchar temas de música en cualquier tipo de formato basado en el MP3 y con cualquier velocidad de *bitrate* fijo ó variable.

**Figura 6: MP3 Player**

- **Reproductor de vídeo Div-X y xVid**: La resolución de pantalla debe de ser como máximo de 320x240 píxeles, la conversión de películas de PC a esta resolución se realiza en unos 5 minutos con herramientas como el VirtualDub. El tamaño de una película en la consola no excede de 80 Mb.


**Figura 7: GP Cinema**

- **Visor de imágenes**: Permite visualizar imágenes con formato .BMP, .GIF, .JPG, .PCX y .TIF de cualquier resolución (por grande que esta sea).

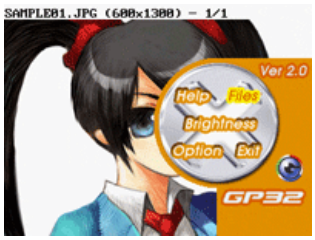



**Figura 8: Image Viewer**

- **Visor de texto**: Visualizador de texto .TXT y .RTF.

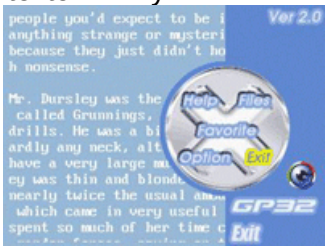



**Figura 9: Text Reader**

- **Juegos comerciales:** El catálogo de juegos comerciales para la consola es realmente escaso y es actualmente de una docena de títulos. Se muestran imágenes de algunos de ellos:


**Figura 10: Tomak saves the Earth Again**

**Figura 11: Her Knights**


**Figura 12: Dungeon & Guarder**


**Figura 13: Blue Angelo**

La falta de juegos comerciales para la consola, se compensa sobradamente por el desarrollo amateur, ya que el entorno de desarrollo de la consola es libre y gratuito. Por ello hay una gran comunidad de desarrolladores por todo el mundo realizando desde cero ó portando software para la consola, videojuegos, emuladores, etc. A continuación se muestra un pequeño resumen del gran volumen de software existente para la consola:

- **Sistemas Operativos:** Existe un port del Linux para la consola (GPLinux) que poco a poco está mejorando, así como diversos entornos visuales como el Wind-ups que simula el Microsoft Windows ó el YAFL que emula el aspecto del MAC OSX.


**Figura 14: Wind-ups**

- **Juegos amateur:** Existen cientos de juegos gratuitos originales realizados específicamente para la consola, de todos los generos. Algunos de ellos tienen un aspecto realmente profesional, como Bolcataxian, Crocoland, Nazca Dreams, etc.

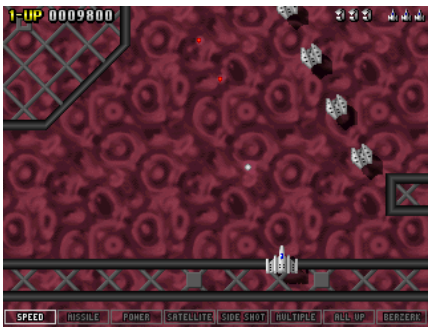



**Figura 15: Bolcataxian**


**Figura 16: Crocoland**


**Figura 17: Nazca Dreams**

- **Remakes de juegos:** Los programadores amateurs también han realizado multitud de juegos para la consola que son remakes de otros juegos clásicos. Por ejemplo: Metal Gear Solid, Lady Killer, etc.

**Figura 18: Metal Gear Solid VR Missions**


**Figura 19: Lady Killer**

- **Motores para realización de juegos:** Estos “motores” no son juegos en sí mismos, sino que sirven como base para realizar distintos juegos. Entre ellos destaca el Beats of Rage (existen decenas de juegos distintos basados en juegos clásicos de lucha 2D como Final Fight, Asterix, Kill Bill, Megaman, etc). También es particularmente interesante el KOF91 (para realización de juegos de lucha VS, como Street Fighter 2, Dragon Ball Z, etc).


**Figura 20: Beats of Rage y KOF91**

- **Adaptaciones de juegos de PC:** A partir del código fuente de los juegos originales para PC, los aficionados portan dicho código para hacerlo funcionar en la consola portátil. Entre esta gran cantidad de juegos destacan: Doom, Heretic, Hexen, Wolfenstein 3D, Quake, Rise of the Triad, Pinball Dreams, Rick Dangerous, Giana Sisters, etc.


**Figura 21: Quake, Pinball Dreams y Doom**

- **Interpretes de juegos de PC**: Son programas que simulan (no estan realizados con el código fuente de los juegos originales) el comportamiento de los ejecutables de los originales de otros sistemas. Entre estos destaca el ScummVM, que permite jugar en la consola a los viejas aventuras gráficas de LucasArts (Monkey Island, Day of the Tentacle, Loom, etc). También esta el Sarien, que sirve para jugar a las viejas aventuras gráficas de Sierra (Larry, King’s Quest, etc).


**Figura 22: ScummVM y Sarien**

- **Emuladores de recreativas:** El panorama de emuladores de recreativas esta ahora bastante menos avanzado que el resto de los otros sistemas, entre ellos destaca el Multipac, que es un emulador de una buena cantidad de recreativas clásicas basadas en el hardware del Pacman.

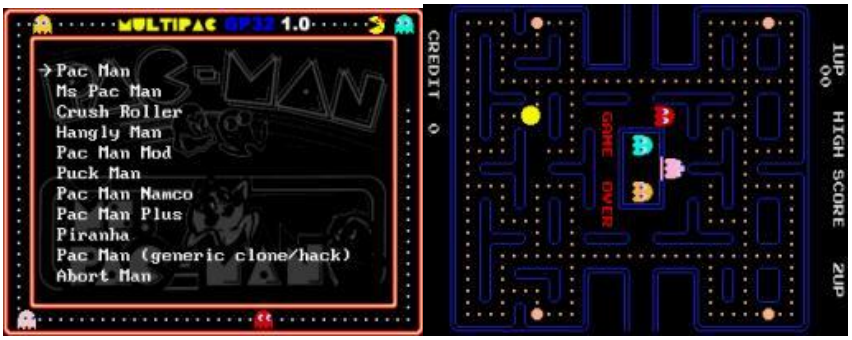



**Figura 23: MultiPac**

- **Emuladores de consolas:** DrMD (Sega Megadrive), OSNES9x (Super Nintendo), Little John (Nintendo NES), GPEngine (PC-Engine/TurboGrafx), etc.


**Figura 24: DrMD y Little John**

- **Emuladores de ordenadores:** fMSX32 (MSX), CastawayGP (Atari ST), fZX32 (Spectrum), Pituka (Amstrad CPC), etc.

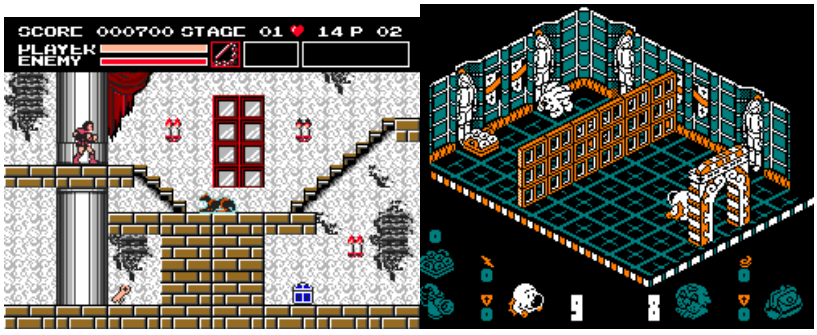



**Figura 25: fMSX32 y Pituka**

- **Reproductores de música** Ogg Vorbis, MOD, etc.

- **Visor de mapas:** Destaca el programa IMappy que permite visualizar en la consola el mapa de las principales ciudades españolas. También existe un programa junto con un hardware adicional que convierte la consola en un sistema GPS.

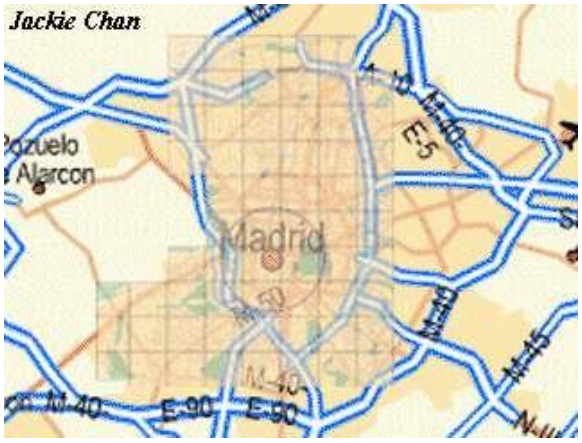



**Figura 26: IMappy**

## 2.3. MULTIPLE ARCADE MACHINE EMULATOR

### 2.3.1. INTRODUCCIÓN

El emulador M.A.M.E (*Multiple Arcade Machine Emulator*) es un gran emulador de recreativas que apareció en enero de 1997. La página web oficial del proyecto es: http://www.mame.net/

**Figura 27: Logotipo del MAME**

El “padre” del proyecto es el programador italiano *Nicola Salmoria*, que después de varios emuladores menos pretenciosos (como el emulador *Multi-Pac* que emulaba los juegos basados en el hardware del Pacman), lanzo la versión 0.1 del MAME con soporte para una decena de juegos clásicos en febrero de 1997.

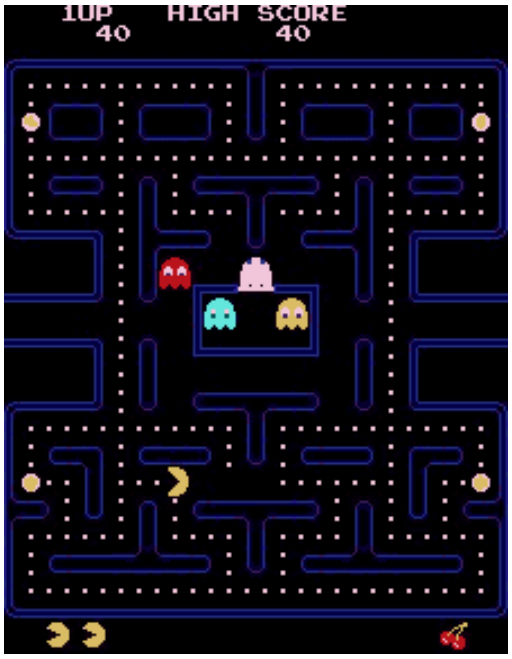



**Figura 28: Pac Man**

Hasta la salida del MAME, había multitud de pequeños emuladores de recreativas, cada uno de ellos con soporte para una pequeña colección de juegos. El objetivo del proyecto MAME era aglutinar en un único programa la emulación de todas las antiguas máquinas recreativas.

Para lograr este ambicioso objetivo se partió de las siguientes premisas, las del software libre:

- Código fuente disponible.
- Cualquier persona puede ayudar en el desarrollo del proyecto.
- Se puede portar el proyecto para que funcione en distintas arquitecturas hardware.

A lo largo de los años el proyecto ha crecido exponencialmente y actualmente la lista de gente que ha colaborado en el emulador a lo largo de los años es extensísima, y con programadores de muchos países. La última versión del emulador hasta el momento es la versión 0.95 (27/03/2005) y con soporte para más de 4000 juegos distintos.

Es sin ninguna duda el mayor proyecto que ha surgido de la gente interesada en la emulación en Internet.

La versión oficial del MAME es original de MS-DOS (ó línea de comandos de Windows en las últimas versiones), aunque también hay *ports* para sistemas tan distintos como:

- Microsoft Windows
- Unix / Linux
- Windows CE
- Teléfonos móviles (Nokia, Ericsson, etc)
- Consolas (X-Box, Sega Dreamcast, etc)
- Amiga PPC
- MAC OSX
- Solaris
- OS/2
- QNX
- RiscOS
- etc…

## 2.3.2. ¿QUÉ ES UN EMULADOR?

Un emulador es un programa que simula el comportamiento de un hardware en otra máquina completamente distinta. En el caso del MAME, es un programa que permite jugar a las antiguas máquinas recreativas en un PC.

Para conseguir el “milagro”, el emulador contiene en su interior emuladores de los antiguos procesadores usados en los juegos originales (Z80, M6809, M68000, etc.), procesadores de audio (procesadores FM, tabla de ondas, ADPCM, etc.)… en general simula el entorno hardware de los juegos originales.

## 2.3.3. ¿QUÉ SON LAS ROMS?

En las máquinas recreativas originales, los juegos estaban grabados en memorias de tipo ROM (*Read Only Memory*). Por ello, para que el MAME pueda hacer funcionar estos juegos, es necesario leer el contenido de estas ROMs y volcar su contenido a ficheros que puedan leerse desde el PC.

La lectura de todas las ROMs de uno de los juegos da lugar a una serie de ficheros que son los que el emulador puede leer para hacerlos funcionar en el PC. La lista completa de ROMs necesaria para hacer funcionar un determinado juego se llama *romset*.

El MAME actualmente soporta más de 4000 *romsets* distintos (es decir, permite hacer funcionar más de 4000 juegos distintos).

## 2.3.4. LEGALIDAD

El emulador MAME es completamente legal, ya que simula el entorno hardware de las máquinas recreativas originales, pero en ningún caso contiene código de los juegos. Además el desarrollo del emulador se ha realizado mediante la documentación técnica libre de procesadores como el Z80, Motorota 68000, procesadores gráficos, chipsets de sonido, etc.

Otro tema distinto es el tema de las ROMs. El copyright de los juegos pertenece a sus creadores, por lo que en principio sólo sería legal poseer las ROMs de un determinado juego si además se es propietario de la recreativa original, pero con las siguientes matizaciones:

- Gran parte de las empresas desarrolladoras ya no existen y sus juegos no resultan de interés para que ninguna empresa pretenda sacar un beneficio económico.

- Determinadas empresas de videojuegos permiten que sus antiguos juegos sean distribuidos libremente por Internet.

- También hay empresas que venden legalmente y por un precio módico las ROMs de determinados juegos (por ejemplo http://www.starroms.com/ vende las ROMs de las más famosas recreativas de Atari).

- El copyright de los juegos más antiguos ha preescrito (por ejemplo los juegos más antiguos de comienzos de los 80).

En general hay un gran vacío legal en el uso de estos programas tan antiguos, de hecho muchas empresas creadoras de recreativas han suministrado información técnica sobre las máquinas originales a los programadores del MAME, y en general estas empresas permiten la distribución de estos viejos juegos siempre que no choquen frente a intereses comerciales, por ejemplo:

- La empresa Hanahoo vende una recopilación del MAME en un CD junto con algunos de los juegos más recientes de la empresa Capcom. Todo absolutamente legal, ya que Hanahoo ha llegado a un acuerdo económico con Capcom.

- Empresas como Sony, Sega, Namco, Capcom ó Nintendo se aprovechan de la emulación para sacar a la venta los viejos juegos para consolas de última generación.

Para finalizar reseñar que aunque el MAME es actualmente capaz de hacer funcionar recreativas actualmente en comercialización, el soporte para estos juegos esta desactivado para no perjudicar a las empresas creadoras de los juegos.

## 2.3.5. *PORT* DEL MAME PARA LA GP32

Un “*Port*” podría definirse como la adaptación del código fuente del MAME original, para que funcione en una máquina diferente a un PC con MS-DOS.

Para realizar esta adaptación, es necesario portar el código fuente para que funcione en el hardware de destino. Esto implica:

- Rescribir las librerías necesarias (salida de vídeo, paleta de colores, reproducción de audio, controles, acceso a disco, etc.).

- Adaptar el código fuente original teniendo en cuenta el entorno de desarrollo y la arquitectura hardware de la máquina de destino.

La labor de este proyecto fin de carrera será la adaptación del MAME para que funcione en la consola portátil GP32, intentando que vaya al 100% de velocidad (proporcionando una experiencia de juego en tiempo real) con un rendimiento similar al de un PC Pentium de similares características a las de la GP32 (ó superiores).

## 2.3.6. ELECCIÓN DE LA VERSIÓN DEL MAME A PORTAR

La versión de la que se ha partido para el *port* a GP32 es la **0.34**, fechada a 31 de diciembre de 1998. Esta versión del MAME emula 1024 juegos distintos, de los cuales 706 funcionan en el *port* a GP32, el resto no ya que necesitan mayor cantidad de memoria y la consola GP32 sólo tiene 8 Mb de memoria RAM, insuficiente para los juegos más grandes.

Se ha elegido esta versión tan antigua del MAME por las siguientes razones:

- Esta versión estaba especialmente optimizada para funcionar en un PC Pentium de prestaciones similares a las de la consola GP32.

- Esta versión 0.34 fue la primera versión con soporte para música FM sin necesidad de utilizar el chip OPL de la tarjeta de sonido del PC, es decir que la música FM se sintetiza por software y se reproduce mediante la salida de audio, esto es especialmente necesario en la GP32, ya que esta consola no tiene chip OPL FM.

- El proyecto MAME ya estaba maduro en esta versión ya que habían pasado dos años de desarrollo software desde la primera versión, por ello la mayoría de juegos clásicos susceptibles de ser emulados por la consola GP32 ya funcionaban con esta versión.

- La máquina GP32 tiene un hardware limitado: 166 MHz y 8 Mb de memoria RAM. Por ello no tiene sentido utilizar versiones más modernas del MAME, ya que los requisitos de proceso y de memoria se elevarían hasta el punto de no ser factible su funcionamiento. Por ejemplo las versiones superiores requieren que el buffer de vídeo y de los sprites sea de color de 16 bit, por lo que las necesidades de memoria se dispararían.

## 2.4. ENTORNO DE DESARROLLO SOFTWARE

### 2.4.1. DISTINTOS ENTORNOS DE DESARROLLO

A continuación se enumeran a grandes rasgos los distintos entornos de desarrollo software existentes para la consola:

- **Entorno de desarrollo oficial de GamePark:** Es el entorno original que proporcionó la empresa junto al lanzamiento de la consola. Se basaba en un entorno propietario, comercial y de pago para programar directamente sobre un PC con librerías específicas para Visual C 6.0. Las librerías forman el *GPSDK,* que son el SDK (Software *Development Kit*) oficial de la consola, y existe documentación abundante sobre este entorno en Internet (en idiomas inglés y coreano). Con este se pueden generar ejecutables .EXE que funcionan sobre entorno Microsoft Windows. Una vez que el programa es depurado y finalizado, sólo es necesario realizar una compilación con el compilador ADS (ARM Developer Suite) junto con las librerías del GPSDK específicas para el procesador ARM que tiene la consola GP32. Cabe destacar que el compilador ADS no es gratuito, es de pago, pero es el mejor compilador de los existentes, con él se consigue el código ejecutable más optimizado para los procesadores de la gama ARM.

  Se puede encontrar abundante información sobre este entorno visitando las webs oficiales de la consola GP32 y de la gama de procesadores ARM:
  http://www.gamepark.com/
  http://www.arm.com/

- **Entorno de desarrollo DEVKITADV + GPSDK**: Se trata de un entorno de programación gratuito y libre para realizar aplicaciones para la consola GP32, usando el compilador GCC, el emulador de GP32 para Windows Geepee32 y el GPSDK (el SDK oficial de la consola). Es el entorno de desarrollo más utilizado actualmente, fue el primer entorno gratuito que apareció. El problema es que sólo apareció una primera versión, y lleva años sin ser actualizado.

  Más información en las siguientes webs:
  http://devkitadv.sourceforge.net/index.html
  http://chn.roarvgm.com/

- **Entorno de desarrollo GCC + MIRKOSDK**: Es un entorno similar al anterior, libre y gratuito, pero con la particularidad de que no se hace uso del SDK oficial de la consola, sino que parte de las funciones de este SDK son asumidas por las funciones estándar de C (gestión de memoria, funciones de cadenas de caracteres, etc.) y la parte más ligada al hardware de la consola con otra librería de funciones distintas al GPSDK.

  Más información en su web oficial:
  http://www.mirkoroller.de/

- **Entorno de desarrollo GCC + SDL**: Con este entorno se consigue un código más portable ante otros sistemas, ya que el SDL (*Simple DirectMedia Layer*) es un estándar bastante asumido por la comunidad mundial para realizar aplicaciones gráficas y juegos multi-plataforma. Por supuesto es un entorno gratuito. El inconveniente es que el ejecutable obtenido es menos óptimo que el obtenido con otros entornos de desarrollo.

  Más información en las páginas web:
  http://sdl-gp32.sourceforge.net/
  http://www.libsdl.org/

- **Entorno DEVKITARM + GPSDK**: Es una evolución del DEVKITADV nombrado anteriormente, con la única diferencia de que el compilador GCC esta absolutamente actualizado. Periódicamente aparecen nuevas versiones de este entorno. Con este se consiguen las mejores relaciones de rendimiento en un entorno de desarrollo gratuito (el compilador ADS nombrado anteriormente es mejor, pero es de pago).

  Más información en las siguientes páginas web:
  http://sourceforge.net/projects/devkitpro/
  http://www.devkit.tk/

A partir de estos entornos de desarrollo, nada impide usar lo mejor de todos ellos, por ejemplo usando el compilador ADS para la compilación, mezclar partes de distintos entornos de desarrollo buscando lo mejor de todos ellos, etc.

## 2.4.2. ELECCIÓN DEL ENTORNO DE DESARROLLO

A raíz de diversas pruebas con cada uno de los entornos de desarrollo nombrados anteriormente, he decidido utilizar la siguiente selección:

- **DEVKITARM r11** (versión r11 fechada el 14/02/2005): Contiene el compilador GCC 3.4.3. Es un entorno gratuito, el compilador es reciente y genera código binario muy optimizado. Parece el mejor compilador gratuito existente actualmente para generar binarios para procesadores ARM.

- **GPSDK 2.1.0:** Es el SDK oficial de *GamePark* preparado para funcionar junto al compilador gratuito GCC. Este SDK no es estándar ante otras plataformas, pero es fácil de programar y más rápido que el estándar SDL.

Todo el entorno fue descargado de las siguientes páginas web:
http://sourceforge.net/projects/devkitpro/
http://www.devkit.tk/

## 2.4.3. INSTALACIÓN DEL ENTORNO DEVKITARM+GPSDK

Para la instalación del entorno de desarrollo se siguen los siguientes pasos:

1. El entorno **DEVKITARM** viene en un ZIP autoextraible. Extraer el contenido de los ficheros en la carpeta `C:\DEKITARM` del disco duro.

2. Tal como se describe en la documentación del entorno de desarrollo y en la página web, es necesario instalar también el componente **MSYS** (*Minimal System MINGW*) disponible en la misma página web. Este componente permite al compilador GCC utilizar diversos comandos originales de Unix sobre Windows. Es necesario este componente para que el compilador GCC funcione sobre entornos Microsoft Windows. Instalar el componente sobre `C:\MSYS` en el disco duro.

3. Es necesario añadir las rutas `C:\DEVKITARM\BIN` y `C:\MSYS\1.0\BIN` al *path* del Windows. Para hacer esto en el sistema operativo Windows XP[1], acceder a “`Inicio -> Configuración -> Panel de Control -> Sistema -> Opciones Avanzadas -> Variables de Entorno`”.

   Añadir en la parte de “*Variables del Sistema*” del Windows, en la variable ***Path*** las rutas. En mi caso el resultado fue el siguiente:

   *%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;**C:\DEVKITARM\bin;C:\msys\1.0\bin***

4. Instalar el **GPSDK**, disponible para descargar desde la propia web del entorno de desarrollo. Decido instalarlo sobre la carpeta `C:\DEVKITARM\gamepark_sdk` del disco duro.

5. Por último decido modificar a mano uno de los ficheros .h del GPSDK, eliminando del fichero: `C:\DEVKITARM\gamepark_sdk\include\gpdef.h` las siguientes líneas de código que no son usadas en todo el SDK y además producen problemas de portabilidad del código fuente:

```
#undef byte
#define byte    char

#undef word
#define word    short

#undef dword
#define dword   long
```

[1] En otras versiones del sistema operativo, la forma de realizarlo puede ser distinta.

### 2.4.4. HERRAMIENTAS ADICIONALES

Además del entorno de desarrollo DEVKITARM+GPSDK, se usarán las siguientes utilidades adicionales:

1. **GEEPEE32 0.40**: Se trata de un emulador de la consola portátil GP32 para el sistema operativo Windows. Con él se puede probar el código ejecutable sin necesidad de acudir constantemente a la consola real. Su uso acelerará el desarrollo del proyecto notablemente.


**Figura 29: Emulador GeePee32**

El mapeo de los mandos de la consola en el PC dentro del emulador es el siguiente:

- Joystick de la consola: Cursores del PC (arriba, abajo, izquierda y derecha).
- Botón A de la consola: Tecla Q del teclado del PC
- Botón B: Tecla B
- Botón L: Tecla L
- Botón R: Tecla R
- Botón START: Tecla S
- Botón SELECT: Tecla D

En teoría es posible cambiar esta definición de teclas en el PC, pero en la práctica no es posible en teclados QWERTY, sólo en teclados AZERTY (idioma francés).
El emulador tiene opciones de menú para cargar un ejecutable FXE de la consola (opción de menú `File->Load->Binary`) y/o activar imágenes de tarjetas de memoria SMC (opción de menú `File->Load->SmartMedia`).

Además tiene algunas opciones de línea de comandos que permiten lanzar directamente un ejecutable de la consola. Por ejemplo:

```
geepee32 /EXE=gpmain.fxe /RUN
```

También se puede lanzar directamente con el emulador un ejecutable frente a una imagen de una tarjeta de memoria SMC. Para ello se utiliza la siguiente sintaxis:

```
geepee32 /EXE=gpmain.fxe /SMC=mame034.smc /RUN
```

La página oficial del emulador es la siguiente: http://users.skynet.be/firefly/gp32/

Aún con toda la ayuda proporcionada por el emulador, habrá ocasiones en que sea estrictamente necesario probar directamente frente al hardware real ya que el emulador tiene diversos fallos y carencias:

- Sólo emula la consola hasta una frecuencia de reloj de 66 MHz, insuficiente a todas luces para probar el emulador en tiempo real, se prevé que será necesario al menos 133 MHz de reloj para conseguir una experiencia en tiempo real en la consola.

- No emula el hardware de sonido de la consola.

**NOTA**: Aunque la última versión del emulador GEEPEE32 es la 0.43, se ha elegido utilizar la versión 0.40 algo más antigua, porque las diferencias con la última versión son muy escasas y además las últimas versiones tienen un mensaje de promoción de una página web (un sponsor) que no se puede desactivar, y que obliga a esperar unos segundos antes de arrancar el emulador, lo cual llega a ser realmente molesto tras unas cuantas pruebas.

2. **MAKESMC_128MB**: Se trata una utilidad que permite realizar imágenes de tarjetas de memoria SMC (*Smart Media Card*) que usa la consola. Con esta utilidad podremos simular el entorno real de la consola en el PC, tanto del programa compilado como de los demás ficheros existentes en la tarjeta SMC. Hay varias utilidades para distintos tamaños de tarjetas de memoria (16Mb, 32Mb, 64 Mb y 128 Mb), elegí la utilidad para 128 Mb para no quedarme corto en la capacidad de la tarjeta.

   En la consola GP32, los ejecutables van ubicados en la carpeta GPMM del raíz de la SMC, y los ficheros de datos en cualquier carpeta adicional, por ejemplo para el proyecto ubiqué los ficheros adicionales en `GPMM\MAME034.`

   Partiendo por ejemplo de los siguientes ficheros en una carpeta C:\PRUEBA\ del disco duro del PC:

   ```
   C:\PRUEBA\GPMM\GPMAIN.FXE
   C:\PRUEBA\GPMM\MAME034\GNG.ZIP
   C:\PRUEBA\GPMM\MAME034\PACMAN.ZIP
   C:\PRUEBA\GPMM\MAME034\ATETRIS.ZIP
   ```

   Para realizar la imagen de la memoria SMC sobre un fichero (por ejemplo MAME034.SMC), se debe ejecutar el siguiente comando:

   ```
   MAKESMC_128MB C:\PRUEBA C:\MAME034.SMC
   ```

Estas utilidades de creación de imágenes SMC están disponibles para descargar en la siguiente dirección web: http://users.skynet.be/firefly/gp32/project/makesmc/index.htm

3. **GP32 CONVERTER 1.3**: Esta utilidad convierte imágenes de tipo bitmap (de extensión BMP) a un formato fácilmente utilizable en la consola GP32.

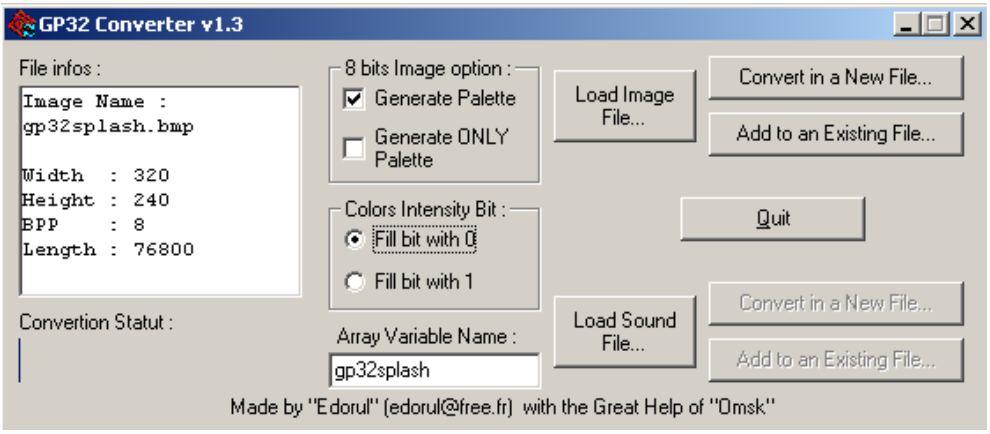



**Figura 30: GP32 Converter**

Para ello es necesario utilizar la opción “`Load Image File`” y seleccionar el fichero bitmap a convertir.

Una vez se ha elegido el bitmap, se activa la opción “`Generate Palette`” para convertir también la paleta de colores de la imagen al formato de color usado por la consola (5551, `rrrrrgggggbbbbbi`, 5 bits de rojo, 5 bits de verde, 5 bits de azul y un bit de intensidad de color).

Además se selecciona en “`Array Variable Name`” el nombre de la variable que contendrá el bitmap y/o la paleta. Por ejemplo si se pone “***gp32splash***”, se generarán dos variables:

- ***gp32splash_Pal*** : Array de de 256 elementos (supongo que la imagen tiene 8 bit de color, 256 colores). Cada entrada de la paleta es de 16 bit (`short int`), por lo que una paleta ocupa 512 bytes de memoria.

- ***gp32splash:*** Array de bytes (`unsigned char`) (supongo que la imagen de entrada es de 8 bit, por lo que cada píxel de la imagen ocupa un byte). El tamaño total del array de bytes es el resultado de multiplicar el ancho por el alto de la imagen en pixels, por ejemplo para una imagen de 320x240 pixels, el tamaño de la pantalla de la consola, el nº de bytes que ocupará en memoria es de 320x240 = 76800 bytes = 75 Kb.

**NOTA**: Mientras que en un buffer VGA (en el PC) la ordenación de los píxeles de la imagen es por filas de izquierda a derecha de la imagen desde arriba hacia abajo (origen de coordenadas en la esquina superior izquierda), en la GP32 el origen de coordenadas está en la esquina inferior izquierda, y además el sistema de coordenadas esta girado (es decir, la ordenación es por columnas de abajo a arriba de la imagen desde la izquierda hacia la derecha).

Ahora sólo queda seleccionar la opción “`Convert in a New File`” para indicar al programa el nombre del fichero de cabecera .h dónde se insertarán los dos anteriores arrays (por ejemplo gp32splash.h).

Para posteriormente utilizar esta imagen dentro del programa, sólo habrá que incluir el fichero .h (#include “gp32splash.h”) dentro del módulo en C y hacer uso de estos arrays con las funciones del GPSDK.

Esta utilidad puede encontrarse en la siguiente dirección web:
http://www.ifrance.com/edorul/gp32/

4. **B2FXEC**: Este programa sirve para convertir un programa binario compilado con el compilador GCC en un fichero ejecutable .FXE que es el formato final de los ejecutables que funcionan en la consola GP32.

   El programa en cuestión esta disponible para descargar desde la siguiente dirección web:
   http://www.deadcoderssociety.tk

   La salida de la compilación y el linkado de un programa con el GCC para ARM es normalmente un fichero con extensión .ELF (por ejemplo gpmain.elf).

   Como primer paso debemos convertir el fichero .ELF en un fichero en binario crudo (extensión .GXB ó .BIN es indiferente) con la siguiente linea de comandos (arm-elf-objcopy.exe es una utilidad que acompaña al compilador):

   ```
   arm-elf-objcopy -O binary gpmain.elf gpmain.gxb
   ```

   Una vez tenemos generado el fichero gpmain.gxb, debemos lanzar el siguiente comando para conseguir el ejecutable final .FXE que puede funcionar en la consola:

   ```
   b2fxec -t "MAME GP32" -b "icon.bmp" -a "Franxis" gpmain.gxb gpmain.fxe
   ```

   Donde:

   - *“MAME GP32”* es el nombre del programa, tal cual aparecerá en el menú de selección existente en la propia consola.
   - *“icon.bmp”* es una imagen de 32 x 32 pixels en 8 bit de color y con una paleta de 256 colores estándar que es la utilizada para el menú de la consola. Este ícono aparecerá junto al título del programa en el menú de selección de la consola.
   - *“Franxis”* es el nombre del autor del programa ó un seudónimo.
   - **gpmain.gxb**: es el nombre del fichero binario de entrada.
   - **gpmain.fxe**: es el nombre del fichero ejecutable de salida.

   **NOTA**: El fichero gpmain.fxe, el ejecutable generado esta comprimido con un algoritmo de compresión del tipo PKZIP.

## 2.5. GAMEPARK SDK (GPSDK)

### 2.5.1. INTRODUCCIÓN

El GamePark SDK (GPSDK) es el API (librería de funciones) proporcionada por el fabricante de la consola GP32 (*GamePark*) para su programación. Hay que destacar que estas librerías son gratuitas y están bastante bien explicadas en un documento disponible en inglés y coreano (`gpsdk_api_ref 2.1.0 eng.doc`), aunque con diversas erratas.

La versión usada del **GPSDK** ha sido la **versión 2.1.0**.

En este capítulo se tratará de hacer un repaso a las funciones más interesantes del SDK, junto con algunos ejemplos y consejos aunque para mayor detalle es recomendable leer el citado documento de referencia de *GamePark*.

**NOTA**: No se ha incluido la lista completa de librerías de funciones, se han obviado algunas como la librería de comunicaciones. Tampoco hay un listado completo de las funciones de cada librería, sólo de las más utilizadas.

## 2.5.2. LIBRERÍA ESTÁNDAR (GPSTDLIB)

Nombre de la librería: **lib\libgpstdlib.a**
Fichero de cabecera: **include\gpstdlib.h**

Contiene las funciones de propósito general, algo parecido a la *stdlib* estándar del C. A grandes rasgos, contiene funciones para:

- E/S de los controles (joystick y botones) de la consola
- Control del programa, temporización, velocidad de la consola y de la memoria
- Funciones de reserva / liberación de memoria dinámica
- Funciones de cadenas de caracteres
- Números aleatorios
- Etc.

A continuación se presenta una tabla con las funciones más significativas:

| **Funciones de E/S de joystick y botones** | | | |
|---|---|---|---|
| ***Nombre*** | ***Alias*** | ***Descripción*** | ***Comentarios*** |
| `GpKeyGet()` | N/A | Devuelve el estado de todos los controles de la consola en un entero (int). | La información esta mapeada en los bits del entero. |
| `GpKeyChanged()` | N/A | Permite saber si el estado de los controles ha cambiado desde la anterior consulta. | True (1) o False (0). |
| **Funciones de control del programa** | | | |
| ***Nombre*** | ***Alias*** | ***Descripción*** | ***Comentarios*** |
| `GpAppExecute()` | N/A | Ejecuta un programa desde la SMC. | Permite ejecutar un programa distinto desde el flujo del programa. |
| `GpAppExit()` | N/A | Abandona la ejecución del programa. | **NO FUNCIONA:** Para resetear la consola, mejor: `__asm("swi #4;");` |
| `GpTickCountGet()` | N/A | Consigue un valor en milisegundos del timer interno de la consola (en un entero int). | Este valor no es fiable para frecuencias de reloj superiores a 66 MHz. En este caso multiplicar el valor conseguido por la frecuencia de reloj dividida por 66. |
| `GpClockSpeedChange()` | N/A | Cambia la velocidad de reloj de la consola así como del bus de comunicaciones con la memoria SDRAM. | La velocidad de reloj de la consola puede configurarse de 66 MHz a 133 MHz. También se puede realizar un *overclocking* seguro hasta 166 MHz en la mayoria de las |

| | | | |
|---|---|---|---|
| | | | consolas GP32, y hay opciones de overclocking de hasta 256 MHz.<br><br>La velocidad de la memoria se debe de configurar en concordancia con la velocidad del micro hasta un máximo de 166 MHz. |
| **Funciones de memoria** | | | |
| ***Nombre*** | ***Alias*** | ***Descripción*** | ***Comentarios*** |
| `GPMEMFUNC.malloc()` | gm_malloc() | Reserva de memoria. | Sintaxis similar al malloc() estándar de C.<br><br>**NOTA**: No se recomienda usar las funciones de memoria estándar de C ya que no manejan bien la caché de la MMU. |
| `GPMEMFUNC.zimalloc()` | gm_zi_malloc() | Reserva de memoria inicializada a cero. | Es altamente recomendable usar en lugar de gm_malloc(). |
| `GPMEMFUNC.calloc()` | gm_calloc() | Reserva de memoria para arrays de datos diferentes de bytes (char). | No es recomendable. |
| `GPMEMFUNC.free()` | gm_free() | Liberación de memoria. | Sintaxis similar al free() estándar de C |
| `GPMEMFUNC.memset()` | gm_memset() | Set de memoria a un determinado valor para cada uno de los bytes de la zona de memoria. | Sintaxis similar al memset() estándar de C |
| `GPMEMFUNC.memcpy()` | gm_memcpy () | Copia de memoria. | Sintaxis similar al memcpy() estándar de C |
| `GPMEMFUNC.availablemem()` | N/A | Devuelve la memoria libre disponible. | La GP32 tiene un tamaño total de memoria de 8 Mb, pero el último Mb esta reservado para el SDK, por lo que el total de memoria disponible para la aplicación es de aproximadamente 7 Mb. |

| **Funciones de cadenas de caracteres** | | | |
|---|---|---|---|
| ***Nombre*** | ***Alias*** | ***Descripción*** | ***Comentarios*** |
| `GPSTRFUNC.sprintf()` | gm_sprintf() | Escribe en una cadena de caracteres mediante un formateo. | Similar a sprintf() estándar de C<br><br>**NOTA**: No se recomienda usar las funciones de cadenas estándar de C ya que no manejan bien la caché de la MMU. |
| `GPSTRFUNC.uppercase()` | N/A | Pasa la cadena de caracteres a mayúsculas. | |
| `GPSTRFUNC.lowercase()` | N/A | Pasa la cadena de caracteres a minúsculas | |
| `GPSTRFUNC.compare()` | gm_compare() | Compara dos cadenas de caracteres. | |
| **Funciones de números aleatorios** | | | |
| ***Nombre*** | ***Alias*** | ***Descripción*** | ***Comentarios*** |
| `GpSrand()` | N/A | Especifica semilla de números aleatorios | |
| `GpRand()` | N/A | Consigue número aleatorio. | |

1. **Funciones de entrada / salida de joystick y botones**: Fundamentalmente se usa la función `GpKeyGet()` que devuelve el estado global de todos los controles de la consola. Como ejemplo se muestra el fragmento de código siguiente:

```
int ExKey;
ExKey=GpKeyGet(); /* Consigue el estado de los controles */

if (ExKey & GPC_VK_START) {
/* Se ha pulsado botón START */
}

if (ExKey & GPC_VK_SELECT) {
/* Se ha pulsado botón SELECT */
}

if (ExKey & GPC_VK_FA) {
/* Se ha pulsado botón A */
}

if (ExKey & GPC_VK_FB) {
/* Se ha pulsado botón B */
}

if (ExKey & GPC_VK_FL) {
/* Se ha pulsado botón L */
}

if (ExKey & GPC_VK_FR) {
/* Se ha pulsado botón R */
}

if (ExKey & GPC_VK_UP) {
/* Se ha pulsado joystick ARRIBA */
}

if (ExKey & GPC_VK_DOWN) {
/* Se ha pulsado joystick ABAJO */
}

if (ExKey & GPC_VK_LEFT) {
/* Se ha pulsado joystick IZQUIERDA */
}

if (ExKey & GPC_VK_RIGHT) {
/* Se ha pulsado joystick DERECHA */
}
```

1. **Funciones de entrada control de programa**: De esta colección de funciones, hay que destacar que `GpAppExit()` no funciona correctamente, por lo que es recomendable usar la llamada en ensamblador siguiente para finalizar la aplicación y resetear la consola:

```
__asm("swi #4;");
```

La frecuencia de reloj de la consola se puede ajustar con la función `GpClockSpeedChange()`. A continuación se presentan algunos ejemplos:

```
/* Select Clock Speed */
switch(gp32_freq) {
case 133: /* Clock Speed 133 mhz */
      GpClockSpeedChange(132000000, 0x3a011, 3);
      break;
case 144: /* Clock Speed 144 mhz */
      GpClockSpeedChange(140000000, 0x3e011, 3);
      break;
case 150: /* Clock Speed 150 mhz */
      GpClockSpeedChange(150000000, 0x43011, 3);
      break;
case 156: /* Clock Speed 156 mhz */
      GpClockSpeedChange(156000000, 0x46011, 3);
      break;
case 160: /* Clock Speed 160 mhz */
      GpClockSpeedChange(160000000, 0x48011, 3);
      break;
case 166: /* Clock Speed 166 mhz */
      GpClockSpeedChange(166000000, 0x4b011, 3);
      break;
}
```

La función de `GpTickCountGet()` devuelve un valor en un entero que supuestamente es un valor en milisegundos del reloj interno de la consola, pero que en la práctica no es cierto para frecuencias de reloj de CPU superiores a 66 MHz. Por ello es necesario corregir el valor devuelto por la función, multiplicándolo por la frecuencia de reloj de la consola partido de 66. Ver el extracto de código siguiente:

```
int count1,count2;

count1=GpTickCountGet();
count2=GpTickCountGet();

while ( count2 – count1 < (1000 * (gp32_freq / 66) ) {
/* Espera 1000 ms = 1 segundo */
count2=GpTickCountGet();
}
```

2. **Funciones de memoria**: Aunque las funciones estándar de ANSI C aparentemente funcionan, en la práctica estas no manejan bien la caché de la MMU (*Memory Management Unit*) del procesador ARM que contiene la consola, por lo que es recomendable:

   - Usar `gm_malloc()` en lugar de `malloc()`
   - Es recomendable usar `gm_zi_malloc()` en lugar de `gm_malloc()` ya que de este modo la memoria queda inicializada a cero.
   - Usar `gm_free()` en lugar de `free()`
   - Usar `gm_memset()` en lugar de `memset()`
   - Usar `gm_memcpy()` en lugar de `memcpy()`

3. **Funciones de cadenas de caracteres**: Lo mismo dicho en el anterior punto se puede aplicar a las funciones de cadenas de caracteres:

   - Usar gm_sprintf() en lugar de sprintf().
   - Usar gm_compare() en lugar de strcmp().
   - Usar lowercase() ó uppercase() junto a gm_compare() en lugar de strcasecmp().

4. **Funciones de números aleatorios**: Usar GpSrand() para seleccionar la semilla de números aleatorios y GpRand() para conseguir los números aleatorios.

## 2.5.3. LIBRERÍA DE GRÁFICOS (GPGRAPHIC)

Nombre de la librería: **lib\libgpgraphic.a**
Fichero de cabecera: **include\gpgraphic.h**

Contiene las funciones de gráficos de la consola y de la paleta de colores:

| **Funciones generales** | | | |
|---|---|---|---|
| ***Nombre*** | ***Alias*** | ***Descripción*** | ***Comentarios*** |
| `GpGraphicModeSet()` | N/A | Inicializa el modo gráfico. | Hay dos modos gráficos:<br>- 320x240 píxeles con 256 colores (color de 8 bit).<br>- 320x240 píxeles con 65536 colores (color de 16 bit). |
| `GpLcdEnable()` | N/A | Activa la pantalla LCD de la consola. | Utilizar justo después de inicializar el modo gráfico. |
| `GpLcdSurfaceGet()` | N/A | Consigue un buffer de memoria de vídeo. | Como máximo se permiten:<br>- 4 buffers en modo de color de 8 bit.<br>- 2 buffers en modo de color de 16 bit. |
| `GpSurfaceSet()` | N/A | Activa un buffer de memoria de vídeo (con sincronización, más lento). | Las diferencias de rendimiento no son significativas para frecuencias de reloj superiores a 66 MHz. |
| `GpSurfaceFlip()` | N/A | Cambia a un buffer de memoria de vídeo (sin sincronización, más rápido). | |
| **Funciones de dibujo** | | | |
| ***Nombre*** | ***Alias*** | ***Descripción*** | ***Comentarios*** |
| `GpPointSet()` | N/A | Dibuja un píxel con un determinado color. | |
| `GpRectFill()` | N/A | Pinta un cuadro de píxeles relleno con un determinado color. | |
| `GpLineDraw()` | N/A | Dibuja una línea. | |
| `GpRectDraw()` | N/A | Dibuja un rectángulo. | |
| `GpEllipseDraw()` | N/A | Dibuja una elipse. | |
| **Funciones de *sprites*** | | | |
| ***Nombre*** | ***Alias*** | ***Descripción*** | ***Comentarios*** |
| `GpBitBlt()` | N/A | Pinta un *sprite* en la pantalla. | |
| `GpTransBlt()` | N/A | Pinta el *sprite* con un determinado color que actúa como color transparente (no se dibuja). | |
| `GpBitLRBlt()` | N/A | Dibuja el *sprite* invertido en | |

| | | | |
|---|---|---|---|
| | | horizontal. | |
| `GpTransLRBlt()` | N/A | *Sprite* invertido en horizontal y con transparencia. | |
| `GpBitUDBlt()` | N/A | Dibuja el *sprite* invertido en vertical. | |
| `GpTransUDBlt()` | N/A | Dibuja invertido en vertical y con transparencia. | |
| **Funciones de paleta de colores** | | | |
| ***Nombre*** | ***Alias*** | ***Descripción*** | ***Comentarios*** |
| `GpPaletteEntryChange()` | N/A | Cambia un determinado color de la paleta de colores. | Sólo aplicables al modo gráfico de 8 bit de color (256 colores). |
| `GpPaletteCreate()` | N/A | Crea una paleta de colores. | |
| `GpPaletteSelect()` | N/A | Activa una determinada paleta de colores. | |
| `GpPaletteRealize()` | N/A | Aplica la paleta sobre la pantalla LCD. | |
| `#define GP_RGB24(r,g,b) (((((r>>3))&0x1f)<<11)\|(((g>>3))&0x1f)<<6)\|((((b>>3))&0x1f)<<1))` | N/A | Este código no forma parte del GPSDK, pero es muy interesante. Convierte un valor RGB de 24 bit a su equivalente en la GP32. | El formato de color de la GP32 es `rrrrrggggggbbbbbi` (5 bits de rojo, 5 bits de verde, 5 bits de azul y 1 bit de intensidad de color). |
| **Funciones de efectos gráficos** | | | |
| ***Nombre*** | ***Alias*** | ***Descripción*** | ***Comentarios*** |
| `GpFxBlt()` | N/A | Realiza efectos gráficos sobre una imagen. | |
| `GpLcdFade()` | N/A | Activa efecto de ‘*fading’* sobre la paleta de colores. | |
| `GpLcdNoFade()` | N/A | Desactiva efecto de ‘*fading’* sobre la paleta de colores. | |

1. **Funciones generales:** Para hacer uso de las funciones gráficas de la consola, hay que tener en cuenta sus características:

   La GP32 tiene disponible una única resolución de pantalla: 320 píxeles de resolución horizontal por 240 píxeles de resolución vertical. Eso sí, se puede seleccionar dos profundidades de color distintas:
   - Color de 8 bit (256 colores)
   - Color de 16 bit (65536 colores)

   El siguiente código de ejemplo inicializa el modo gráfico de 8 bit de color:

```
GPDRAWSURFACE gpDraw[2]; /* Buffers de vídeo */
int nflip;               /* Índice de los buffers de vídeo */
int i=0;

/* Activa modo gráfico */
GpGraphicModeSet(8,0);  /* 320x240, 256 colores */

/* Reserva dos buffers de vídeo */
for( i=0; i<2; i++ ) {
GpLcdSurfaceGet( &gpDraw[i],i ); /* Consigue buffer */

GpRectFill( NULL, &gpDraw[i], 0, 0, gpDraw[i].buf_w,
            gpDraw[i].buf_h, 0x0 ); /* Limpia buffer */
}

nflip=0;
GpSurfaceSet( &gpDraw[nflip] ); /* Activa 1er buffer */
GpLcdEnable(); /* Activa pantalla LCD de la consola */
```

   En la consola GP32 se permite acceder a varios *buffers* de vídeo (pantallas de vídeo virtuales en memoria) y cambiar entre ellas a voluntad en cualquier momento. Con ello se pueden aplicar técnicas de doble buffer ó triple buffer de vídeo:
   - En el modo gráfico de 8 bit se permite definir hasta 4 *buffers* de vídeo.
   - En el modo gráfico de 16 bit se permite un máximo de 2 *buffers* de vídeo.

   Con `GpLcdSurfaceGet()` se consiguen los distintos *buffers* de vídeo, y con las siguientes llamadas se puede cambiar a voluntad entre estos buffers:
   - `GpSurfaceSet`(): Con sincronización de vídeo, más lento.
   - `GpSurfaceFlip`(): Sin sincronización de vídeo, más rápido.

   Aun así las diferencias de rendimiento a velocidades de reloj de la consola superiores a 66 MHz son realmente escasas.

   La estrategia consistiría en dibujar cada *frame* de vídeo cada vez en un *buffer* distinto mostrando el *buffer* anteriormente dibujado mientras se esta dibujando el siguiente. Con esto se consigue evitar parpadeos ó imperfecciones en la imagen y también mejorar la visualización gráfica en objetos en movimiento ya que el usuario no ve como se dibuja la pantalla.

2. **Funciones de dibujo:** A las clásicas funciones de dibujo de formas geométricas (punto, recta, elipses, rectángulos, etc.) se añade otra función bastante interesante que dibuja rectángulos rellenos de un color determinado. El siguiente código de ejemplo limpia un *buffer* (pinta un rectángulo que ocupa toda la pantalla y con el color 0 de la paleta de colores) :

```
GpRectFill( NULL, &gpDraw[i], 0, 0, gpDraw[i].buf_w,
            gpDraw[i].buf_h, 0x0 ); /* Limpia buffer */
```

3. **Funciones de *sprites*:** Un *sprite* es una imagen gráfica de menor tamaño que la pantalla y que se utiliza para ser colocado en distintas posiciones de esta, en función de las necesidades del programa. Estos *sprites* se pueden dibujar girados, invertidos, con un determinado color transparente, etc.

   El siguiente código de ejemplo dibuja un *sprite* en un *buffer*:

```
GpBitBlt( NULL,
      &gpDraw[nflip], /* Buffer */
      100,  /* Posición X */
      200, /* Posición Y */
      gpDraw[nflip].buf_w,
      gpDraw[nflip].buf_h,
      misprite, /* El sprite */
      0, 0,
      50, /* ancho del sprite */
      40); /* alto del sprite */
```

   Se puede utilizar por ejemplo la función `GpTransBlt()` en lugar de `GpBitBlt()` si se quiere conseguir que el *sprite* se dibuje con un determinado color transparente. Por ejemplo esto puede ser muy útil para dibujar unos *sprites* encima de otros.

4. **Funciones de paleta de colores:** Estas funciones sólo son necesarias para el modo de color de 8 bit, ya que en el modo de color de 16 bit cada píxel de la imagen ocupa dos bytes (tipo de datos short), el propio valor del píxel identifica su color en función del formato de color de la consola.

   Fundamentalmente usaremos la función `GpPaletteEntryChange()` para cambiar el valor de la paleta para un determinado índice de color (el índice puede tener un valor comprendido entre 0 y 255, son 256 colores).

   El formato de color de la consola es `rrrrrgggggbbbbbi`:
   - `rrrrr`: 5 bits de rojo
   - `ggggg`: 5 bits de verde
   - `bbbbb`: 5 bits de azul
   - `i`: 1 bit de intensidad de color (0 ó 1)

   El siguiente #define nos puede resultar de gran utilidad si pretendemos convertir un color de 24 bit del PC al formato de color usado por la consola GP32:

```
#define GP_RGB24(r,g,b)
(((((r>>3))&0x1f)<<11)|((((g>>3))&0x1f)<<6)|((((b>>3))&0x1f)<<1))
```

Un ejemplo de su utilización es el siguiente:

```
GP_PALETTEENTRY colorgp;
colorgp = GP_RGB24(red,green,blue);
GpPaletteEntryChange(indice,1,&colorgp,GPC_PAL_ASYNC_REALIZE);
```

**NOTA:** Aparte de las funciones del SDK para dibujar sobre un determinado buffer, también se puede acceder directamente al buffer de vídeo y cambiarlo a voluntad a mano. Para ello es necesario acceder al campo `o_buffer` de la estructura `GPDRAWSURFACE` del buffer en cuestión (normalmente `gpDraw[nflip].o_buffer`) que es el array de bytes con la imagen del buffer. Por ejemplo:

```
unsigned char *buffer_scr;
buffer_scr = gpDraw[nflip].o_buffer;
```

Si suponemos una resolución de color de 8 bit (cada píxel de la imagen ocupa 1 byte), para acceder al píxel correspondiente a una determinada posición (x,y) de la imagen, es necesario utilizar la siguiente fórmula:

```
buffer_scr[239-y+x*240]
```

Es necesario realizar esta transformación de coordenadas **(239-y+x*240)** porque la ordenación de los píxeles de la imagen en la pantalla de la GP32 es distinta a la de un búfer VGA de un PC, es decir:

- En el PC el origen de coordenadas (0,0) esta en la esquina superior izquierda de la pantalla, y la ordenación de los píxeles en el *array* de vídeo es por filas desde arriba hasta abajo de la imagen.

- En la GP32 el origen de coordenadas es la esquina inferior izquierda de la pantalla y además la orientación de la pantalla esta girada, es decir la ordenación de los píxeles en el *array* de vídeo es por columnas desde la izquierda hasta la derecha de la imagen.

El siguiente ejemplo convierte un buffer VGA de PC al correspondiente para la consola GP32:

```
for (y = 0; y < 240; y++)
      for (x = 0; x < 320; x++)
            buffer_scr[239-y+x*240] = *scrbitmap++;
```

Con estas técnicas se podrían implementar funciones alternativas a las del SDK para dibujado de *sprites* ó formas geométricas. De hecho las funciones de dibujado del SDK son bastante lentas y fácilmente optimizables.

## 2.5.4. LIBRERÍA DE SONIDO (GPSOUND)

Nombre de la librería: **lib\libgpsound.a**
Fichero de cabecera: **include\gpmm.h**

Contiene las funciones de sonido:

| **Funciones de sonido** | | | |
|---|---|---|---|
| ***Nombre*** | ***Alias*** | ***Descripción*** | ***Comentarios*** |
| `GpPcmInit()` | N/A | Inicializa el acceso al sonido de la consola. | - Frecuencia de 11 a 44 KHz<br>- Estéreo ó monoaural<br>- Resolución de 8 ó 16 bit |
| `GpPcmPlay()` | N/A | Reproduce un sonido. | Como máximo pueden reproducirse 4 sonidos de forma simultanea. |
| `GpPcmLock()` | N/A | Indica posición de fin de reproducción de sonido. | Usar después de `GpPcmPlay()` |
| `GpPcmStop()` | N/A | Para reproducción de un sonido. | |
| `GpPcmRemove()` | N/A | Elimina el sonido de la lista de reproducción de la GP32. | Usar después de `GpPcmStop`(). |

Lo primero que hay que hacer para utilizar las capacidades de sonido de la consola es inicializarlo. A continuación se muestra un código fuente de ejemplo que inicializa el acceso al sonido con características de 22 KHz monaural de 8 bit de resolución:

```
GpPcmInit(PCM_M22,PCM_8BIT);
```

La reproducción de sonido se puede configurar con las siguientes opciones:

- Frecuencia de muestreo: 11 KHz, 22 KHz ó 44 KHz.
- Monaural ó estéreo.
- Resolución de 8 bit ó de 16 bit.

Después de inicializar el sonido, se pueden usar las funciones `GpPcmPlay()` para reproducir un sonido y `GpPcmStop()` para pararlo en uno de los cuatro canales de sonido. En cualquier caso y dado que el SDK mezcla por software los cuatro canales de sonido, nada nos impide mezclar los canales nosotros mismos y sólo utilizar un canal de sonido del SDK para la reproducción.

El formato de sonido utilizado por la consola GP32 es **PCM RAW**.

En monoaural, el formato de sonido en modo 8 bit es ‘*unsigned’* (*array* de bytes unsigned char), mientras que en modo de 16 bit es ‘*signed’* (*array* de bytes signed char). Además hay que tener en cuenta que si se selecciona 16 bit, cada muestra de sonido consta realmente de 2 bytes. En caso de utilizar estéreo, el nº de bytes se duplica obviamente.

Para utilizar un sonido en la GP32 debemos de tenerlo convertido en formato PCM RAW (sólo con los bytes de la muestra de sonido y con el formato correcto en función de la resolución: *signed* en modo 8 bit y *unsigned* en modo 16 bit). El sonido se puede utilizar posteriormente en nuestro programa convirtiendo la muestra RAW a un array de bytes en nuestro código en C. Para ello es conveniente utilizar un programa como el *GP BIN Converter* disponible para descargar desde la web: http://membres.lycos.fr/illusionstudio/ aquafish/dev.htm.

Si queremos tener un *ring buffer* de sonido similar al proporcionado por las librerías SDL, lo mejor es utilizar las librerías *GpSoundBuff* que se pueden conseguir desde la dirección web http://www.gp32.co.nz/snippet.php ? in_sectionid = 3. Es una librería muy sencilla que crea un buffer de sonido circular actualizado mediante una función de interrupción activada mediante *timer*.

**NOTA**: El sonido es uno de los temas más complicados de realizar con la GP32, debido a las limitaciones del propio hardware de la consola, ya que no tiene un verdadero procesador de sonido, sino solo una salida de audio controlada por software. Como todo el procesado de sonido se realiza por software, consume mucho tiempo de CPU (en torno a un 30% de CPU). Además si el código no se optimiza mucho, la calidad del sonido se resiente, ya que podemos saturar el bus de memoria de la consola. En estas circunstancias además la pantalla puede empezar a mostrar líneas negras verticales (la CPU no puede actualizar la pantalla de la consola en tiempo real y vemos como se dibuja la pantalla) y aparecen fluctuaciones en el brillo de la imagen.

## 2.5.5. LIBRERÍA DE FUENTES DE TEXTO (GPFONT)

Nombre de la librería: **lib\libgpfont.a**
Fichero de cabecera: **incluye\gpfont.h**

Contiene las funciones de fuentes de texto y salida de texto por pantalla:

| **Funciones de texto** | | | |
|---|---|---|---|
| ***Nombre*** | ***Alias*** | ***Descripción*** | ***Comentarios*** |
| `GpTextOut()` | N/A | Imprime un texto en la pantalla. | |
| `GpFontInit()` | N/A | Inicializa el acceso a las fuentes de texto. | No usar, realizado automáticamente por el SDK. |
| `GpFontResSet()` | N/A | Registra una fuente de texto de usuario. | |
| `GpEngFontResGet()` | N/A | Consigue una fuente de texto para su manipulación. | |
| `GpSysFontGet()` | N/A | Consigue la fuente de texto activa. | |

Podemos utilizar la función de `GpTextOut()` directamente desde nuestro programa, ya que por defecto esta activada una fuente estándar. Por ejemplo:

```
GpTextOut( NULL,
&gpDraw[nflip] /* Buffer de vídeo */,
pos_x /* Posición X de la pantalla */,
pos_y /* Posición Y de la pantalla*/,
texto /* Texto a mostrar */,
color /* Color */);
```

En el caso de querer utilizar fuentes de texto alternativas, es necesario utilizar el resto de las funciones listadas anteriormente. Toda la información necesaria se encuentra en la documentación del GPSDK.

## 2.5.6. LIBRERÍA DE ENTRADA / SALIDA (GPSTDIO)

Nombre de la librería: **lib\libgpstdio.a**
Fichero de cabecera: **lib\gpstdio.h**

Contiene las funciones de entrada / salida de ficheros del disco (tarjeta de memoria SMC = *SmartMediaCard*):

| **Funciones de entrada / salida** | | | |
|---|---|---|---|
| ***Nombre*** | ***Alias*** | ***Descripción*** | ***Comentarios*** |
| `GpFatInit()` | N/A | Inicializa el acceso a la memoria SMC. | **NOTA**: Es imprescindible usar esta sentencia una única vez al comienzo del programa. |
| `GpRelativePathSet()` | N/A | Especifica una ruta de path relativa. | |
| `GpRelativePathGet()` | N/A | Consigue la ruta de path relativa. | |
| `GpFileOpen()` | N/A | Abre un fichero en la SMC. | Las rutas de directorio de la GP32 son del tipo: `“gp:\\dir1\\dir2\\fichero.ext”` |
| `GpFileRead()` | N/A | Lee un determinado nº de bytes de un fichero. | |
| `GpFileSeek()` | N/A | Busca una posición dentro de un fichero. | La posición dentro del fichero viene indicada con un *offset* en bytes desde el comienzo del fichero. |
| `GpFileClose()` | N/A | Cierra el fichero. | **NOTA**: Es muy recomendable cerrar el fichero después de terminar las operaciones sobre la SMC. |
| `GpFileWrite()` | N/A | Escribe un determinado nº de bytes en un fichero. | **NOTA 1**: Imprescindible usar GpFileClose() después de escribir sobre el fichero, en caso contrario se puede corromper el sistema de ficheros de la tarjeta de memoria.<br><br>**NOTA 2**: Las tarjetas SMC tienen un número máximo de escrituras, tras las cuales dejan de ser fiables (aproximadamente varios miles de operaciones de escritura), por lo tanto se deben limitar al máximo las operaciones de escritura de la tarjeta. |
| `GpFileCreate()` | N/A | Crea un fichero en la SMC. | |
| `GpFileRemove()` | N/A | Borra el fichero de la SMC. | |
| `GpFileGetSize()` | N/A | Consigue el tamaño en bytes del fichero. | |
| `GpFileMove()` | N/A | Mueve un fichero de ubicación en la SMC. | |
| `GpDirCreate()` | N/A | Crea un directorio en la | |

| | | | |
|---|---|---|---|
| | | SMC. | |
| `GpDirRemove()` | N/A | Elimina un directorio de la SMC. | |
| `GpDirEnumNum()` | N/A | Consigue el nº de entradas de fichero y directorio en una determinada ruta de la SMC. | |
| `GpDirEnumList()` | N/A | Consigue una lista de las entradas de fichero y directorio en una determinada ruta de la SMC. | |
| `GpFileAttr()` | N/A | Consigue los atributos de un determinado fichero ó directorio de la SMC. | |
| `GpVolumeInfo()` | N/A | Información general sobre la memoria SMC. | |
| `GpFormat()` | N/A | Formatea una tarjeta de memoria SMC. | **PELIGRO**: El contenido completo de la tarjeta SMC es eliminado tras el formateo. |
| `GpNoFATUpdate()` | N/A | Utilizar justo antes de realizar una operación de escritura en la SMC. | Sirven para acelerar el rendimiento de las operaciones de escritura en la SMC. |
| `GpFATUpdate()` | N/A | Utilizar justo después de realizar la escritura en la SMC. | |

La utilización de las funciones de acceso a disco (memoria SMC) son sencillas de utilizar y están perfectamente explicadas en la documentación del GPSDK. Lo mejor es crear lo que se conoce como un *wrapper* (librería que simula el comportamiento de las funciones estándar de entrada / salida de ficheros de C mediante el GPSDK). Esto es lo que se realizó para la adaptación del MAME.

Lo primero que hay que hacer para acceder a la memoria SMC es llamar a la función `GpFatInit()` del GPSDK. Esto sólo hay que hacerlo una sola vez al comienzo del programa.

El formato de los *path* de ficheros en la GP32 es similar a las de un PC con MS-DOS, por ejemplo `gp:\dir1\dir2\fichero.txt` (nótese que gp: es la unidad de disco de la GP32).
Una vez que se ha inicializado el acceso ya podemos hacer uso del resto de funciones de acceso a disco. Hay que tener mucho cuidado con el acceso a los ficheros de la tarjeta SMC, debemos de cerrar los accesos a los ficheros después de realizar las operaciones sobre ellos utilizando `GpFileClose()`.

En el caso de operaciones de escritura sobre la SMC hay que tener especial cuidado ya que si no cerramos el acceso a los ficheros después de escribirlos podemos llegar a corromper el sistema de ficheros de la SMC.

Además hay que tener en cuenta que las tarjetas de memoria SMC tienen un número máximo de escrituras (varios miles de operaciones de escritura). Si superamos este límite la tarjeta SMC deja de ser fiable. Por tanto hay que extremar las precauciones a la hora de escribir ficheros en la SMC, restringiendo estas operaciones lo máximo posible.

## 2.5.7. LIBRERÍA DEL SISTEMA OPERATIVO (GPOS)

Nombre de la librería: **lib\libgpos.a**
Fichero de cabecera: **lib\gpos_def.h**

Contiene las funciones del sistema operativo interno de la GP32, que son bastantes y que es mejor no usar ya que son internamente utilizadas por el sistema operativo, haré un extracto de las más interesantes accesibles al usuario:

| **Funciones generales** | | | |
|---|---|---|---|
| ***Nombre*** | ***Alias*** | ***Descripción*** | ***Comentarios*** |
| `GpMain()` | N/A | Es la función principal del programa disponible para el usuario. | Es la función equivalente al main() de un programa ANSI C.<br><br>Esta función es la principal que debemos codificar para nuestro programa. |
| `main()` | N/A | Es la función principal real del programa, pero no tenemos acceso a ella, esta previamente codificada y disponible para el programador pero no es recomendable su modificación. | Esta función principal esta disponible en un módulo fuente **`gpstart.c`** que acompaña al GPSDK y que debemos compilar junto con nuestro programa.<br><br>Dentro de este módulo se inicializa el acceso al hardware de la consola y posteriormente se llama a la función **`GpMain()`** que es la función a codificar por el usuario. |
| **Funciones del *timer*** | | | |
| ***Nombre*** | ***Alias*** | ***Descripción*** | ***Comentarios*** |
| `GpTimerOptSet()` | N/A | Especifica una función activada por timer. | Estas funciones sirven para llamar periódicamente a una función (la función se llama por interrupción / evento). |
| `GpTimerSet()` | N/A | Activa una función activada por timer. | |
| `GpTimerKill()` | N/A | Desactiva una función activada por timer. | |
| `GpTimerPause()` | N/A | Pausa una función activada por timer. | |
| `GpTimerResume()` | N/A | Reactiva una función pausada. | |

De toda la colección de funciones de la librería GPOS, cabe detenerse especialmente en `main()` y `GpMain()`:

- **`main()`** es la función principal del programa, pero no tenemos acceso a ella, ya que viene ya escrita en un fichero `gpstart.c` que acompaña al GPSDK y no es recomendable su modificación. Tan sólo debemos linkar nuestro programa junto con esta librería.

- **`GpMain()`** es la función principal del usuario, y es la que debemos codificar para nuestro programa.

Lógicamente la función `main()` inicializa el acceso al hardware de la consola GP32 y posteriormente llama a `GpMain()`, que es la función que debemos codificar.

El resto de funciones de la librería GPOS son internamente utilizadas por main() y no debemos utilizarlas desde nuestro programa.

Las únicas funciones adicionales que tienen interés para nosotros son las de acceso al *timer* de la consola GP32: `GpTimerOptSet()`, `GpTimerSet()`, `GpTimerKill()`, `GpTimerPause()`, `GpTimerResume()`. Con estas podemos conseguir llamadas periódicas a una función creada por nosotros. Para ello se utiliza el *timer* de la consola y hay que tener en cuenta que estas funciones se ejecutan por interrupción / evento. Estas funciones del *timer* son especialmente útiles por ejemplo para desarrollar un *ring buffer* del estilo del *GpSoundBuff* del que se ha hablado antes en la parte del sonido GPSOUND.

## 2.6. MAME 0.34 PARA MS-DOS

### 2.6.1. INTRODUCCIÓN

Como se ha comentado en capítulos anteriores, MAME son las siglas *Multiple Arcade Machine Emulator*, el emulador de máquinas recreativas. En este capítulo se analizará el código fuente original de la versión 0.34 para MS-DOS.

Esta organizado en 830 módulos fuente junto con un único fichero `Makefile` en el raíz de la distribución, así como distintos directorios dentro de la carpeta “`src`” del proyecto para organizar los módulos:

- **`Makefile`**: Es el fichero necesario para compilar el emulador.
- **`src/`**: Dentro del raíz de la carpeta “`src`” se encuentran los módulos principales del emulador, entre ellos un fichero “`drivers.c`” que contiene la lista completa de drivers de juegos soportados por el emulador.
- **`src/drivers/`**: Contiene los distintos drivers para hacer funcionar cada uno de los juegos soportados.
- **`src/msdos/`**: En esta carpeta se encuentran las librerías específicas de la versión original para MS-DOS.
- **`src/I86/`**: Emulador de *Intel* 8086.
- **`src/I8039/`**: Emulador de *Intel* 8039.
- **`src/M6502/`**: Emulador de *Motorola* 6502.
- **`src/M6805/`**: Emulador de *Motorola* 6805.
- **`src/M6808/`**: Emulador de *Motorola* 6808.
- **`src/M6809/`**: Emulador de *Motorola* 6809.
- **`src/M68000/`**: Emulador de *Motorola* 68000.
- **`src/machine/`**: Contiene los distintos entornos hardware para emular cada una de las familias de juegos soportados.
- **`src/S2650/`**: Emulador de *Signetics* 2650.
- **`src/sndhrdw/`**: Contiene los distintos emuladores de procesadores de sonido.
- **`src/T11/`**: Emulador de DEC T-11.
- **`src/TMS9900/`**: Emulador de *Texas Instruments* TMS9900.
- **`src/TMS34010/`**: Emulador de *Texas Instruments* TMS34010.
- **`src/vidhrdw/`**: Contiene los distintos emuladores de procesadores gráficos.
- **`src/Z80/`**: Emulador del procesador Z80.

En principio se supone que sólo habría que re-escribir las librerías de **`src/msdos`** para portar el MAME 0.34 a otra arquitectura diferente como la GP32.

## 2.6.2. DESCRIPCIÓN DE LOS MÓDULOS FUENTE

A continuación se muestran una lista de los módulos más interesantes del MAME, así como su descripción y comentarios sobre la labor necesaria (estimada a priori) para portarlos a GP32:

| *Carpeta* | *Fichero* | *Descripción* | *Comentarios* |
|---|---|---|---|
| ./ | Makefile | Fichero necesario para compilar el MAME en MS-DOS. | Será necesario adaptarlo para generar el ejecutable en GP32. |
| src/drivers/ | *246 módulos:*<br>1942.c<br>1943.c<br>8080bw.c<br>aeroboto.c<br>amidar.c<br>arabian.c<br>arkanoid2.c<br>arkanoid.c<br>asteroid.c<br>etc… | Contiene la lista completa de drivers de juegos soportados por el emulador (246 ficheros). | Será necesario eliminar los drivers de juegos que no es posible hacer funcionar en la GP32 por limitaciones de memoria ó resolución de pantalla. |
| src/I86/ | I86.c<br>I86.h<br>I86dasm.c<br>I86intrf.h<br>instr.h<br>modrm.h<br>ea.h<br>host.h | Contiene el emulador de Intel 8086. | Habrá que quitar parte del código para ahorrar memoria (por ejemplo el desensamblador). |
| src/I8039/ | I8039.c<br>I8039.h<br>8039dasm.c<br>dis48.c | Contiene el emulador de Intel 8039 | Quitar desensamblador para ahorrar memoria. |
| src/I8085/ | I8085.c<br>I8085.h<br>I8085CPU.h<br>I8085DAA.h<br>8085DASM.c | Contiene el emulador de Intel 8085. | Quitar desensamblador. |
| src/M6502/ | 6502dasm.c<br>m6502.c<br>m6502.h<br>m6502ops.h<br>tbl65c02.c<br>tbl6502.c<br>tbl6510.c | Emulador de Motorola 6502. | Quitar desensamblador. |
| src/M6805/ | 6805dasm.c<br>6805ops.c<br>m6805.c<br>m6805.h | Emulador de Motorota 6805. | Quitar desensamblador. |
| src/M6808/ | 6808dasm.c<br>6808ops.c<br>m6808.c<br>m6808.h | Emulador de Motorola 6808. | Quitar desensamblador. |

| *Carpeta* | *Fichero* | *Descripción* | *Comentarios* |
|---|---|---|---|
| | make6808.c | | |
| src/M6809/ | 6809dasm.c<br>6809ops.c<br>m6809.c<br>m6809.h | Emulador de Motorola 6809. | Quitar desensamblador. |
| src/M68000/ | Asmintf.c<br>cpudefs.h<br>cpufunc.c<br>cputbl.h<br>cycletbl.h<br>dis68k.c<br>m68kdasm.c<br>m68000.h<br>make68k.c<br>mc68kmem.c<br>readcpu.h<br>opcode0.c<br>opcode1.c<br>opcode2.c<br>opcode3.c<br>opcode4.c<br>opcode5.c<br>opcode6.c<br>opcode7.c<br>opcode8.c<br>opcode9.c<br>opcodeb.c<br>opcodec.c<br>opcoded.c<br>opcodee.c | Emulador de Motorola 68000. | Quitar desensamblador.<br><br>Y además ya que el emulador ocupa 1.12 Mb de código fuente (unas 2 Mb en memoria), habrá que eliminar de él todo lo que no sea indispensable para el port a GP32 (ya que la GP32 sólo tiene 8 Mb de memoria): optimizaciones en ensamblador para 80x86, código de de depuración, etc |
| src/machine/ | *96 módulos:*<br>6821pia.c<br>6821pia.h<br>8080bw.c<br>arabian.c<br>arkanoid.c<br>asteroid.c<br>etc… | Contiene los distintos entornos hardware para emular cada una de las familias de juegos soportados. | Eliminar los entornos de los juegos que no pueden funcionar en la GP32 por limitaciones de memoria. |
| src/msdos/ | config.c | Selección de opciones del emulador a partir de los argumentos de entrada de línea de comandos de MS-DOS. | Re-escribir para que la selección de opciones se realice desde un menú de configuración dentro del port a GP32. |
| | fileio.c | Entrada / Salida de ficheros de disco. | Re-escribir para leer de la tarjeta SMC (*Smart Media Card*) de la consola GP32. |
| | fronthlp.c | Módulo auxiliar para comprobar las ROMs. | A eliminar. |

| *Carpeta* | *Fichero* | *Descripción* | *Comentarios* |
|---|---|---|---|
| | input.c | Lectura del teclado y del joystick. | Re-escribir para leer los controles de la GP32. |
| | msdos.c | Función main() principal del MAME. | Adaptar para GP32. |
| | osd_cpu.h | Macros y definiciones de datos para Intel 80386 ó superior. | Modificarlas para la arquitectura de la consola GP32. |
| | osinline.h | Optimización de multiplicación de vectores para 80386. | A eliminar, no aplicable a GP32. |
| | profiler.c | Librería para medir el rendimiento del emulador. | A eliminar. |
| | sound.c | Librería de reproducción de audio usando las librerías *Allegro* para MS-DOS. | Re-escribir para GP32. |
| | twkuser.c | Librería para la inicialización de los distintos modos gráficos en el PC. | A eliminar, no es aplicable a GP32. |
| | twkuser.h | Fichero de cabecera del módulo anterior. | A eliminar. |
| | vector.c | Librería de gráficos vectoriales. | A eliminar. |
| | vgafreq.h | Definiciones de frecuencias de refresco de modos gráficos en el PC. | A eliminar. |
| | vídeo.c | Librería de salida de vídeo, manejo de la paleta de colores y reserva de memoria para bitmaps y buffers de vídeo. | Re-escribir para la consola GP32. |
| | ym2203.h | Librería de reproducción de música FM YM2203 a través del chip OPL de la tarjeta Sound Blaster del PC. | A eliminar. |
| src/S2650/ | 2650dasm.c<br>s2650.c<br>s2650.h<br>s2650cpu.h | Emulador de *Signetics* 2650. | Eliminar desensamblador, código de depuración, etc. |
| src/sndhrdw/ | *97 módulos:*<br>2151intf.c<br>2151intf.h<br>2203intf.c | Contiene los distintos emuladores de procesadores de sonido. | Eliminar los emuladores no usados por los juegos a emular en GP32. |

| *Carpeta* | *Fichero* | *Descripción* | *Comentarios* |
|---|---|---|---|
| | 2413intf.h<br>2610intf.c<br>etc… | | |
| src/T11/ | t11.c<br>t11.h<br>t11dasm.c<br>t11ops.c<br>t11table.c | Emulador de DEC T-11. | Quitar desensamblador, código de depuración, etc. |
| src/TMS9900/ | 9900dasm.c<br>tms9900.c<br>tms9900.h<br>tms9900c.h | Emulador de *Texas Instruments* TMS9900. | A eliminar ya que los juegos que utilizan este procesador no van a funcionar en GP32 por limitaciones de memoria de la consola. |
| src/TMS34010/ | 34010dsm.c<br>34010fld.c<br>34010ops.c<br>34010ops.h<br>34010tbl.c<br>dis34010.c<br>tms34010.c<br>tms34010.h | Emulador de *Texas Instruments* TMS34010. | A eliminar ya que los juegos que utilizan este procesador no van a funcionar en GP32 por limitaciones de memoria de la consola. |
| src/vidhrdw/ | *239 módulos :*<br>1942.c<br>1943.c<br>8080bw.c<br>aeroboto.c<br>amidar.c<br>arabian.c<br>arkanoid.c<br>etc... | Contiene los distintos emuladores de procesadores gráficos. | Habrá que quitar los emuladores de procesadores gráficos de los juegos no emulados por el port a GP32 (debido a limitaciones de memoria). |
| src/Z80/ | Z80.c<br>Z80.h<br>Z80CDx86.h<br>Z80Codes.h<br>Z80DAA.h<br>Z80Dasm.c<br>Z80Dasm.h<br>Z80Debug.c<br>Z80io.h | Emulador del procesador Z80. | Quitar desensamblador, eliminar parte en ensamblador para 80x86, quitar todo el código de depuración para ahorrar memoria, etc. |
| src/ | artwork.c<br>artwork.h | Librería de soporte para artwork. | A eliminar. |
| src/ | asg.c | Funciones de debug. | A eliminar. |
| src/ | audit.c<br>audit.h | Librería para auditar las ROMs. | A eliminar. |
| src/ | cheat.c<br>cheat.h | Librería de soporte de trucos de juegos. | A eliminar. |
| src/ | common.c<br>common.h | Funciones de propósito general. | Portar y simplificar todo lo posible. |
| src/ | cpuintrf.c<br>cpuintrf.h | Librería de interfaces con los distintos emuladores de | Portar, simplificar todo lo posible y eliminar los |

| *Carpeta* | *Fichero* | *Descripción* | *Comentarios* |
|---|---|---|---|
| | | procesadores. | procesadores no usados. |
| src/ | crc32.c | Librería de manejo de CRC de 32 bits. | Eliminar el módulo, pasar la función de cálculo de CRC a otro módulo. |
| src/ | driver.c<br>driver.h | Definición y listado de todos los drivers de juegos emulados por el MAME. | Por limitaciones de memoria de la GP32 será necesario crear distintos ficheros driver.c para distintas colecciones de juegos. Habrá que generar múltiples ficheros ejecutables, cada uno con soporte para distintos juegos. |
| src/ | inflate.c<br>inflate.h | Librería para descomprimir ficheros ZIP. | Adaptar y simplificar todo lo posible. |
| src/ | info.c<br>info.h | Informes en pantalla del MAME sobre los drivers. | A eliminar. |
| src/ | inptport.c<br>inptport.h | Lectura y escritura de puertos, así como lectura de los controles, etc. | Adaptar para la GP32. |
| src/ | mame.c<br>mame.h | Módulo principal de emulación. | Portar y simplificar todo lo posible. |
| src/ | mamedbg.c<br>mamedbg.h<br>os_dbg.h | Librería de debug. | Eliminar. |
| src/ | memory.c<br>memory.h | Librería de entrada / salida en memoria de los juegos. | Portar y simplificar todo lo posible. |
| src/ | palette.c<br>palette.h | Librería de manejo de la paleta de colores de los juegos. | Portar y simplificar todo lo posible. |
| src/ | png.c | Lectura de ficheros de imágenes .PNG. | Eliminar. |
| src/ | romcmp.c | Utilidad externa de chequeo de ROMs. | Eliminar. |
| src/ | timer.c<br>timer.h | Librería de funciones de sincronización entre distintos procesadores gráficos, de vídeo y sonido. | Portar y simplificar todo lo posible. |
| src/ | types.h | Definición de tipos usados en el MAME. | Adaptar de MS-DOS (80x86) a GP32 (ARM). |
| src/ | unzip.c | Librería de | Simplificar todo lo |

| *Carpeta* | *Fichero* | *Descripción* | *Comentarios* |
|---|---|---|---|
| | `unzip.h` | tratamiento de ficheros comprimidos .ZIP. | posible. |
| `src/` | `usrintrf.c`<br>`usrintrf.h` | Entorno de usuario en pantalla del MAME. | Eliminar por completo, para ganar memoria, se utilizarán en su lugar las funcione del SDK de la GP32 para el entorno gráfico. |

## 2.6.3. FLUJOGRAMAS DE EJECUCIÓN DEL MAME

A continuación se muestra una figura con el flujo de ejecución del MAME, a nivel muy general:

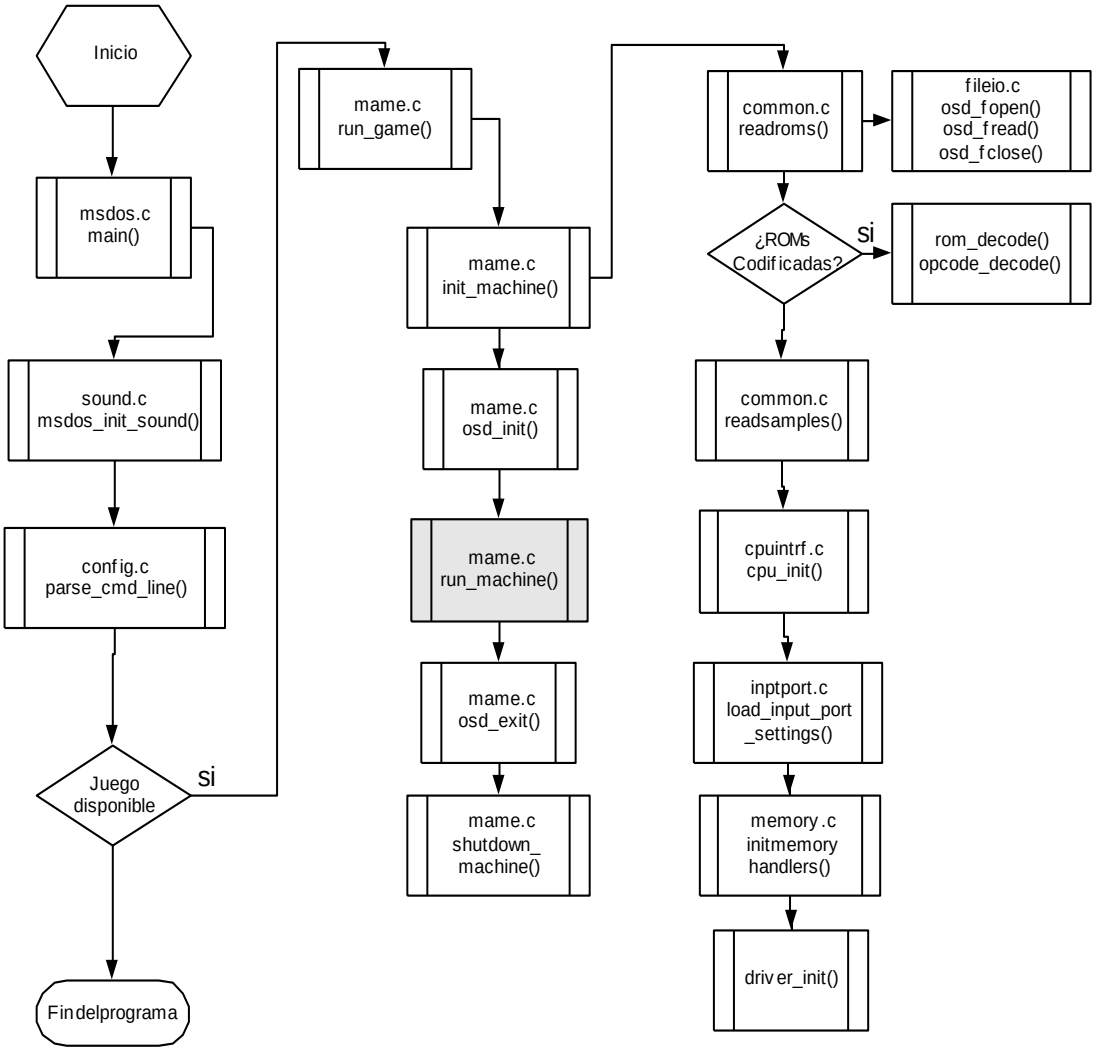



**Figura 31: Flujograma general de ejecución del MAME**

En la función **`main()`** principal del programa:

- Primero inicializa el acceso al sonido a través de las librerías Allegro de MS-DOS mediante **`msdos_init_sound().`**
- Después el programa toma de los argumentos de línea de órdenes del MS-DOS el juego requerido con **`parse_cmd_line().`** Si no existe tal argumento muestra un mensaje de error y termina.
- Si todo está aparentemente OK ejecuta la función **`run_game()`** de `mame.c`.

La función **`run_game()`** sigue los siguientes pasos:

- Llama a la función **`init_machine()`** de *mame.c* que realiza las tareas previas antes de comenzar la emulación del juego seleccionado:
    - Con la función **`readroms()`** de *common.c* lee las ROMs del juego seleccionado. Para ello utiliza las funciones de fileio.c.
    - Si las ROMs están encriptadas utiliza las funciones de decodificación específicas del juego, que vienen en los módulos del juego en cuestión (generalmente con nombre **`rom_decode()`** y/o **`opcode_decode()`**).
    - Si el juego requiere de muestras de sonido para funcionar, las lee con la función **`read_samples().`**
    - Se procede a inicializar cada uno de los procesadores necesarios para ejecutar el juego. Para ello se llama a la función **`cpu_init()`** de *cpuintrf.c*. Esta función genérica lee las estructuras del driver del juego e inicializa los emuladores de procesadores necesarios.
    - Con **`load_input_ports_settings()`** de *inptport.c* se inicializa el acceso a los controles necesarios para manejar el juego (teclas del PC, joystick digital, analógico ó ratón dependiendo del juego en cuestión).
    - Con **`init_memory_handlers()`** de *memory.c* se inicializan las funciones necesarias para manejar la memoria y los puertos de entrada / salida del juego.
    - **`driver_init().`** Es una función específica del juego a emular, y está ubicada en uno de los módulos propios del juego a emular.
- La función **`osd_init()`** de *mame.c* realiza la inicialización del entorno gráfico del emulador.
- La función **`run_machine()`** de *mame.c* es la que propiamente realiza la emulación del juego, y no termina hasta que el usuario pulsa la tecla ESC para salir del programa.
- Una vez que el usuario requiere salir del programa, se llama a las funciones **`osd_exit()`** y **`shutdown_machine()`** de *mame.c* antes de terminar, para liberar toda la memoria dinámica reservada por el programa.

A continuación se muestra el flujograma de la función **`run_machine():`**

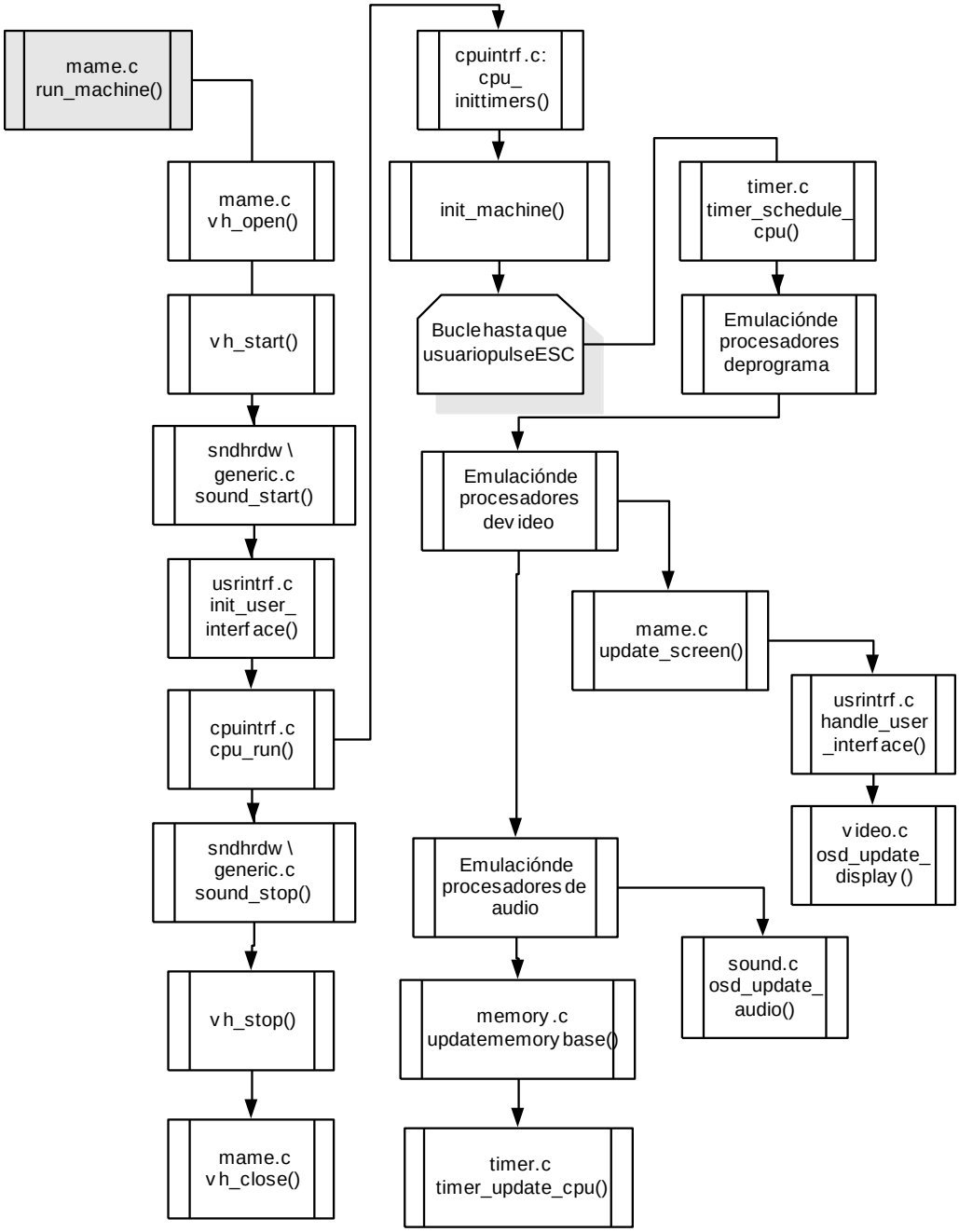



**Figura 32: Flujograma de la función run_machine()**

La función **`run_machine()`** parte con las *ROMs* del juego a emular ya cargadas y se dispone a comenzar la emulación. Para ello sigue los siguientes pasos:

- Llama a **`vh_open()`** de *mame.c* para inicializar el acceso a las funciones genéricas de vídeo y a la función específica del *driver* **`vh_start()`** para inicializar el acceso a las funciones de los procesadores de vídeo específicos del *driver*.
- Inicializa el acceso a las funciones de acceso a los procesadores de sonido, con la función **`sound_start()`** del módulo *sndhrdw\generic.c.*
- Inicializa el interfaz de usuario del emulador, con **`init_user_interface()`** de *usrintrf.c.*
- Ejecuta la función **`cpu_run()`** de *cpuintrf.c*, que es la que llevará a cabo la emulación propiamente dicha. En esta función permanecerá el emulador funcionando hasta que el usuario pulse ESC:
    - Inicializa el control de la temporización del emulador, que servirá para hacer funcionar virtualmente en paralelo los distintos procesadores del juego a emular: **`cpu_inittimers()`** de *cpuintrf.c*
    - Inicializa la máquina virtual, con la función **`init_machine()`** específica del *driver*.
    - Mientras que el usuario no pulse la tecla ESC se realiza el bucle general de la emulación:
        - Calcula los ciclos a emular de cada uno de los procesadores y planifica la temporización con la función **`timer_schedule_cpu()`** de *timer.c.*
        - Internamente se llama a las funciones específicas para emular los distintos procesadores de programa de vídeo y de audio.
        - **`update_screen()`** de *mame.c* lleva a cabo la actualización de la pantalla, que internamente llama a las funciones:
            - **`handle_user_interface()`** de *usrintrf.c*: Esta función comprueba las pulsaciones de teclado del usuario (reset del emulador, pausa, acceso a los menús del entorno gráfico, controles, joystick, etc).
            - **`osd_update_screen()`** de *vídeo.c*: Esta función de bajo nivel específica del port a MS-DOS es la que realmente actualiza la pantalla.
        - Para actualizar el audio se utilizan las funciones específicas del port a MS-DOS de *sound.c*, especialmente **`osd_update_audio().`**
        - **`updatememorybase()`** de *memory.c* realiza la actualización de la memoria de la máquina virtual.
        - **`timer_update_cpu()`** de *timer.c* realiza la actualización de los temporizadores después de los ciclos emulados.
- Ejecuta la función del *driver* de vídeo **`vh_stop()`** y la función genérica **`vh_close()`** de *mame.c.*

## 2.6.4. DRIVERS DE JUEGOS

En el punto anterior hemos visto como el emulador en función del juego seleccionado por el usuario, inicializa una máquina virtual y permite emularlo como si estuviéramos delante de la máquina recreativa original. Pero ¿qué necesita el emulador para emular cada uno de los juegos? A grandes rasgos es lo siguiente:

- **El propio driver del juego:** Se trata de una serie de estructuras dentro de un módulo en el directorio **`src/drivers/`**, con la siguiente información:
    - Definición del *driver*, junto con la declaración de las funciones específicas del *driver*.
    - Definición de la máquina virtual donde funciona el juego: Procesadores de programa, de gráficos y de sonido.
    - Definición de las *ROMs* necesarias para hacer funcionar el juego. Con el nombre de las *ROMs*, tamaño, direcciones de memoria asociadas y CRC.
    - Definición de los puertos de lectura de memoria: Direcciones de memoria, tipo de acceso, etc.
    - Definición de los puertos de escritura de memoria: Direcciones, modo de acceso, etc.
    - Puertos de entrada / salida.
    - Definición de los *dip-switches* y controles de la recreativa.
    - Definición de los gráficos y de los *sprites*.

- **Las funciones específicas de la máquina virtual:** Se encuentran en un módulo dentro del directorio **`src/machine/.`**

- **Las funciones de los procesadores de sonido:** Ubicados en un módulo dentro del directorio **`src/sndhrdw/.`**

- **Las funciones de los procesadores de vídeo:** Ubicadas en un módulo dentro del directorio **`src/vidhdrdw/.`**

## 2.6.5. EJEMPLO DE UN DRIVER: PAC-MAN

La versión original del MAME para MS-DOS carece de entorno gráfico, es necesario lanzar desde la línea de comandos el ejecutable del emulador (mame.exe) seguido del identificador del juego. Por ejemplo:

```
mame pacman
```

La anterior línea lanza la emulación del juego Pac-Man. Para que el juego funcione, será necesario tener dentro de la carpeta ROMS/ del emulador un fichero pacman.zip ó una sub-carpeta ROMS/PACMAN/ con los siguientes ficheros (las ROMs del juego en cuestión):

| ***Nombre de la ROM*** | ***Tamaño*** | ***Descripción*** |
|---|---|---|
| 82s123.7f | 32 bytes | PROM con la paleta de colores |
| 82s126.4a | 256 bytes | PROM con la definición de colores |
| 82s126.1m | 256 bytes | PROM con las muestras de sonido |
| 82s126.3m | 256 bytes | PROM de la temporización del sonido |
| pacman.5e | 4096 bytes (4 Kb) | ROM de vídeo (1 de 2) |
| pacman.5f | 4096 bytes (4 Kb) | ROM de vídeo (2 de 2) |
| pacman.6e | 4096 bytes (4 Kb) | ROM de programa (1 de 4) |
| pacman.6f | 4096 bytes (4 Kb) | ROM de programa (2 de 4) |
| pacman.6h | 4096 bytes (4 Kb) | ROM de programa (3 de 4) |
| pacman.6j | 4096 bytes (4 Kb) | ROM de programa (4 de 4) |

Lista de ROMs del Pac-Man

Las características técnicas del juego son las siguientes:

- Lanzado en 1980, realizado por la empresa *Namco* con licencia de la empresa *Midway*.
- Emulador original por Allard Van der Bas. Integrado en MAME por Nicola Salmoria.
- 16 Kb de ROM de programa.
- 1 Kb de RAM de vídeo.
- 1 Kb de RAM de paleta de colores.
- 1 Kb de memoria RAM.
- Juego funciona sobre un procesador Z80 de 3.072 MHz.
- Resolución de pantalla: 228 píxeles de resolución horizontal por 288 píxeles de resolución vertical (el mismo procesador gráfico que el de la recreativa *Pengo* de *Sega*).
- Paleta de 16 colores simultáneos en pantalla.
- Procesador de sonido de *Namco*: 3 canales de sonido con *samples* de 8 bytes de longitud.

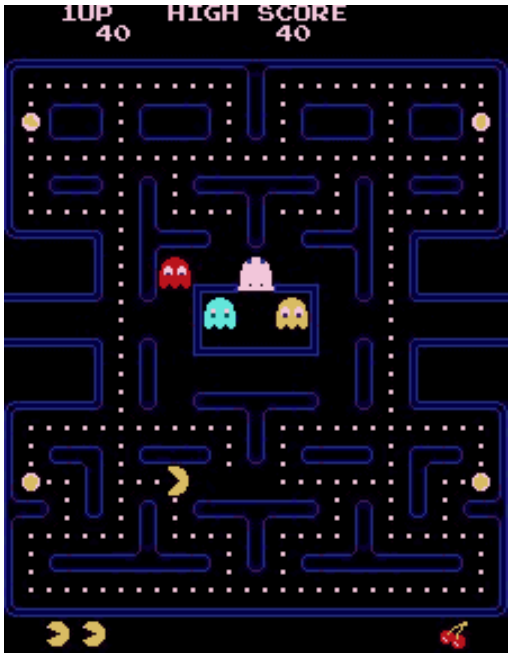



**Figura 33: Pantalla del Pac Man**

1) La definición del *driver* está ubicada en el fichero **`src/drivers/pacman.c`**, y consta de las siguientes estructuras:

Definición final del driver:

```
struct GameDriver pacman_driver =
{
__FILE__,
0,
"pacman",
"Pac Man (Midway)",
"1980",
"[Namco] (Midway license)",
BASE_CREDITS,
0,
&machine_driver,
0,

pacman_rom,
0, 0,
0,
0,    /* sound_prom */

pacman_input_ports,

PROM_MEMORY_REGION(2), 0, 0,
ORIENTATION_ROTATE_90,

0, 0
};
```

La definición de la máquina virtual dónde corre el juego:

```
static struct MachineDriver machine_driver =
{
/* basic machine hardware */
{
        {
                CPU_Z80,
                18432000/6,     /* 3.072 Mhz */
                0,
                readmem,writemem,0,writeport,
                pacman_interrupt,1
        }
},
60, 2500,       /* frames per second, vblank duration */
1,      /* single CPU, no need for interleaving */
pacman_init_machine,

/* vídeo hardware */
36*8, 28*8, { 0*8, 36*8-1, 0*8, 28*8-1 },
gfxdecodeinfo,
16, 4*32,
pacman_vh_convert_color_prom,

VÍDEO_TYPE_RASTER | VÍDEO_SUPPORTS_DIRTY,
0,
pacman_vh_start,
generic_vh_stop,
```

```
pengo_vh_screenrefresh,

/* sound hardware */
0,0,0,0,
{
	{
		SOUND_NAMCO,
		&namco_interface
	}
}
};
```

La definición de las ROMs necesarias para el juego:

```
ROM_START( pacman_rom )
ROM_REGION(0x10000)     /* 64k for code */
ROM_LOAD( "pacman.6e",    0x0000, 0x1000, 0xc1e6ab10 )
ROM_LOAD( "pacman.6f",    0x1000, 0x1000, 0x1a6fb2d4 )
ROM_LOAD( "pacman.6h",    0x2000, 0x1000, 0xbcdd1beb )
ROM_LOAD( "pacman.6j",    0x3000, 0x1000, 0x817d94e3 )

ROM_REGION_DISPOSE(0x2000)    /* temporary space for graphics (disposed
after conversion) */
ROM_LOAD( "pacman.5e",    0x0000, 0x1000, 0x0c944964 )
ROM_LOAD( "pacman.5f",    0x1000, 0x1000, 0x958fedf9 )

ROM_REGION(0x0120)      /* color PROMs */
ROM_LOAD( "82s123.7f",    0x0000, 0x0020, 0x2fc650bd )
ROM_LOAD( "82s126.4a",    0x0020, 0x0100, 0x3eb3a8e4 )

ROM_REGION(0x0200)      /* sound PROMs */
ROM_LOAD( "82s126.1m",    0x0000, 0x0100, 0xa9cc86bf )
ROM_LOAD( "82s126.3m",    0x0100, 0x0100, 0x77245b66 )     /*  timing  -
not used */
ROM_END
```

La definición de los puertos de lectura de memoria:

```
static struct MemoryReadAddress readmem[] =
{
{ 0x0000, 0x3fff, MRA_ROM },
{ 0x4000, 0x47ff, MRA_RAM },  /* vídeo and color RAM */
{ 0x4c00, 0x4fff, MRA_RAM },  /* sprites at 4ff0-4fff */
{ 0x5000, 0x503f, input_port_0_r }, /* IN0 */
{ 0x5040, 0x507f, input_port_1_r }, /* IN1 */
{ 0x5080, 0x50bf, input_port_2_r }, /* DSW1 */
{ 0x50c0, 0x50ff, input_port_3_r }, /* DSW2 */
{ 0x8000, 0xbfff, MRA_ROM },  /* Ms.PacMan/Ponpoko only */
{ -1 }      /* end of table */
};
```

La definición de los puertos de escritura de memoria:

```
static struct MemoryWriteAddress writemem[] =
{
{ 0x0000, 0x3fff, MWA_ROM },
{ 0x4000, 0x43ff, vídeoram_w, &vídeoram, &vídeoram_size },
{ 0x4400, 0x47ff, colorram_w, &colorram },
{ 0x4c00, 0x4fef, MWA_RAM },
```

```
{ 0x4ff0, 0x4fff, MWA_RAM, &spriteram, &spriteram_size },
{ 0x5000, 0x5000, interrupt_enable_w },
{ 0x5001, 0x5001, pengo_sound_enable_w },
{ 0x5002, 0x5002, MWA_NOP },
{ 0x5003, 0x5003, pengo_flipscreen_w },
      { 0x5004, 0x5005, osd_led_w },
      { 0x5006, 0x5006, MWA_NOP },
      { 0x5007, 0x5007, coin_counter_w },
{ 0x5040, 0x505f, pengo_sound_w, &pengo_soundregs },
{ 0x5060, 0x506f, MWA_RAM, &spriteram_2 },
{ 0x50c0, 0x50c0, watchdog_reset_w },
{ 0x8000, 0xbfff, MWA_ROM },  /* Ms.PacMan/Ponpoko only */
{ 0xc000, 0xc3ff, vídeoram_w }, /* mirror vídeo ram, */
{ 0xc400, 0xc7ef, colorram_w }, /* HIGH SCORE & CREDITS */
{ 0xffff, 0xffff, MWA_NOP },  /* Eyes writes location */
{ -1 }      /* end of table */
};
```

Los puertos de entrada / salida:

```
static struct IOWritePort writeport[] =
{
{ 0, 0, interrupt_vector_w }, /* Pac Man only */
{ -1 }      /* end of table */
};
```

La definición de los *dip-switches* y controles de la recreativa:

```
INPUT_PORTS_START( pacman_input_ports )
PORT_START  /* IN0 */
PORT_BIT( 0x01, IP_ACTIVE_LOW, IPT_JOYSTICK_UP | IPF_4WAY )
PORT_BIT( 0x02, IP_ACTIVE_LOW, IPT_JOYSTICK_LEFT | IPF_4WAY )
PORT_BIT( 0x04, IP_ACTIVE_LOW, IPT_JOYSTICK_RIGHT | IPF_4WAY )
PORT_BIT( 0x08, IP_ACTIVE_LOW, IPT_JOYSTICK_DOWN | IPF_4WAY )
PORT_BITX(     0x10, 0x10, IPT_DIPSWITCH_NAME | IPF_CHEAT, "Rack Test",
OSD_KEY_F1, IP_JOY_NONE, 0 )
PORT_DIPSETTING(    0x10, "Off" )
PORT_DIPSETTING(    0x00, "On" )
PORT_BIT( 0x20, IP_ACTIVE_LOW, IPT_COIN1 )
PORT_BIT( 0x40, IP_ACTIVE_LOW, IPT_COIN2 )
PORT_BIT( 0x80, IP_ACTIVE_LOW, IPT_COIN3 )

PORT_START  /* IN1 */
PORT_BIT( 0x01, IP_ACTIVE_LOW, IPT_JOYSTICK_UP | IPF_4WAY | IPF_COCKTAIL
)
PORT_BIT(   0x02,   IP_ACTIVE_LOW,   IPT_JOYSTICK_LEFT   |   IPF_4WAY   |
IPF_COCKTAIL )
PORT_BIT(   0x04,   IP_ACTIVE_LOW,   IPT_JOYSTICK_RIGHT   |   IPF_4WAY   |
IPF_COCKTAIL )
PORT_BIT(   0x08,   IP_ACTIVE_LOW,   IPT_JOYSTICK_DOWN   |   IPF_4WAY   |
IPF_COCKTAIL )
PORT_BITX(      0x10,  0x10,  IPT_DIPSWITCH_NAME  |  IPF_TOGGLE,  "Service
Mode", OSD_KEY_F2, IP_JOY_NONE, 0 )
PORT_DIPSETTING(    0x10, "Off" )
PORT_DIPSETTING(    0x00, "On" )
PORT_BIT( 0x20, IP_ACTIVE_LOW, IPT_START1 )
PORT_BIT( 0x40, IP_ACTIVE_LOW, IPT_START2 )
PORT_DIPNAME( 0x80, 0x80, "Cabinet", IP_KEY_NONE )
PORT_DIPSETTING(    0x80, "Upright" )
```

```
PORT_DIPSETTING(    0x00, "Cocktail" )

PORT_START  /* DSW 1 */
      PORT_DIPNAME( 0x03, 0x01, "Coinage", IP_KEY_NONE )
PORT_DIPSETTING(    0x03, "2 Coins/1 Credit" )
PORT_DIPSETTING(    0x01, "1 Coin/1 Credit" )
PORT_DIPSETTING(    0x02, "1 Coin/2 Credits" )
PORT_DIPSETTING(    0x00, "Free Play" )
PORT_DIPNAME( 0x0c, 0x08, "Lives", IP_KEY_NONE )
PORT_DIPSETTING(    0x00, "1" )
PORT_DIPSETTING(    0x04, "2" )
PORT_DIPSETTING(    0x08, "3" )
PORT_DIPSETTING(    0x0c, "5" )
PORT_DIPNAME( 0x30, 0x00, "Bonus Life", IP_KEY_NONE )
PORT_DIPSETTING(    0x00, "10000" )
PORT_DIPSETTING(    0x10, "15000" )
PORT_DIPSETTING(    0x20, "20000" )
PORT_DIPSETTING(    0x30, "Never" )
PORT_DIPNAME( 0x40, 0x40, "Difficulty", IP_KEY_NONE )
PORT_DIPSETTING(    0x40, "Normal" )
PORT_DIPSETTING(    0x00, "Hard" )
PORT_DIPNAME( 0x80, 0x80, "Ghost Names", IP_KEY_NONE )
PORT_DIPSETTING(    0x80, "Normal" )
PORT_DIPSETTING(    0x00, "Alternate" )

PORT_START  /* DSW 2 */
      PORT_BIT( 0xff, IP_ACTIVE_LOW, IPT_UNUSED )

PORT_START  /* FAKE */
/* This fake input port is used to get the status of the fire button */
/* and activate the speedup cheat if it is. */
PORT_BITX(      0x01,  0x00,  IPT_DIPSWITCH_NAME  |  IPF_CHEAT,  "Speedup
Cheat", OSD_KEY_LCONTROL, OSD_JOY_FIRE1, 0 )
PORT_DIPSETTING(    0x00, "Off" )
PORT_DIPSETTING(    0x01, "On" )
INPUT_PORTS_END
```

Definición de gráficos:

```
static struct GfxLayout tilelayout =
{
8,8,  /* 8*8 characters */
    256,    /* 256 characters */
    2,  /* 2 bits per pixel */
    { 0, 4 },    /* the two bitplanes for 4 pixels are packed into one
byte */
    { 8*8+0, 8*8+1, 8*8+2, 8*8+3, 0, 1, 2, 3 }, /* bits are packed in
groups of four */
    { 0*8, 1*8, 2*8, 3*8, 4*8, 5*8, 6*8, 7*8 },
    16*8    /* every char takes 16 bytes */
};
```

Definición de sprites:

```
static struct GfxLayout spritelayout =
{
16,16,      /* 16*16 sprites */
```

```
64,   /* 64 sprites */
2,    /* 2 bits per pixel */
{ 0, 4 },   /* the two bitplanes for 4 pixels are packed into one byte
*/
{ 8*8, 8*8+1, 8*8+2, 8*8+3, 16*8+0, 16*8+1, 16*8+2, 16*8+3,
            24*8+0, 24*8+1, 24*8+2, 24*8+3, 0, 1, 2, 3 },
{ 0*8, 1*8, 2*8, 3*8, 4*8, 5*8, 6*8, 7*8,
            32*8, 33*8, 34*8, 35*8, 36*8, 37*8, 38*8, 39*8 },
64*8  /* every sprite takes 64 bytes */
};
```

2) Las funciones de la máquina virtual se encuentran en el fichero **`src/machine/pacman.c`** y son las siguientes:

- `pacman_init_machine()`**:** Inicialización de la máquina virtual.
- `pacman_interrupt()`**:** Función de tratamiento de interrupciones.

3) Las funciones de actualización del sonido se encuentran en el fichero **`src/sndhrdw/namco.c`** y son las siguientes (sólo se listan las más significativas):

- `namco_sh_update()`**:** Actualización del sonido para cada *frame* de vídeo.
- `namco_sh_start()`**:** Función de inicialización (inicialización de variables, reserva de memoria, etc).
- `namco_sh_stop()`**:** Función de finalización (destrucción de memoria dinámica).

4) Las funciones de actualización de vídeo se encuentran en el fichero **`src/vidhrdw/pengo.c`** y son las siguientes (sólo se listan las más significativas):

- `pengo_vh_screenrefresh()`**:** Actualización de la pantalla (1 *frame* de vídeo).
- `pacman_vh_start()`**:** Función de inicialización.

## 2.6.6. CRÉDITOS DEL MAME 0.34

A continuación se muestran los créditos del MAME 0.34 original para MS-DOS:

- MAME 0.34 versión original para DOS por Nicola Salmoria y el MAME Team (http://www.mame.net).
- Emulador de Z80: Z80Em Portable Zilog Z80 Emulator Copyright (C) Marcel de Kogel 1996,1997.
- Emulador de M6502 por Juergen Buchmueller.
- Emulador I86 David Hedley, modificado por Fabrice Frances.
- Emulador M6809 por John Butler, basado en L.C. Benschop's 6809 Simulator V09.
- M6808 basado en L.C. Benschop's 6809 Simulator V09.
- 80x86 asm M6808 emulator Copyright 1998, Neil Bradley
- M68000 emulator tomado de System 16 Arcade Emulator por Thierry Lescot.
- Emulador de 8039 por Mirko Buffoni, basado en 8048 emulator por Dan Boris.
- Emulador de T-11 Copyright (C) Aaron Giles 1998
- Emulador TMS34010 por Alex Pasadyn y Zsolt Vasvari.
- Emulador TMS9900 por Andy Jones, basado en el código original de Ton Brouwer.
- Emulador TMS5220 por Frank Palazzolo.
- AY-3-8910 basado en el trabajo de la siguiente gente: Ville Hallik, Michael Cuddy, Tatsuyuki Satoh, Fabrice Frances, Nicola Salmoria.
- Emulación de YM-2203 y YM-2151 por Tatsuyuki Satoh.
- YM-2203 basada en OPL por Ishmair
- Emulador POKEY por Ron Fries y la colaboración de Eric Smith, Hedley Rainnie y Sean Trowbridge.
- Información sobre el hardware de sonido de la NES por Jeremy Chadwick y Hedley Rainne.
- Emulación de YM3812 y YM3526 por Carl-Henrik Skårstedt.
- Emulación de YM2610 por Hiromitsu Shioya.

## 2.7. PORT DEL MAME A GP32

### 2.7.1. INTRODUCCIÓN

El *port* del MAME para GP32 es el objetivo del proyecto fin de carrera. Partiendo de la versión original para MS-DOS se ha realizado la adaptación del código fuente para que funcione en la consola portátil GP32.

Está organizado en 670 módulos fuente junto con varios ficheros `Makefile` en el directorio raíz de la distribución (cada uno de ellos es para cada uno de los distintos ejecutables), así como distintos directorios dentro de la carpeta “`src`” del proyecto para organizar los módulos:

- **`Makefile_*`**: Ficheros necesarios para compilar los distintos ejecutables del emulador (13 ficheros).
- **`M*.bmp`**: Son ficheros bitmap de 32x32 píxeles, los íconos de los distintos ejecutables dentro del lanzador de aplicaciones de la consola GP32.
- **`src/`**: Dentro del directorio raíz de la carpeta “`src`” se encuentran los módulos principales del emulador. Como la consola GP32 no tiene suficiente memoria para contener en un único ejecutable todos los *drivers* (sólo tiene 8 Mb), existen varios ficheros `Drivers_*.c` (Ejemplo: `Drivers_Capcom.c, Drivers_Atari.c, etc`), cada uno de ellos para un ejecutable distinto. Cada uno de estos ficheros, contiene la lista completa de drivers de juegos soportados por cada uno de los ejecutables.
- **`src/drivers/`**: Contiene los distintos *drivers* para hacer funcionar cada uno de los juegos soportados.
- **`src/gp32/`**: En esta carpeta se encuentran las librerías específicas del *port* para GP32 (re-escritas las originales de MS-DOS).
- **`src/I86/`**: Emulador de *Intel* 8086.
- **`src/I8039/`**: Emulador de *Intel* 8039.
- **`src/M6502/`**: Emulador de *Motorola* 6502.
- **`src/M6805/`**: Emulador de *Motorola* 6805.
- **`src/M6808/`**: Emulador de *Motorola* 6808.
- **`src/M6809/`**: Emulador de *Motorola* 6809.
- **`src/M68000/`**: Emulador de *Motorola* 68000.
- **`src/machine/`**: Contiene los distintos entornos hardware para emular cada una de las familias de juegos soportados.
- **`src/S2650/`**: Emulador de *Signetics* 2650.
- **`src/sndhrdw/`**: Contiene los distintos emuladores de procesadores de sonido.
- **`src/T11/`**: Emulador de DEC T-11.
- **`src/vidhrdw/`**: Contiene los distintos emuladores de procesadores gráficos.
- **`src/Z80/`**: Emulador del procesador Z80.

## 2.7.2. DESCRIPCIÓN DE LOS MÓDULOS FUENTE

A continuación se muestran una lista de los módulos del MAME portado a GP32, así como su descripción y comentarios:

| *Carpeta* | *Fichero* | *Descripción* | *Comentarios* |
|---|---|---|---|
| ./ | Makefile_atari<br>Makefile_capcom<br>Makefile_classics<br>Makefile_dataeast<br>Makefile_irem<br>Makefile_konami<br>Makefile_namco<br>Makefile_others<br>Makefile_sega<br>Makefile_taito<br>Makefile_tecmo<br>Makefile_technos<br>Makefile_Williams | Ficheros necesario para compilar el ejecutables del MAME para la consola GP32. | Realizados a partir del *Makefile* original para MS-DOS.<br><br>Se ha decidido dividir la colección completa de drivers en 13 ejecutables diferentes, cada uno para ejecutar una parte de esos drivers. |
| ./ | Matari.bmp<br>Mcapcom.bmp<br>Mclassic.bmp<br>Mdataeas.bmp<br>Mirem.bmp<br>Mkonami.bmp<br>Mnamco.bmp<br>Mothers.bmp<br>Msega.bmp<br>Mtaito.bmp<br>Mtecmo.bmp<br>Mtechnos.bmp<br>Mwilliams.bmp | Íconos de cada uno de los ejecutables. | Son bitmaps de 32x32 píxeles con una paleta de 8 bit estándar de la consola GP32.<br><br>Estos íconos aparecen en lanzador de aplicaciones de la consola GP32. |
| src/drivers/ | *201 módulos:*<br>1942.c<br>1943.c<br>8080bw.c<br>amidar.c<br>arabian.c<br>arkanoid2.c<br>arkanoid.c<br>etc… | Contiene la lista completa de drivers de juegos soportados por el emulador (201 ficheros). | Se han eliminado 45 ficheros de drivers de la versión original de MS-DOS. |
| src/I86/ | I86.c<br>I86.h<br>I86intrf.h<br>instr.h<br>modrm.h<br>ea.h<br>host.h | Contiene el emulador de Intel 8086. | Simplificado el código y eliminado desensamblador. |
| src/I8039/ | I8039.c<br>I8039.h | Contiene el emulador de Intel 8039 | Simplicado y sin desensamblador. |
| src/I8085/ | I8085.c | Contiene el | Simplicado y sin |

| *Carpeta* | *Fichero* | *Descripción* | *Comentarios* |
|---|---|---|---|
| | I8085.h<br>I8085CPU.h<br>I8085DAA.h | emulador de Intel 8085. | desensamblador. |
| src/M6502/ | m6502.c<br>m6502.h<br>m6502ops.h<br>tbl65c02.c<br>tbl6502.c<br>tbl6510.c | Emulador de Motorola 6502. | Simplicado y sin desensamblador. |
| src/M6805/ | 6805ops.c<br>m6805.c<br>m6805.h | Emulador de Motorota 6805. | Simplicado y sin desensamblador. |
| src/M6808/ | 6808ops.c<br>m6808.c<br>m6808.h | Emulador de Motorola 6808. | Simplicado y sin desensamblador. |
| src/M6809/ | 6809ops.c<br>m6809.c<br>m6809.h | Emulador de Motorola 6809. | Simplicado y sin desensamblador. |
| src/M68000/ | Asmintf.c<br>cpudefs.h<br>cpufunc.c<br>cputbl.h<br>cycletbl.h<br>m68000.h<br>mc68kmem.c<br>readcpu.h<br>opcode0.c<br>opcode1.c<br>opcode2.c<br>opcode3.c<br>opcode4.c<br>opcode5.c<br>opcode6.c<br>opcode7.c<br>opcode8.c<br>opcode9.c<br>opcodeb.c<br>opcodec.c<br>opcoded.c<br>opcodee.c | Emulador de Motorola 68000. | Simplicado y sin desensamblador.<br><br>El emulador se ha reducido en tamaño de 1.12 Mb de código fuente a 957 Kb. |
| src/machine/ | *68 módulos:*<br>6821pia.c<br>6821pia.h<br>8080bw.c<br>arabian.c<br>arkanoid.c<br>atarigen.c<br>etc… | Contiene los distintos entornos hardware para emular cada una de las familias de juegos soportados. | Se han eliminado 18 módulos de la versión original de MS-DOS. |
| src/gp32/ | config.c | Selección de opciones del emulador a partir de las opciones en los menús. | Re-escrito para funcionar en GP32 |

| *Carpeta* | *Fichero* | *Descripción* | *Comentarios* |
|---|---|---|---|
| | `enablecache.s`<br>`enablecache.o` | Módulo en ensamblador que activa la caché de la consola GP32 (por defecto desactivada). | Módulo nuevo. Sacado de Internet (autor anónimo). |
| | `fileio.c` | Entrada / Salida de ficheros de disco. | Re-escrito en parte para funcionar en la consola GP32. |
| | `gp32.c` | Función *GpMain()* principal del MAME para GP32. | Realizada a partir del *main()* original de MS-DOS. |
| | `gp32_fileio.c`<br>`gp32_fileio.h` | *Wrapper* de las funciones estándar de C de acceso a ficheros (`stdio.h`) a partir de las funciones de acceso a la SMC del GPSDK. | Realizado a partir de un módulo que venía con el entorno de desarrollo *DevKitAdvance*. |
| | `gp32_mame.h` | Fichero de cabecera general del *port* del MAME para GP32. | Fichero nuevo. Contiene diversas declaraciones de funciones y variables usadas desde distintos módulos del programa. |
| | `gp32_menu.c`<br>`gp32_menu.h` | Distintos menus de selección que se muestran al operador. | Módulos nuevos. |
| | `gp32menu.h`<br>`gp32menu.bmp` | Imagen de fondo de las ventanas de selección del emulador. | Ficheros nuevos. El fichero utilizado es gp32menu.h. Se consigue a partir de gp32menu.bmp (utilizando el programa GP32 Converter) |
| | `gp32splash.h`<br>`gp32splash.bmp` | Imagen de presentación del emulador. | Ficheros nuevos. El fichero utilizado es gp32splash.h. Se consigue a partir de gp32splash.bmp (utilizando el programa GP32 Converter) |

| ***Carpeta*** | ***Fichero*** | ***Descripción*** | ***Comentarios*** |
|---|---|---|---|
| | gpsoundbuf.c<br>gpsoundbuf.h | Librería de bajo nivel de acceso al sonido. | Módulos nuevos. Versión ligeramente modificada de las librerías de sonido originales de *Christian Novak*. |
| | Gpstart.c | Función main() del programa. | Proporcionada por el entorno de desarrollo. La función principal del MAME es GpMain() y está ubicada en el fichero gp32.c |
| | input.c | Lectura del teclado y del joystick. | Re-escrito a partir del original de MS-DOS. |
| | osd_cpu.h | Macros y definiciones de datos GP32. | Modificado desde el original de MS-DOS. |
| | sound.c | Librería de reproducción de audio utilizando *GpSoundBuf* para GP32. | Re-escrita a partir de la original utilizando *Allegro* para MS-DOS. |
| | vídeo.c | Librería de salida de vídeo, manejo de la paleta de colores y reserva de memoria para bitmaps y buffers de vídeo. | Re-escrito a partir del original de MS-DOS. |
| src/S2650/ | s2650.c<br>s2650.h<br>s2650cpu.h | Emulador de *Signetics* 2650. | Simplicado y sin desensamblador. |
| src/sndhrdw/ | *88 módulos:*<br>2151intf.c<br>2151intf.h<br>2203intf.c<br>2413intf.h<br>2610intf.c<br>etc… | Contiene los distintos emuladores de procesadores de sonido. | Eliminados 11 módulos de la versión original de MS-DOS. |
| src/T11/ | t11.c<br>t11.h<br>t11ops.c<br>t11table.c | Emulador de DEC T-11. | Simplicado y sin desensamblador. |

| ***Carpeta*** | ***Fichero*** | ***Descripción*** | ***Comentarios*** |
|---|---|---|---|
| src/vidhrdw/ | *239 módulos :*<br>1942.c<br>1943.c<br>8080bw.c<br>amidar.c<br>arabian.c<br>arkanoid.c<br>etc... | Contiene los distintos emuladores de procesadores gráficos. | Eliminados 40 ficheros de la versión original de MS-DOS. |
| src/Z80/ | z80.c<br>z80.h<br>z80daa.h | Emulador del procesador Z80. | Se ha cambiado el emulador de Z80 por otro que ofrece mejor rendimiento en arquitectura ARM: *Z80 Portable Emulator por Juergen Buchmueller.* |
| src/ | common.c<br>common.h | Funciones de propósito general. | Portado y simplificado todo lo posible. Entre otras cosas se ha eliminado el soporte para color de 16 bit en el dibujado de la pantalla. Con ello se reducen notablemente las necesidades de memoria. |
| src/ | cpuintrf.c<br>cpuintrf.h | Librería de interfaces con los distintos emuladores de procesadores. | Portado, simplificado todo lo posible y eliminados los procesadores que se quitaron de la versión de MS-DOS. |
| src/ | Driver_atari.c<br>Driver_capcom.c<br>Driver_classics.c<br>Driver_dataeast.c<br>Driver_irem.c<br>Driver_konami.c<br>Driver_namco.c<br>Driver_others.c<br>Driver_sega.c<br>Driver_taito.c<br>Driver_tecmo.c<br>Driver_technos.c<br>Driver_Williams.c<br>driver.h | Definición y listado de todos los drivers de juegos emulados por el MAME para la consola GP32. | Por limitaciones de memoria de la GP32 ha sido necesario crear distintos ficheros driver.c para distintas colecciones de juegos. |
| src/ | inflate.c<br>inflate.h | Librería para descomprimir ficheros ZIP. | Adaptada y simplificada. |
| src/ | inptport.c | Lectura y | Adaptada para la |

| *Carpeta* | *Fichero* | *Descripción* | *Comentarios* |
|---|---|---|---|
| | inptport.h | escritura de puertos, así como lectura de los controles, etc. | GP32. |
| src/ | mame.c<br>mame.h | Módulo principal de emulación. | Portada y simplificada todo lo posible. |
| src/ | memory.c<br>memory.h | Librería de entrada / salida en memoria de los juegos. | Portada y simplicada todo lo posible. |
| src/ | palette.c<br>palette.h | Librería de manejo de la paleta de colores de los juegos. | Portada, simplificada y restringida a color de 8 bit. |
| src/ | timer.c<br>timer.h | Librería de funciones de sincronización entre distintos procesadores gráficos, de vídeo y sonido. | Portada y simplificada. |
| src/ | types.h | Definición de tipos usados en el MAME. | Adaptada a la arquitectura de la consola GP32 (ARM). |
| src/ | unzip.c<br>unzip.h | Librería de tratamiento de ficheros comprimidos .ZIP. | Simplicada y adaptada para funcionar con la memoria SMC. |
| src/ | usrintrf.c<br>usrintrf.h | Entorno de usuario en pantalla del MAME. | Eliminado el entorno gráfico, sólo queda una función interesante: handle_user_ interface() que gestiona los controles especiales durante la emulación (pausa, salir del juego, etc) . |

## 2.7.3. ELIMINANDO CÓDIGO NO APLICABLE

Debido al limitado tamaño de memoria de la consola GP32 (8 Mb de memoria) se hace necesario (además de dividir el ejecutable original en varios ejecutables con soporte para una parte de los juegos cada uno de ellos) eliminar todo el código fuente no esencial para el funcionamiento del emulador.

A continuación se presenta un resumen del código eliminado:

- Eliminar todo el código fuente de depuración, ya que normalmente va dirigido a la salida estándar de C y en GP32 no la tenemos disponible. Además se consigue disminuir el tamaño del código fuente. Por último es indiferente este *debug*, ya que la versión del MAME que se está portando es muy antigua y no se pretende arreglar fallos que por otra parte deben de estar corregidos en futuras versiones.
- Eliminar debugger integrado en pantalla.
- Eliminar entorno gráfico y menús en pantalla de la versión de MS-DOS (se realizará un entorno gráfico propio).
- Eliminar el proceso analizador de rendimiento.
- Eliminar el soporte de grabación y lectura de records de puntuación.
- Eliminar el soporte de grabación y reproducción de partidas.
- Eliminar el soporte de *artwork* (marcos de pantalla presentes en algunos juegos clásicos, que se emulan mediante bitmaps leidos del disco y visualizados alrededor de la imagen real del juego).
- Eliminar el módulo de chequeo extensivo de las ROMS.
- Eliminar soporte para trucos en los juegos.
- Eliminar módulo de informes sobre los *drivers*.
- Eliminar la opción de cambio de los *dip-switches* de las recreativas.
- Eliminar el soporte de color de 16 bit en el MAME, ya que si se usase esa opción las necesidades de memoria aumentarían bastante debido a que los *buffers* de pantalla en memoria ocuparían el doble de memoria (lo mismo para los *sprites* y gráficos de los juegos).
- Eliminar el código de los *drivers* que será imposible hacer funcionar en la consola GP32 por limitaciones de memoria, fundamentalmente son:
    - Los juegos modelo Neo·Geo de la empresa SNK, que no caben en la memoria de la GP32 porque ocupan varias megas de memoria.
    - Los juegos de Sega de modelo System-16, por idéntica razón.
    - Los juegos de Capcom de modelo CPS-1.
    - Los juegos de Midway con procesadores *Texas Intstruments* TMS9900 ó TMS34010, ya que los juegos ocupan mucha memoria y los drivers del MAME 0.34 eran muy lentos.
    - Otros juegos diversos que ocupan demasiada memoria para la GP32 (8 Mb) ó por tamaño de pantalla superior a la resolución de la GP32 (320x240 píxeles).
- Eliminar código de los procesadores no usados por los juegos a soportar en la consola GP32: Texas Instruments TMS9900 y TMS34010.
- Eliminar librería de gráficos vectoriales y los juegos que requieren de ella, ya que el número de juegos que utilizan esta funcionalidad son pocos y la complejidad de la librería es grande.

## 2.7.4. PORTANDO LOS MÓDULOS FUENTE A GP32

A continuación se muestran a grandes rasgos algunos pasos esenciales llevados a cabo en todos los módulos para portarlos a la consola GP32:

- Sustituir las llamadas a funciones estándar de memoria de C a las funciones respectivas específicas del GPSDK: `malloc`() por `gm_zi_malloc`(), `free`() por `gm_free`(), `memcpy`() por `gm_memcpy`(), etc.
- Sustituir las llamadas a funciones estándar de manejo de caracteres de C por sus equivalentes del GPSDK: Fundamentalmente `sprintf`() por `gm_sprintf`(), `strcmp`() por `gm_strcmp`(), etc.
- Eliminar el código que utiliza la entrada estándar (*stdin*), salida estándar (*stdout*) y salida estándar de error (*stderr*). En los casos en que se necesite sacar algo esencial por pantalla, se hará mediante funciones específicas del GPSDK.
- Eliminar el código y las funcionalidades del MAME que impliquen escritura en el disco: grabación de las *eprom* de las recreativas tras su configuración mediante *dip-switches*, grabación de capturas de pantalla en el disco, etc.
- Habilitar las funciones necesarias para leer la memoria de forma estrictamente alineada. En los fuentes del MAME original hay código que en función de la dirección de memoria en que se encuentren los datos, lee los enteros de 32 bit tal cual con una indirección o bien leyendo byte a byte para luego convertir el resultado a un entero. Este código originalmente específico para ACORN es necesario activarlo en GP32, ya que en caso contrario, la MMU de la GP32 resetea la consola si se intenta leer un entero de forma no alineada en memoria, ya que lo considera un acceso a memoria inválido. Para activar este código es necesario definir la clave “ACORN” durante la compilación (hay muchos `#ifdef ACORN...#endif` en el código fuente para activar esa opción en determinados casos).
- Estudiar el código fuente esencial que acceda al disco (fundamentalmente la lectura de ROMs), ya que será necesario portarlo a la GP32, sustituyendo las funciones estándar de ficheros de C por otras funciones con igual funcionalidad construidas mediante las funciones de acceso a la tarjeta SMC del GPSDK.
- Análisis de los módulos a re-escribir para el port a GP32 (acceso a la pantalla, al sonido, los controles de la consola, el entorno gráfico del emulador, etc).

## 2.7.5. MÓDULOS ESPECÍFICOS DEL *PORT* A GP32

### 2.7.5.1 SRC\GP32\CONFIG.C

Este módulo se ha transformado para conseguir los parámetros de emulación de los menús de selección del port a GP32, en lugar de la línea de comandos del MS-DOS. Esta configuración es introducida en la estructura `GameOptions` que usa el MAME.

El código fuente del módulo es el siguiente:

```
#include "driver.h"
#include <ctype.h>

/* from vídeo.c */
extern int gfx_width, gfx_height;

/* from sound.c */
extern int soundcard;
extern int use_emulated_ym3812;

/* from input.c */
extern int use_mouse, joystick;
static int mame_argc;
static char **mame_argv;
static int game;

/* from gp32_menu.c */
extern int gp32_frameskip;
extern int gp32_sound;
extern int gp32_freq;

/* from vídeo.c */
extern int gp32_timer;

void parse_cmdline (int argc, char **argv, struct GameOptions *options,
int game_index)
{
mame_argc = argc;
mame_argv = argv;
game = game_index;

options->frameskip = gp32_frameskip; /* 1,2,3... */
options->norotate  = 0;
options->ror       = 0;
options->rol       = 0;
options->flipx     = 0;
options->flipy     = 0;

/* read sound configuration */
soundcard=gp32_sound;   /*   1   sound,   0   not   emulated   sound,
                                2 no sound but emulated */

use_emulated_ym3812=1;
options->samplerate = 22050;
options->samplebits = 8;

/* read input configuration */
```

```
use_mouse = 0;
joystick  = 0;

/* misc configuration */
options->cheat      = 0;
/*options->mame_debug = 0;*/

/*  Number  of  GP32  ticks  in  a  second,  depending  on  GP32
            Frequency */
gp32_timer=(gp32_freq*1950)/133;

}
```

Las variables siguientes son escritas por el menú de selección de opciones del emulador (`gp32_menu.c`):

- **`extern int gp32_frameskip`**: Valor de frameskip de vídeo durante la emulación. Un valor óptimo es 0, ya que se dibujan todos los frames de vídeo (la velocidad de actualización de la pantalla es de 60 imágenes por segundo). Un valor mayor implica salto de frames (1=30 frames por segundo, 2 = 20 frames por segundo, etc). Se permiten valores de 0 a 5 de frameskip.
- **`extern int gp32_sound`**: Configuración del sonido. 1 = sonido activado, 0 = sonido desactivado pero emulado, 2 = sonido desactivado y no emulado. Algunos juegos no funcionan con la opción 2, ya que necesitan que los procesadores de sonido estén funcionando (realizan sincronización con ellos).
- **`extern int gp32_freq`**: Frecuencia del procesador de la consola GP32. El valor por defecto en el menú es la frecuencia más baja (133 MHz). A mayor velocidad se consigue mejor rendimiento, pero también mayor gasto de pilas de la consola. Un valor bueno es 166 MHz, aunque hay opciones de overclocking hasta 256 MHz. La mayoría de las consolas GP32 no soportan frecuencias superiores a 166 MHz.

A partir de la frecuencia del procesador `(gp32_freq)`, se calcula la variable global **`gp32_timer`**, que es el valor de *ticks* del procesador en un segundo para la frecuencia en cuestión. Esta variable es utilizada para que el emulador funcione en tiempo real con independencia de la velocidad del micro (ni muy rápido ni muy lento). El cálculo consiste en multiplicar la frecuencia del procesador por (1950/133). Este valor difiere algo del valor teórico, y se ha conseguido tras diversas pruebas. El valor teórico es gp32_freq * ( 1000 / 66 ), y es así ya que el valor de *ticks* en un segundo a una frecuencia de 66 MHz es aproximadamente igual al número de milisegundos transcurridos.

### 2.7.5.2 SRC\GP32\ENABLECACHE.S

Este módulo nuevo en ensamblador activa la caché de alineamiento de la GP32, ya que por defecto está desactivada. Con ello se consigue mejor rendimiento. Es utilizada al comienzo de la función GpMain() de src\gp32\gp32.c.

```
@ asmfunc.s
@ void CPU_alignment_cache(void);
.ARM
.GLOBAL EnableCache

EnableCache:
stmfd r13!,{r5,r6}

  mrc       p15, 0, r5, c1, c0, 0
  bic       r5,r5,#2
      ldr         r6,=0x1004
  orr       r5,r5,r6
  mcr       p15, 0, r5, c1, c0, 0

ldmfd r13!,{r5,r6}
bx    lr
```

Este fichero se ha conseguido de Internet, su autor es anónimo.

### 2.7.5.3 SRC\GP32\GP32.C

Entre otras funciones de propósito general, en este módulo está la función **GpMain()** principal del programa (la función **main()** es proporcionada por el propio entorno de desarrollo, se encuentra en el fichero **gpstart.c**).

Lo primero que la función **GpMain()** es inicializar el hardware de la consola GP32:

- `EnableCache()`: Habilita caché de alineamiento de la CPU.
- `GpFatInit()`: Inicializa el acceso a la tarjeta de memoria SMC.
- `GpClockSpeedChange()`: Cambia frecuencia de reloj de la consola a 133 MHz.
- `gp32_vídeo_init()`: Inicializa modo gráfico de la consola (Ojo: es necesario además hacerlo cada vez que se cambia la velocidad de reloj de la consola).
- `intro_screen()`: Muestra la pantalla de presentación del emulador en pantalla.
- `game_list_init()`: Esta función realiza el chequeo de los juegos encontrados en la tarjeta SMC, para despues mostrarlos en el menú de selección.

Una vez se ha inicializado el entorno del emulador, se pasa a un bucle infinito. En este bucle infinito se realizan los siguientes pasos:

- `GpClockSpeedChange()`: Se cambia frecuencia de reloj de la consola a 66 MHz para ahorrar baterías, además de volver a llamar a gp32_vídeo_init().
- `select_game()`: Se muestra la pantalla de selección y los menús del emulador. Cuando el operador selecciona un juego de la lista y se eligen las opciones de emulación se termina la ejecución de esta función.
- `GpClockSpeedChange()`: Se cambia la frecuencia de reloj de la consola por la seleccionada por el usuario en los menús (variable global gp32_freq). Se realiza `gp32_vídeo_init()` de nuevo.
- `msdos_init_seal()`: Se inicializa el acceso al audio de la consola GP32. Es necesario hacerlo además cada vez que se cambia la frecuencia de reloj de la consola.
- Realiza algunas inicializaciones y chequeos varios sin importancia.
- `run_game()`: Llama a la función original del MAME para hacer funcionar el juego seleccionado.

A continuación se muestra el código fuente de la función **GpMain():**

```
void GpMain(void *arg)
{
/* Command Line Arguments Emulation */
int argc=2;
char *argv[] = {"mame","builtinn"};
char text[64];

int res, i, j, game_index;

/* CPU Alignment Cache */
EnableCache();

/* SMC Access Init */
      GpFatInit();

/* Clock Speed 133 mhz */
GpClockSpeedChange(132000000, 0x3a011, 3);

/* GP32 Vídeo Init */
gp32_vídeo_init();

/* Show Intro Screen */
intro_screen();

/* Initialize Game Listings */
      game_list_init();

while(1) {

      /* Clock Speed 66 MHz */
      GpClockSpeedChange(67500000,0x25002,2);

      /* GP32 Vídeo Init */
      gp32_vídeo_init();

      /* Select Game */
      select_game(argv);

      /* Select Clock Speed */
      switch(gp32_freq) {
            case 133: /* Clock Speed 133 mhz */
                  GpClockSpeedChange(132000000,
                              0x3a011, 3);
                  break;
            case 144: /* Clock Speed 144 mhz */
                        GpClockSpeedChange(140000000,
                               0x3e011, 3);
                  break;
            case 150: /* Clock Speed 150 mhz */
                        GpClockSpeedChange(150000000,
                               0x43011, 3);
                  break;
            case 156: /* Clock Speed 156 mhz */
                  GpClockSpeedChange(156000000,
                              0x46011, 3);
                  break;
            case 160: /* Clock Speed 160 mhz */
                        GpClockSpeedChange(160000000,
```

```
                0x48011, 3);
        break;
case 166: /* Clock Speed 166 mhz */
        GpClockSpeedChange(166000000,
                0x4b011, 3);
        break;
case 168: /* Clock Speed 168 mhz */
        GpClockSpeedChange(168000000,
                0x14000, 3);
        break;
case 172: /* Clock Speed 172 mhz */
        GpClockSpeedChange(172000000,
                0x23010, 3);
        break;
case 176: /* Clock Speed 176 mhz */
        GpClockSpeedChange(176000000,
                0x24010, 3);
        break;
case 180: /* Clock Speed 180 mhz */
        GpClockSpeedChange(180000000,
                0x16000, 3);
        break;
case 184: /* Clock Speed 184 mhz */
        GpClockSpeedChange(184000000,
                0x26010, 3);
        break;
case 188: /* Clock Speed 188 mhz */
        GpClockSpeedChange(188000000,
                0x27010, 3);
        break;
case 192: /* Clock Speed 192 mhz */
        GpClockSpeedChange(192000000,
                0x18000, 3);
        break;
case 196: /* Clock Speed 196 mhz */
        GpClockSpeedChange(196000000,
                0x29010, 3);
        break;
case 200: /* Clock Speed 200 mhz */
        GpClockSpeedChange(200000000,
                0x2A010, 3);
        break;
case 204: /* Clock Speed 204 mhz */
        GpClockSpeedChange(204000000,
                0x2b010, 3);
        break;
case 208: /* Clock Speed 208 mhz */
        GpClockSpeedChange(208000000,
                0x2c010, 3);
        break;
case 212: /* Clock Speed 212 mhz */
        GpClockSpeedChange(212000000,
                0x2d010, 3);
        break;
case 216: /* Clock Speed 216 mhz */
        GpClockSpeedChange(216000000,
                0x2e010, 3);
        break;
case 220: /* Clock Speed 220 mhz */
        GpClockSpeedChange(220000000,
```

```
                        0x2f010, 3);
                break;
        case 224: /* Clock Speed 224 mhz */
                GpClockSpeedChange(224000000,
                        0x30010, 3);
                break;
        case 228: /* Clock Speed 228 mhz */
                GpClockSpeedChange(228000000,
                        0x1e000, 3);
                break;
        case 232: /* Clock Speed 232 mhz */
                GpClockSpeedChange(232000000,
                        0x32010, 3);
                break;
        case 236: /* Clock Speed 236 mhz */
                GpClockSpeedChange(236000000,
                        0x33010, 3);
                break;
        case 240: /* Clock Speed 240 mhz */
                GpClockSpeedChange(240000000,
                        0x20000, 3);
                break;
        case 244: /* Clock Speed 244 mhz */
                GpClockSpeedChange(244000000,
                        0x35010, 3);
                break;
        case 248: /* Clock Speed 248 mhz */
                GpClockSpeedChange(248000000,
                        0x36010, 3);
                break;
        case 252: /* Clock Speed 252 mhz */
                GpClockSpeedChange(252000000,
                        0x22000, 3);
                break;
        case 256: /* Clock Speed 256 mhz */
                GpClockSpeedChange(256000000,
                        0x38010, 3);
                break;
}

/* GP32 Vídeo Init */
gp32_vídeo_init();

/* Initialize the audio library */
msdos_init_seal();

/* Restore MAME Palette */
gp32_mame_palette();

/* Set Log Messages start at row 0 */
gp32_gamelist_zero();

/* take the first commandline argument without "-"
            as the game name */
for (j = 1; j < argc; j++)
      if (argv[j][0] != '-') break;
gm_lowercase(argv[j],gm_lstrlen(argv[j]));
game_index = -1;

/* do we have a driver for this? */
```

```
	for (i = 0; drivers[i] && (game_index == -1); i++)
	{
		if (gm_compare(argv[j],drivers[i]->name) == 0)
		{
			game_index = i;
			break;
		}
	}

	if (game_index == -1)
	{
		sprintf(text,"Game\"%s\"not supported\0",argv[1]);
		gp32_text_log(text);
		while(1); /* MAME never ends xD */
	}

	/* parse generic (os-independent) options */
	parse_cmdline (argc, argv, &options, game_index);

	{	/* Mish:  I need sample rate initialised
			_before_ rom loading for optional rom
			regions */
		extern int soundcard;

		if (soundcard == 0) {	/* silence, this would
				be -1 if unknown in which case all roms
				are loaded */
			Machine->sample_rate = 0; /* update the
				Machine structure to show that sound is disabled
				*/
			options.samplerate=0;
		}
	}

	sprintf(text,"Loading \"%s\"...",
			drivers[game_index]->description);
	gp32_text_log(text);

	/* go for it */
	res = run_game (game_index , &options);

	/* Wait 5 seconds */
	if (res!=0)
		Delay(10000);

} /* MAME never ends xD */
}
```

### 2.7.5.4 SRC\GP32\GP32_FILEIO.C

Este módulo nuevo simula las funciones estándar de entrada / salida de ficheros de C (stdio.h) mediante las funciones de manejo de ficheros en la tarjeta de memoria SMC proporcionados por el GPSDK.

**NOTA**: Es necesario realizar `GpFatInit()` antes de utilizar estas funciones (Se hace en `gp32.c:GpMain()`).

Lista de funciones:

- `gp32_fexists()`: Indica si un fichero existe.
- `gp32_fopen()`: Abre un fichero.
- `gp32_close()`: Cierra un fichero.
- `gp32_getc() y gp32_fgetc()`: Lectura de caracteres.
- `gp32_putc() y gp32_fputc()`: Escritura de caracteres.
- `gp32_ftell()`: Devuelve posición en el fichero.
- `gp32_fseek()`: Busca posición en el fichero.
- `gp32_fread()`: Lectura de bytes.
- `gp32_fwrite()`: Escritura de bytes.
- `gp32_frewind()`: Puntero al principio del fichero.

Este módulo se ha realizado partiendo de otro ya existente y de autor anónimo (venía con el entorno de desarrollo *DevKitAdvance*).

A continuación se muestran algunas de las funciones más interesantes:

```
BOOL gp32_fexists ( const char * filename )
  {
    GPFILEATTR attr;
    if (GpFileAttr(filename, &attr)!=SM_OK)
      return FALSE;
    return TRUE;
  }

GP32_FILE * gp32_fopen ( const char * filename, const char * mode )
  {
    GP32_FILE * fh;
    F_HANDLE gpfh;
    unsigned long m = 0;
    BOOL append=FALSE;
    BOOL change=FALSE;
    if (mode[0]=='\0')
      return NULL;
    else
    if (tolower(mode[0])=='r')
      m |= OPEN_R;
    else
    if (tolower(mode[0])=='w')
      m |= OPEN_W;
    else
    if (tolower(mode[0])=='a')
      append=TRUE;
    else
```

```
      return NULL;

    if (mode[1]!='\0')
      {
        if (mode[1]=='+')
          change=TRUE;
      }
    if (m&OPEN_W) /* W */
      {
        /* Flush file */
        if (GpFileCreate(filename, ALWAYS_CREATE, &gpfh)!=SM_OK)
          return NULL;
      } else
    if (append)   /* A */
      {
        if (gp32_fexists(filename))
          {
            long s;

            if (GpFileOpen(filename, OPEN_R|(change?OPEN_W:0),
                   &gpfh)!=SM_OK)
              return NULL;
            GpFileSeek ( gpfh, FROM_END, 0, &s );
          } else
          {
                 if (GpFileCreate(filename, ALWAYS_CREATE,
                        &gpfh)!=SM_OK)
              return NULL;
          }
      } else      /* R */
    if (m&OPEN_R)
      {
        ERR_CODE err =GpFileOpen(filename, m|(change?OPEN_W:0),
            &gpfh);
        if (err!=SM_OK)
          return NULL;
      } else
      return NULL;

    if ( (fh = gm_zi_malloc ( sizeof(GP32_FILE) ))==NULL )
      {
        GpFileClose ( gpfh );
        return NULL;
      }

    *fh = gpfh;

    return fh;
  }

int gp32_fclose ( GP32_FILE * fh )
  {
    if (fh==NULL)
      return EOF;
    if (GpFileClose(*fh)!=SM_OK)
      return EOF;
    gm_free(fh);
    return 0;
  }
```

```
size_t gp32_fread ( void * ptr, size_t size, size_t nobj, GP32_FILE * fh
)
  {
    unsigned int obj;
    unsigned int ofs;
    if (nobj==0 || size==0)
      return 0;
    if (fh==NULL)
      return 0;

    for (obj=0,ofs=0;obj<nobj;obj++,ofs+=size)
      {
        unsigned long r;
        ERR_CODE err;
        err=GpFileRead(*fh, ((u8*)ptr)+ofs, size, &r);
        if (err!=SM_OK || r!=size)
          return obj;
      }
    return obj+1;
  }

size_t gp32_fwrite ( const void * ptr, size_t size, size_t nobj,
GP32_FILE * fh )
  {
    int obj,ofs;
    if (fh==NULL)
      return 0;
    if (nobj==0 || size==0)
      return 0;
    for (obj=0,ofs=0;obj<nobj;obj++,ofs+=size)
      {
        ERR_CODE err;
        err=GpFileWrite ( *fh, (void*)(((u8*)ptr)+ofs), size );
        if (err!=SM_OK)
          return obj;
      }
    return obj+1;
  }
```

### 2.7.5.5 SRC\GP32\FILEIO.C

Este módulo nuevo implementa las funciones de entrada / salida de ficheros del MAME para la consola GP32. Esta realizado siguiendo la estructura del módulo original para MS-DOS.

El módulo está restringido sólo para leer, no para escribir en la tarjeta de memoria SMC (lo que desgastaría la tarjeta de memoria y además podría corromper el sistema de ficheros si se hiciera indebidamente ó se gastan las baterías durante una escritura).

Estas funciones leen ROMs desde la tarjeta SMC, siendo transparente si estan descomprimidas ó comprimidas en ficheros ZIP.

Las funciones más interesantes del módulo son las siguientes:

- `cache_stat()`: Indica si un determinado fichero existe en el disco.
- `osd_fopen()`: Abre fichero ROM desde el disco.
- `osd_fread()`: Lee ROM del disco.
- `osd_fclose()`: Cierra fichero ROM del disco.

Si se utilizan ficheros ZIP el consumo de memoria del programa es mayor, ya que automáticamente al abrir una ROM en un ZIP, se lee todo el fichero y se descomprime completo en memoria, por lo que luego en los osd_fread() se consigue la ROM desde la memoria. Al cerrar con osd_fclose() se libera dicha memoria.

A continuación se presenta como ejemplo la función **cache_stat()** que indica si un determinado fichero existe en la memoria SMC:

```
int cache_stat(char *name) {

    char filename[128];
    unsigned int i;
    int found=0;

    sprintf(filename,"gp:\\gpmm\\mame034\\%s\0",name);

    if (gp32_fexists(filename))
      return 0;
    else
      return 1;

}
```

Aquí la función **`osd_fopen()`** que abre una ROM o bien lee el fichero ZIP completo en el caso de ficheros comprimidos:

```
void *osd_fopen(const char *game,const char *filename,int filetype,int
_write)
{
char name[256];
char *gamename;
int found = 0;
FakeFileHandle *f;

f = (FakeFileHandle *)gm_zi_malloc(sizeof(FakeFileHandle));
if (!f)
      return 0;
gm_memset(f,0,sizeof(FakeFileHandle));

gamename = (char *)game;

switch (filetype)
{
      case OSD_FILETYPE_ROM:

            /* only for reading */
            if (_write)
                  break;

            if (!found) {
                  if (cache_stat(gamename)==0) {

                              sprintf(name,"gp:\\gpmm\\mame034\\
                              %s\\%s\0",gamename,filename);
                        if (checksum_file (name, &f->data,
                                    &f->length, &f->crc)==0) {
                              f->type = kRAMFile;
                              f->offset = 0;
                              found = 1;
                        }
                  }
            }

            if (!found) {
                  /* try with a .zip extension */
                  sprintf(name,"%s.zip\0", gamename);
                  if (cache_stat(name)==0) {

                              sprintf(name,"gp:\\gpmm\\mame034\\
                                    %s.zip\0", gamename);
                        if (load_zipped_file(name,
                                    filename, &f->data,
                                    &f->length)==0) {
                              f->type = kZippedFile;
                              f->offset = 0;
                              f->crc = crc32 (0L, f->data,
                                          f->length);
                              found = 1;
                        }
                  }
            }

            break;
```

```
	case OSD_FILETYPE_SAMPLE:
		break;
	case OSD_FILETYPE_HIGHSCORE:
		break;
	case OSD_FILETYPE_CONFIG:
		break;
	case OSD_FILETYPE_INPUTLOG:
		break;
	case OSD_FILETYPE_STATE:
		break;
	case OSD_FILETYPE_ARTWORK:
		break;
}

if (!found) {
	gm_free(f);
	return 0;
}

return f;
}
```

La función **osd_fread()** que lee la ROM del disco si esta descomprimida o bien del buffer del ZIP descomprimido en memoria en el caso de ROMs descomprimidas.

```
int osd_fread(void *file,void *buffer,int length)
{
FakeFileHandle *f = (FakeFileHandle *)file;

switch (f->type)
{
	case kPlainFile:
		return gp32_fread(buffer,1,length,f->file);
		break;
	case kZippedFile:
	case kRAMFile:
		/* reading from the RAM image of a file */
		if (f->data)
		{
			if (length + f->offset > f->length)
				length = f->length - f->offset;
			gm_memcpy(buffer, f->offset + f->data,
					length);
			f->offset += length;
			return length;
		}
		break;
}

return 0;
}
```

Y por último la función **osd_fclose()** que cierra el fichero en disco si la ROM está descomprimida o libera la memoria dinámica si es un ZIP.

```
void osd_fclose(void *file)
{
FakeFileHandle *f = (FakeFileHandle *) file;

switch(f->type)
{
      case kPlainFile:
            gp32_fclose(f->file);
            break;
      case kZippedFile:
      case kRAMFile:
            if (f->data)
                  gm_free(f->data);
            break;
}
gm_free(f);
}
```

### 2.7.5.6 SRC\GP32\GP32_MENU.C

Este módulo nuevo implementa el menú de selección de juegos y el menú de configuración del emulador, así como la inicialización de la lista de juegos.

Las funciones que contiene son las siguientes:

- `game_list_init()`: Inicializa la lista de juegos que el usuario tiene en la tarjeta SMC de su consola. Verifica en el disco la existencia de los ficheros de cada uno de los juegos.
- `select_game()`: Función que muestra el menú de selección de juegos, espera a que el operador seleccione uno de los juegos, muestra el menú de configuración del emulador y espera a que el operador configure las opciones.
- `game_list_view()`: Visualiza el menú de selección de juegos.
- `game_list_select()`: Devuelve el nombre del juego seleccionado por el usuario.
- `game_list_description()`: Devuelve la descripción del driver del juego seleccionado.
- `game_options()`: Muestra al operador las opciones del emulador.
- `Delay()`: Función que pausa el programa un determinado número de milisegundos.
- `intro_screen()`: Muestra la pantalla de presentación del emulador.

A continuación se muestra el listado de la función **game_list_init()**:

```
void game_list_init(void) {

unsigned int c;
char filename[128];

/* Check Available Games */
for (c=0;(c<game_num && drivers[c]);c++) {

      sprintf(filename,"gp:\\gpmm\\mame034\\%s.zip\0",
                  drivers[c]->name);
      if (gp32_fexists(filename)) {
            game_avail[c]=1;
            game_num_avail++;
            continue;
      }
      sprintf(filename,"gp:\\gpmm\\mame034\\%s\0",
                  drivers[c]->name);
      if (gp32_fexists(filename)) {
            game_avail[c]=1;
            game_num_avail++;
            continue;
      }

}
}
```

El listado de la función **select_game()**:

```
void select_game(char *argv[]) {

int ExKey;
int c;

/* No Selected game */
gm_strcpy(argv[1],"builtinn");

/* Clean screen*/
      GpRectFill( NULL, &gpDraw[0],0,0, gpDraw[0].buf_w,
            gpDraw[0].buf_h, 0x0 );
      GpRectFill( NULL, &gpDraw[1],0,0, gpDraw[1].buf_w,
            gpDraw[1].buf_h,0x0 );

/* Wait until no key pressed */
while(GpKeyGet()!=GPC_VK_NONE) { }

/* Available games? */
if(game_num_avail==0) {
      GpTextOut( NULL, &gpDraw[nflip],35,110,
                  "ERROR: NO AVAILABLE GAMES FOUND", 254 );
      while(1) { }
}

for (c=0;c<256;c++) {
            GpPaletteEntryChange(c,1,&gp32menu_Pal[c],0);
}

/* Wait until user selects a game */
while(1) {
      nflip=(nflip==1?0:1);
      GpRectFill( NULL, &gpDraw[nflip], 0, 0,
                  gpDraw[nflip].buf_w, gpDraw[nflip].buf_h,0x0 );
      game_list_view(&last_game_selected);
      GpSurfaceSet( &gpDraw[nflip] );
      while((ExKey=GpKeyGet())!=GPC_VK_NONE) {
            /* Reset GP32 */
            if ((ExKey & GPC_VK_FL) && (ExKey & GPC_VK_FR))
                  {
                  GpClockSpeedChange(67800000, 0x69032, 3);
                  __asm("swi #4;");
            }
      }
      while((ExKey=GpKeyGet())==GPC_VK_NONE) {}

      /* Reset GP32 */
      if ((ExKey & GPC_VK_FL) && (ExKey & GPC_VK_FR)) {
            GpClockSpeedChange(67800000, 0x69032, 3);
            __asm("swi #4;");
      }
      if (ExKey & GPC_VK_UP) last_game_selected--;
      if (ExKey & GPC_VK_DOWN) last_game_selected++;
      if (ExKey & GPC_VK_FL) last_game_selected-=11;
      if (ExKey & GPC_VK_FR) last_game_selected+=11;
      if (!(ExKey & GPC_VK_FA) && !(ExKey & GPC_VK_FB))
                  continue;
      /* Emulation Options */
      if (game_options()) break;
```

```
}

/* Select the game */
game_list_select(last_game_selected,argv[1]);

 /* Clean screen */
      GpRectFill( NULL, &gpDraw[0], 0,0, gpDraw[0].buf_w,
             gpDraw[0].buf_h,0x0 );
      GpRectFill( NULL, &gpDraw[1],0,0, gpDraw[1].buf_w,
            gpDraw[1].buf_h,0x0 );
}
```

### 2.7.5.7 SRC\GP32\GPSOUNDBUF.C

Este módulo es una versión ligeramente modificada del original de *Christian Nowak*. Se trata de una librería de reproducción de sonido que utiliza un buffer de sonido circular actualizado por una función de interrupción activada periódicamente mediante timer.

```
int GpSoundBufStart ( GPSOUNDBUF * sb )
{
    frame = 0;

    /* Copy the structure */
    gm_memcpy ( &soundBuf, sb, sizeof(GPSOUNDBUF) );

    /* Calculate size of a single sample in bytes
     * and a corresponding shift value */
    shiftVal = 0;
    switch (soundBuf.freq)
      {
        case PCM_S11:
        case PCM_S22:
        case PCM_S44:
          shiftVal++;
          break;
      }
    if (soundBuf.format == PCM_16BIT)
      shiftVal++;
    soundBuf.samplesize = 1<<shiftVal;

    /* Allocate memory for the playing buffer */
    buffer = gm_zi_malloc(soundBuf.samplesize*soundBuf.samples*2 );
    gm_memset(buffer, 0x80, soundBuf.samplesize*soundBuf.samples*2 );

    /* Set timer interrupt #0 */
    if ( GPOS_ERR_ALREADY_USED ==
 GpTimerOptSet ( 0, soundBuf.pollfreq, 0, soundtimer ) ) {
      GpTimerKill(0);
    }
    GpTimerSet ( 0 );

    /* Start playing */
    GpPcmPlay( (unsigned short*)buffer,
            soundBuf.samples*soundBuf.samplesize*2, 1 );
    GpPcmLock ( (unsigned short*)buffer, (int*)&idx_buf,
            (unsigned int*)&soundPos );
    return 0;
}


void GpSoundBufStop ( void )
{
    GpPcmStop();
    GpPcmRemove ( (unsigned short*)buffer );
    GpTimerKill ( 0 );
    gm_free ( buffer );
}
```

### 2.7.5.8 SRC\GP32\GPSTART.C

Este módulo contiene la función `main()` principal del programa que viene con el entorno de desarrollo y no es conveniente modificar. Esta función configura e inicializa el acceso al hardware de la consola y después llama internamente a la función `GpMain()` que es la función principal del MAME (está implementada en `gp32.c`).

El código fuente de la función `main()` es el siguiente:

```
int main (int arg_len, char * arg_v)   {

GM_HEAP_DEF gm_heap_def;
#if CHANGE_MCLK
      GpClockSpeedChange (CLK_SPEED, DIV_FACTOR, CLK_MODE);
#endif
_gp_sdk_init();

/* keyboard polling count setting */
GpKeyPollingTimeSet (KEYPOLLING_NUM);

#ifdef USE_GP_MEM
      gm_heap_def.heapstart = (void*)(HEAPSTART);
      gm_heap_def.heapend = (void *)(HEAPEND & ~3);
      gm_heap_init (&gm_heap_def);
      gp_mem_func.malloc = gm_malloc;
      gp_mem_func.zimalloc = gm_zi_malloc;
      gp_mem_func.calloc = gm_calloc;
      gp_mem_func.free = gm_free;
      gp_mem_func.availablemem = gm_availablesize;
      gp_mem_func.malloc_ex = gm_malloc_ex;
      gp_mem_func.free_ex = gm_free_ex;
      gp_mem_func.make_mem_partition = gm_make_mem_part;
      gp_str_func.memset = gm_memset;
      gp_str_func.memcpy = gm_memcpy;
      gp_str_func.strcpy = gm_strcpy;
      gp_str_func.strncpy = gm_strncpy;
      gp_str_func.strcat = gm_strcat;
      gp_str_func.strncat = gm_strncat;
      gp_str_func.gpstrlen = gm_lstrlen;
      gp_str_func.sprintf = gm_sprintf;
      gp_str_func.uppercase = gm_uppercase;
      gp_str_func.lowercase = gm_lowercase;
      gp_str_func.compare = gm_compare;
      gp_str_func.trim_right = gm_trim_right;
#endif /*USE_GP_MEM*/

/*Font initialize*/
InitializeFont ();
GpKernelInitialize ();
GpKernelStart ();
   GpAppExit ();
   while (1) {};
   }
```

### 2.7.5.9 SRC\GP32\INPUT.C

Este módulo contiene las funciones de lectura del joystick y de los botones de la consola GP32 y su mapeo como interfaces del MAME.

La lista de funciones más interesantes del módulo es la siguiente:

- `osd_key_pressed()`: Simulación de las teclas del PC mediante los mandos de la consola GP32. Están mapeadas la tecla ESC (como botones L+R simultáneos), la tecla para resetear el juego (como botones START+SELECT simultáneos), la tecla de pausa (como botón R), insertar créditos (botón L) y comenzar partida (botón START). Las demás teclas no están mapeadas y devuelven siempre el estado no activo.

- `osd_joy_pressed()`: Simulación del joystick del PC mediante los mandos de la consola GP32. Se simulan los dos primeros joysticks del PC de la siguiente manera:
  - Direcciones del joystick: Mediante joypad de la consola: izquierda, derecha, arriba y abajo. En el caso de tener la pantalla de visualización rotada, se realiza también la rotación de los mandos para que se corresponda con la pantalla.
  - Botones A,B,C,D,E,F del joystick: Respectivamente botón A, botón B, botón SELECT, botón L, botón START y botón R de la GP32.

- `osd_analogjoy_read()`: Simulación de un joystick analógico del PC mediante los mandos digitales del joypad de la consola GP32.

- `osd_trak_read()`: Simulación del ratón del PC mediante los mandos digitales del joypad de la consola GP32.

El código fuente de estas funciones se muestra a continuación:

```
int osd_key_pressed(int keycode)
{
if (keycode > OSD_MAX_KEY)
{
      switch (keycode)
            {
                        case OSD_KEY_CANCEL:
                              return ((ExKey & GPC_VK_FL) &&
                                    (ExKey & GPC_VK_FR));

                        case OSD_KEY_SHOW_GFX:
                        return 0;

                  case OSD_KEY_RESET_MACHINE:
                        return ((ExKey & GPC_VK_START) &&
                                    (ExKey & GPC_VK_SELECT));

                        case OSD_KEY_CHEAT_TOGGLE:
                        return 0;

                        case OSD_KEY_JOY_CALIBRATE:
                        return 0;
                        case OSD_KEY_FRAMESKIP:
                              return 0;
                        case OSD_KEY_THROTTLE:
```

```
                              return 0;
                        case OSD_KEY_SHOW_FPS:
                              return 0;
                        case OSD_KEY_SHOW_PROFILE:
                              return 0;
                        case OSD_KEY_CONFIGURE:
                        return 0;
                        case OSD_KEY_ON_SCREEN_DISPLAY:
                        return 0;
                        case OSD_KEY_DEBUGGER:
                        return 0;

                        case OSD_KEY_PAUSE:
                        case OSD_KEY_UNPAUSE:
                        ExKey=GpKeyGet();
                        return (ExKey & GPC_VK_FR);
            default:
                  return 0;
            }
      }

if (keycode==OSD_KEY_3) {
      return (ExKey & GPC_VK_FL);
}

if (keycode==OSD_KEY_1) {
      return (ExKey & GPC_VK_START);
}
return 0;
}


int osd_joy_pressed(int joycode)
{
/* which joystick are we polling? */
if (joycode < OSD_JOY_LEFT || joycode >= OSD_JOY3_LEFT)
      return 0;

      ExKey=GpKeyGet();

switch (joycode)
{
      case OSD_JOY_LEFT:
            if (!gp32_rotate)
                  return (ExKey & GPC_VK_LEFT);
            else
                  return (ExKey & GPC_VK_UP);
            break;
      case OSD_JOY_RIGHT:
            if (!gp32_rotate)
                  return (ExKey & GPC_VK_RIGHT);
            else
                  return (ExKey & GPC_VK_DOWN);
            break;
      case OSD_JOY_UP:
            if (!gp32_rotate)
                  return (ExKey & GPC_VK_UP);
            else
                  return (ExKey & GPC_VK_RIGHT);
            break;
```

```
case OSD_JOY_DOWN:
      if (!gp32_rotate)
            return (ExKey & GPC_VK_DOWN);
      else
            return (ExKey & GPC_VK_LEFT);
      break;
case OSD_JOY_FIRE1:
      if (!gp32_rotate)
            return (ExKey & GPC_VK_FA);
      else
            return ((ExKey & GPC_VK_FA) ||
                        (ExKey & GPC_VK_START));
      break;
case OSD_JOY_FIRE2:
      return (ExKey & GPC_VK_FB);
      break;
case OSD_JOY_FIRE3:
      return (ExKey & GPC_VK_SELECT);
      break;
case OSD_JOY_FIRE4:
      return (ExKey & GPC_VK_FL);
      break;
case OSD_JOY_FIRE5:
      return (ExKey & GPC_VK_START);
      break;
case OSD_JOY_FIRE6:
      return (ExKey & GPC_VK_FR);
      break;
case OSD_JOY_FIRE7:
      break;
case OSD_JOY_FIRE8:
      break;
case OSD_JOY_FIRE9:
      break;
case OSD_JOY_FIRE10:
      break;
case OSD_JOY_FIRE:
      return ( (ExKey & GPC_VK_FA) ||
                  (ExKey & GPC_VK_FB) ||
                  (ExKey & GPC_VK_SELECT) ||
            (ExKey & GPC_VK_FL) ||
                  (ExKey & GPC_VK_START) ||
                  (ExKey & GPC_VK_FR) );
      break;
case OSD_JOY2_LEFT:
      if (!gp32_rotate)
            return (ExKey & GPC_VK_LEFT);
      else
            return (ExKey & GPC_VK_UP);
      break;
case OSD_JOY2_RIGHT:
      if (!gp32_rotate)
            return (ExKey & GPC_VK_RIGHT);
      else
            return (ExKey & GPC_VK_DOWN);
      break;
case OSD_JOY2_UP:
      if (!gp32_rotate)
            return (ExKey & GPC_VK_UP);
      else
```

```
                  return (ExKey & GPC_VK_RIGHT);
            break;
      case OSD_JOY2_DOWN:
            if (!gp32_rotate)
                  return (ExKey & GPC_VK_DOWN);
            else
                  return (ExKey & GPC_VK_LEFT);
            break;
      case OSD_JOY2_FIRE1:
            if (!gp32_rotate)
                  return (ExKey & GPC_VK_FA);
            else
                  return ((ExKey & GPC_VK_FA) ||
                              (ExKey & GPC_VK_START));
            break;
      case OSD_JOY2_FIRE2:
            return (ExKey & GPC_VK_FB);
            break;
      case OSD_JOY2_FIRE3:
            return (ExKey & GPC_VK_SELECT);
            break;
      case OSD_JOY2_FIRE4:
            return (ExKey & GPC_VK_FL);
            break;
      case OSD_JOY2_FIRE5:
            return (ExKey & GPC_VK_START);
            break;
      case OSD_JOY2_FIRE6:
            return (ExKey & GPC_VK_FR);
            break;
      case OSD_JOY2_FIRE7:
            break;
      case OSD_JOY2_FIRE8:
            break;
      case OSD_JOY2_FIRE9:
            break;
      case OSD_JOY2_FIRE10:
            break;
      case OSD_JOY2_FIRE:
            return ( (ExKey & GPC_VK_FA) ||
                        (ExKey & GPC_VK_FB) ||
                        (ExKey & GPC_VK_SELECT) ||
                  (ExKey & GPC_VK_FL) ||
                        (ExKey & GPC_VK_START) ||
                        (ExKey & GPC_VK_FR) );
            break;
}
return 0;
}
```

```
void osd_analogjoy_read(int *analog_x, int *analog_y)
{
if (!gp32_rotate) {
      if(ExKey & GPC_VK_LEFT) pos_analog_x-=10;
      if(ExKey & GPC_VK_RIGHT) pos_analog_x+=10;
      if( (!(ExKey & GPC_VK_LEFT)) &&
                  (!(ExKey & GPC_VK_RIGHT)) ) pos_analog_x=0;
      if(ExKey & GPC_VK_UP) pos_analog_y-=10;
      if(ExKey & GPC_VK_DOWN) pos_analog_y+=10;
      if( (!(ExKey & GPC_VK_UP)) &&
                  (!(ExKey & GPC_VK_DOWN)) ) pos_analog_y=0;
} else {
      if(ExKey & GPC_VK_UP) pos_analog_x-=10;
      if(ExKey & GPC_VK_DOWN) pos_analog_x+=10;
      if( (!(ExKey & GPC_VK_UP)) &&
                  (!(ExKey & GPC_VK_DOWN)) ) pos_analog_x=0;
      if(ExKey & GPC_VK_RIGHT) pos_analog_y-=10;
      if(ExKey & GPC_VK_LEFT) pos_analog_y+=10;
      if( (!(ExKey & GPC_VK_LEFT)) &&
                  (!(ExKey & GPC_VK_RIGHT)) ) pos_analog_y=0;
}

if (pos_analog_x<-128) pos_analog_x=-128;
if (pos_analog_x>128) pos_analog_x=128;
if (pos_analog_y<-128) pos_analog_y=-128;
if (pos_analog_y>128) pos_analog_y=128;

*analog_x = pos_analog_x;
*analog_y = pos_analog_y;
}


void osd_trak_read(int *deltax,int *deltay)
{
*deltax = *deltay = 0;
if (!gp32_rotate) {
      if(ExKey & GPC_VK_LEFT) *deltax=-20;
      if(ExKey & GPC_VK_RIGHT) *deltax=20;
      if(ExKey & GPC_VK_UP) *deltay=20;
      if(ExKey & GPC_VK_DOWN) *deltay=-20;
} else {
      if(ExKey & GPC_VK_UP) *deltax=-20;
      if(ExKey & GPC_VK_DOWN) *deltax=20;
      if(ExKey & GPC_VK_RIGHT) *deltay=20;
      if(ExKey & GPC_VK_LEFT) *deltay=-20;
}
}
```

### 2.7.5.10 SRC\GP32\SOUND.C

Este módulo contiene las funciones de reproducción de sonido. La versión del MAME para MS-DOS usaba las librerías de audio *Allegro*. Para el *port* a GP32 se ha elegido realizar estas funciones a través de una versión ligeramente modificada del código *GpSoundBuf* realizado por *Christian Nowak*, que se fundamenta en un buffer de sonido circular actualizado con una función ejecutada periódicamente mediante el *timer* de la consola.

El sonido del MAME 0.34 consiste en 16 canales de sonido monoaural / estéreo de 8 ó 16 bit de resolución, cada uno de estos canales de frecuencia de muestreo variable.

Para el port a GP32 se ha decidido restringir este sonido a monoaural en resolución de 8 bit, para no gastar demasiado tiempo de CPU ni demasiada memoria en el sonido.

En el código de la librería se ha utilizado un array de estructuras que mantiene en tiempo real el estado de cada uno de los 16 canales de sonido.

El código es el siguiente:

```
/* audio related stuff */
#define NUMVOICES 16
#define SAMPLE_NUMBER 1600

struct lpWaves {
signed char lpData[SAMPLE_NUMBER];
int len; /* length of sound buffer */
int freq; /* frequency of sound buffer */
int active; /* 1 active, 0 inactive */
int volume; /* volume 0 - 64 */
int loop; /* loop 1 yes, 0 no */
unsigned int pos; /* position into the sample */
unsigned int step; /* step of the sample */
};
static struct lpWaves lpWave[NUMVOICES];
```

Para cada uno de los canales de sonido (`lpWave[número de canal]`) se almacena la siguiente información:

- **lpData**: Es la muestra de sonido en formato con signo (*signed*). Como máximo el tamaño de la muestra es de 1600 bytes.
- **len**: Es la longitud de la muestra de sonido.
- **freq**: Es la frecuencia de la muestra (valores típicos: 11 KHz, 22 KHz, etc). Cada uno de los canales de sonido puede tener muestras de diferente frecuencia de muestreo.
- **active**: Este *flag* identifica si la muestra debe de estar sonando en un momento determinado (1) ó no (0). Dinámicamente se puede cambiar este *flag* (lo mismo que el resto de los campos de la estructura).
- **volume**: Es el volumen de la muestra de sonido. Los valores permitidos son enteros entre 0 (sin volumen) y 64 (volumen máximo).
- **loop**: Identifica si la muestra debe sonar cíclicamente, es decir que después de su reproducción debe de sonar interminablemente hasta que se ponga el *flag* active a 0

desde el emulador. En caso contrario, al terminar la reproducción el *flag* ‘active’ se pone automáticamente a 0.

- **pos**: Es la última posición de la muestra que ha sido reproducida.
- **step**: Este valor identifica el paso de frecuencias necesario para reproducir el sonido. En el caso de que la muestra sea de 22 KHz ó similar el paso será 0 y no será necesario realizar este escalado. En caso de frecuencias diferentes, será necesario adaptar la muestra a los 22 KHz que es la frecuenta de reproducción final del sonido.

La frecuencia final de reproducción del sonido es de 22 KHz 8 bit monoaural. A partir de la información de cada uno de los canales de sonido, la función `gp32_sound_mixer()` realiza la mezcla del sonido en tiempo real. Esta función es activada periódicamente mediante el timer de la consola GP32. Esto se realiza mediante el código *GpSoundBuf* de forma automática.

La mezcla del sonido consiste en los siguientes pasos (se parte de que las muestras de sonido están en formato con signo, *signed*):

1. Lo primero a realizar previamente es el escalado de las muestras. Esto se realiza multiplicando el valor de las muestras por el valor del volumen de cada canal y dividiendo por el volumen máximo (64). Con ello se consigue tener las muestras con el volumen adecuado para posteriormente mezclarlas.
2. Mezclar el sonido supone sumar las muestras de cada uno de los canales de sonido que están en formato con signo. Una vez se consigue la muestra mezcla de todos los canales de sonido, es necesario escalar ésta para que ocupe un byte. Se deben de mezclar byte a byte hasta llenar el buffer de sonido de salida.
3. En el caso de muestras de origen con distinta frecuencia de reproducción, es necesario realizar un muestreo para que la mezcla de salida resultante de 22 KHz contenga en su interior las muestras de origen, cada una con su frecuencia original. Esto se consigue repitiendo muestras en el caso de que la frecuencia sea inferior a 22 KHz ó saltándose muestras en el caso de que la frecuencia sea mayor de 22 KHz.
4. Para la reproducción final del sonido en la GP32, se debe de convertir la muestra resultante a formato sin signo, que es el formato de sonido que reproduce la consola GP32 en modo monoaural de 8 bit. Para ello se debe de realizar la operación `sample^0x80`, para modificar el bit más significativo del byte de cada muestra con signo.

Como el consumo de CPU de esta función es muy grande, se han realizado gran cantidad de optimizaciones. En función de la frecuencia de las muestras origen a mezclar, se realizan diversos trucos para evitar las operaciones matemáticas más costosas (multiplicación, división y módulo de la división). Por ejemplo dividir un byte entre 8 (mix/8) es lo mismo que desplazar 3 bits a la derecha (mix>>3).

El código fuente de **gp32_sound_mixer()** se muestra a continuación:

```
void gp32_sound_mixer ( void * userdata, unsigned char * stream, int len
) {

      register int channel;
register int i;
signed long mix;
int t;

/* Sound Enable ? */
if (!sound_enable) {
      /* Buffer Set to Zero */
      for (i=0;i<len;i++)
            stream[i]=0x80;
      return;
}

/* Initialize position in origin samples to 0 */
t=0; /* Calculate max channels */
for (channel=0;channel<NUMVOICES;channel++)
      {
      /* Channel active ? */
      if (!lpWave[channel].active)
            continue;

      /* initial position of the sample */
      lpWave[channel].pos=0;

      /* Calculate step of the sample */
      if(lpWave[channel].freq>20000 &&
                  lpWave[channel].freq<25000) {
            /* Sound Updated ? */
            if(mixer_count==mixer_count_ok) return;
            lpWave[channel].step=0; /* pos++ */
      } else {
            lpWave[channel].step=1;
      }

      t++;
      }
if (t==0) return;
if (t>chn_max)
      chn_max=t;

/* Fill Sound Buffer */
for(i=0;i<len;i++) {

      /* Initialize value of mixed sample to 0 */
      mix=0;

      /* For each of the channels */
      for (channel=0;channel<NUMVOICES;channel++) {

            /* Channel active ? */
            if (!lpWave[channel].active)
                  continue;

                  if (!lpWave[channel].step) {
                  /* Update sample mix */
```

```
                        mix+=( lpWave[channel].lpData
                              [lpWave[channel].pos] *
                              lpWave[channel].volume)>>6;
                  /* Increase sample position */
                  lpWave[channel].pos++;
                  if (lpWave[channel].pos ==
                              lpWave[channel].len-1)
                        lpWave[channel].pos=0;
                  } else {
                  /* Update sample position*/
                  lpWave[channel].pos = (i *
                              lpWave[channel].freq) /SAMPLE_RATE;
                  if (lpWave[channel].pos >
                              lpWave[channel].len)
                        lpWave[channel].pos %=
                                    lpWave[channel].len;
                  else if (lpWave[channel].pos ==
                              lpWave[channel].len)
                        lpWave[channel].pos=0;
                  /* Update sample mix */
                  mix+=
                              (lpWave[channel].lpData
                              [lpWave[channel].pos] *
                              lpWave[channel].volume)>>6;
            }

      }

      /* Scale mixed sample and assign */
      /*stream[i]= (mix/chn_max)^0x80;*/
      switch (chn_max) {
            case 1: stream[i]=mix^0x80; break;
            case 8: stream[i]=(mix>>3)^0x80; break;
            case 4: stream[i]=(mix>>2)^0x80; break;
            case 2: stream[i]=(mix>>1)^0x80; break;
            case 16: stream[i]=(mix>>4)^0x80; break;
            default: stream[i]= (mix/chn_max)^0x80;
break;
      }

}

/* Update status */
for (channel=0;channel<NUMVOICES;channel++) {

      /* Channel active ? */
      if (!lpWave[channel].active)
            continue;

      /* If no loop -> Not active */
      if (!lpWave[channel].loop) {
            lpWave[channel].active=0;
      }

}

/* Sound Updated ! */
mixer_count_ok = mixer_count;
}
```

La siguiente función **osd_update_audio()** va ordenando la actualización del sonido. Para ello hace una actualización de la variable `mixer_count`. Esta variable es leída desde la función de evento `gp32_sound_mixer()`, por lo que sólo funcionará el mezclador cada vez que se llame `a osd_update_audio()`. Con ello se consigue que el procesado del sonido no consuma demasiada CPU. En el caso de que la emulación del MAME consuma demasiado, se evita actualizar el sonido para no ralentizarlo aún más. Si la GP32 no puede hacer funcionar el juego en tiempo real, lo que ocurrirá es que el sonido se entrecortará, pero siempre intentando que el juego funcione en tiempo real:

```
void osd_update_audio(void)
{

/* Sound Enable ? */
if (Machine->sample_rate == 0) return;

/* Sound updated ? */
if (mixer_count!=mixer_count_ok) return;

/* Sound updated ! */
mixer_count++;
mixer_count=mixer_count & 0xff;
/*mixer_count=mixer_count%256;*/
}
```

La siguiente función **playsample()** ordena la reproducción de un sonido en un canal determinado. Realmente actualiza la estructura del canal en cuestión, para que posteriormente el mezclador de sonido mezcle el canal en la reproducción real. Internamente sube un poco el volumen de las mezclas para que los sonidos de volumen más bajo sean audibles con los pequeños altavoces de la consola. Además en el caso de muestras de 16 bit, realiza su conversión a 8 bit (descarta los 8 primeros bits de la muestra).

```
static void playsample(int channel,signed char *data,int len,int
freq,int volume,int loop, int pan, int bits)
{
int i;
if (Machine->sample_rate == 0 || channel >= NUMVOICES)
            return;

if (len<1 || freq < 16 || volume < 4) {
      return;
}

lpWave[channel].active=0;

/* backwards compatibility with old 0-255 volume range */
if (volume > 100) volume = volume * 25 / 255;

if (volume > 0) {
      volume = (volume<<6)/100;
      if (pan != OSD_PAN_CENTER) volume=volume>>1;
      if (volume>0) volume+=20;
      if (volume>64) volume=64;
}

lpWave[channel].volume=volume;

if (bits==8) {
```

```
      for (i=0;i<len;i++)
            lpWave[channel].lpData[i]= data[i];
      lpWave[channel].len=len;
} else {
      for (i=0;i<len;i++) {
            i++; /* ignore low 8 bits */
            lpWave[channel].lpData[i]= data[i];
      }
      lpWave[channel].len=len>>1;
}

lpWave[channel].freq=freq;
lpWave[channel].loop=loop;
lpWave[channel].active=1;
}
```

Hay determinados juegos que varían dinámicamente la frecuencia ó el volumen de las muestras de cada canal. Por ello se hace necesario la siguiente función **`osd_adjust_sample()`**, para cambiar el *setting* de uno de los canales de sonido:

```
void osd_adjust_sample(int channel,int freq,int volume)
{

if (Machine->sample_rate == 0 || channel >= NUMVOICES)
            return;

/* backwards compatibility with old 0-255 volume range */
if (volume > 100) volume = volume * 25 / 255;

if (freq != -1)
      lpWave[channel].freq=freq;
if (volume != -1) {
      volume = (volume<<6)/100;
      if (volume>0) volume+=20;
      if (volume>64) volume=64;
      lpWave[channel].volume = volume;
}
}
```

Por último hay algunas funciones adicionales bastante interesantes, como las siguientes, cuya utilidad es obvia: parar un canal de sonido, reactivar un canal de sonido y otra función que indica si un canal está activo:

```
void osd_stop_sample(int channel)
{
      if (Machine->sample_rate == 0 || channel >= NUMVOICES)
            return;
lpWave[channel].active = 0;
}


void osd_restart_sample(int channel)
{
if (Machine->sample_rate == 0 || channel >= NUMVOICES)
            return;
lpWave[channel].active = 1;
}


int osd_get_sample_status(int channel)
{
if (Machine->sample_rate == 0 || channel >= NUMVOICES)
            return -1;
return !lpWave[channel].active;
}
```

### 2.7.5.11 SRC\GP32\VÍDEO.C

Este módulo implementa las siguientes funcionalidades:

- Librería de salida de vídeo.
- Limitador de velocidad del emulador (sincronización).
- Manejo de la paleta de colores.
- Reserva y liberación de memoria para bitmaps y buffers de vídeo.

Las funciones más interesantes del módulo son las siguientes:

- `gp32_vídeo_init()`: Inicializa el modo gráfico de la consola: 320x240 píxeles con 8 bit de resolución de color.
- `gp32_mame_palette()`: Inicializa la paleta del emulador.
- `update_screen_gp32()`: Actualiza la pantalla de la consola.
- `gp32_adjust_display()`: Inicializa una serie de variables para mostrar correctamente la pantalla de cada juego en la pantalla de la consola (320x240 píxeles): Centrado y recorte de la imagen.
- `gp32_text_out()`, `gp32_text_out_int()`, `gp32_gamelist_text_out()`, `gp32_text_pause()`, `gp32_gamelist_zero()`, `gp32_text_log()`, `gp32_text_log_int()`: Funciones utilizadas para sacar texto por pantalla.
- `osd_new_bitmap()`: Función de reserva de memoria e inicialización de una imagen bitmap.
- `osd_clearbitmap()`: Limpia un determinado bitmap.
- `osd_free_bitmap()`: Libera la memoria de un determinado bitmap.
- `osd_create_display()`: Inicializa el acceso a la pantalla de la GP32, reserva memoria para el bitmap que actuará como framebuffer de la pantalla, etc. Internamente llama a `osd_new_bitmap()`, gp32_adjust_display(), etc.
- `osd_close_display()`: Libera la memoria del framebuffer de la pantalla. Internamente llama a `osd_free_bitmap()`.
- `osd_allocate_colors()`: Inicializa la paleta de colores de un determinado juego.
- `osd_modify_pen()`: Modifica un determinado color de la paleta. Realmente no lleva a cabo la modificación de la paleta, sino que el color queda marcado para ser modificado.
- `osd_get_pen()`: Consigue los valores R,G,B de un determinado color de la paleta.
- `gp32_clear_screen()`: Limpia la pantalla de la consola GP32.
- `osd_update_display()`: Es la función principal de la actualización de pantalla: Lleva a cabo la sincronización de la emulación (si funciona demasiado rápido, realiza la pausa necesaria para que el juego funcione en tiempo real), modifica la paleta si es necesario y llama a la función de actualización de la pantalla.

La función **gp32_vídeo_init()** realiza la inicialización del modo gráfico elegido de la consola: 320 píxeles de resolución horizontal por 240 píxeles de resolución vertical con 256 colores (8 bit de profundidad de color). Además la inicialización de los dos *buffers* virtuales donde se actualizará la pantalla del emulador:

```
/* GP32 Vídeo Init */
void gp32_vídeo_init(void) {

int i=0;

/* GP32 Vídeo Init */
GpGraphicModeSet(8,0);  /* 320x240, 256 colores */

for( i=0; i<2; i++ ) {
      GpLcdSurfaceGet( &gpDraw[i],i );
      GpRectFill( NULL, &gpDraw[i], 0,0, gpDraw[i].buf_w,
                  gpDraw[i].buf_h,0x0 );
}
nflip=0;
GpSurfaceSet( &gpDraw[nflip] );
GpLcdEnable();
GpSurfaceFlip( &gpDraw[nflip] );


}
```

La función **gp32_mame_palette()** inicializa los colores usados por el MAME que no pueden ser modificados, sea cual sea la paleta de colores del juego a emular. El color 0 es negro y el color 1 es blanco.

```
void gp32_mame_palette(void) {

GP_PALETTEENTRY colorgp;

/* Color #0 is black */
colorgp = GP_RGB24(0,0,0);
      GpPaletteEntryChange(0,1,&colorgp,0);

/* Color #1 is white */
colorgp = GP_RGB24(255,255,255);
      GpPaletteEntryChange(1,1,&colorgp,0);

}
```

La función **update_screen_gp32()** es la que actualiza la pantalla de la consola. Dibuja la pantalla y la visualiza en la consola. Para ello utiliza dos *buffers* virtuales que se van mostrando alternativamente. Mientras se dibuja la pantalla, en la pantalla real se esta mostrando la pantalla dibujada anteriormente. Al terminar de dibujar se muestra la nueva imagen dibujada.

El dibujado de la pantalla se realiza accediendo directamente al array de vídeo de cada uno de los buffers y escribiendo en ellos directamente. Ya se vio en anteriores capítulos que la ordenación de los píxeles en un array de pantalla de la consola GP32 es diferente a la de un buffer VGA estándar de un PC. Por ello hay que dibujar los píxeles en el array de pantalla a partir del buffer de memoria de vídeo del MAME siguiendo la fórmula:

```
buffer_scr[239-y+x*240] = scrbitmap->line[y][x];
```

- En el PC el origen de coordenadas (0,0) esta en la esquina superior izquierda de la pantalla, y la ordenación de los píxeles en el *array* de vídeo es por filas desde arriba hasta abajo de la imagen.

- En la GP32 el origen de coordenadas es la esquina inferior izquierda de la pantalla y además la orientación de la pantalla esta girada, es decir la ordenación de los píxeles en el *array* de vídeo es por columnas desde la izquierda hasta la derecha de la imagen.

Para evitar las operaciones de multiplicación que son bastante lentas en la consola GP32, se realiza una rutina de dibujado bastante más complicada, en la que se calculan los datos a priori y se limitan al máximo los cálculos matemáticos.

Además la función debe de dibujar la pantalla en su orientación normal y también girada, ya que desde el emulador en la consola se puede cambiar de orientación pulsando el botón L. Esto es así porque determinados juegos tienen una resolución de pantalla que se adapta mejor a la pantalla de la consola si se gira (más píxeles de resolución vertical que horizontal).

El código de la función **update_gp32_screen()** es el siguiente:

```
/* Update GP32 Screen */
void update_screen_gp32(void)
{
register unsigned char *buffer_scr;
register unsigned char *buffer_mem;
register unsigned char *buffer_mem_line;
register unsigned int buffer_mem_offset;
register int x,y,px,pxi,py;

buffer_scr = gpDraw[nflip].o_buffer;
buffer_mem = scrbitmap->line[gp32_yoffset]+gp32_xoffset;
buffer_mem_offset = (unsigned int)(scrbitmap->line[1] –
                              scrbitmap->line[0]);

if (!gp32_rotate) {

      y=0;
      pxi=gp32_xscreen*240;
      py=239-gp32_yscreen;
      while (y < gp32_height) {
            buffer_mem_line=buffer_mem;
            buffer_mem+=buffer_mem_offset;
            x=0;
```

```
            px=pxi;
            while (x < gp32_width) {
                  buffer_scr[px+py] = *buffer_mem_line++;
                  x++;
                  px+=240;
            }
            y++;
            py--;
      }
} else {
      y=0;
      pxi=239-gp32_xscreen;
      py=(319-gp32_yscreen)*240;
      while (y < gp32_height) {
            buffer_mem_line=buffer_mem;
            buffer_mem+=buffer_mem_offset;
            x=0;
            px=pxi;
            while (x < gp32_width) {
                  buffer_scr[px+py] = *buffer_mem_line++;
                  x++;
                  px--;
            }
            y++;
            py-=240;

      }
}

/* Flip Vídeo Surface */
GpSurfaceFlip( &gpDraw[nflip] );

/* Change Vídeo Buffer Number */
nflip=!nflip;
}
```

La función **gp32_adjust_display()** analiza el tamaño de pantalla del juego a emular y elige la mejor manera de visualizar la imagen. Para ello es necesario centrar la imagen en la pantalla y dependiendo del juego puede ser necesario recortar la imagen. Todo ello depende de la resolución de pantalla del juego original. En función de ello actualiza una serie de variables:

- **gp32_xoffset**: Si la imagen no esta recortada el valor es 0, en caso contrario indica píxeles a recortar de resolución horizontal.
- **gp32_yoffset**: Si la imagen no esta recortada, tiene como valor 0. En caso contrario contiene el número de píxeles a recortar de resolución vertical.
- **gp32_xscreen**: Posición X de la pantalla de la consola donde empezar a dibujar. Es necesario poner un valor mayor de 0 si el tamaño de pantalla del juego es menor en resolución horizontal que el tamaño de pantalla de la consola.
- **gp32_yscreen**: Posición Y de la pantalla de la consola donde empezar a dibujar. Es necesario si el tamaño de pantalla del juego en resolución vertical es menor que la resolución de la consola.
- **gp32_width**: Ancho de la imagen del juego.
- **gp32_weidth**: Alto de la imagen del juego.

La variable **gp32_rotate** también indica si la pantalla debe de aparecer rotada o no. Si la pantalla no esta rotada, la resolución física de la pantalla de la consola es de 320 píxeles de resolución horizontal por 240 de resolución vertical. En el caso que la pantalla esté rotada es necesario realizar los cálculos pensando en 240 píxeles de resolución horizontal y 320 de vertical.

El listado de la función es el siguiente:

```
void gp32_adjust_display(void) {

/* Image not rotated */
if (gp32_rotate==0) {

      gp32_rotate=0;
      gp32_xoffset=0;
      gp32_yoffset=0;
      gp32_xscreen=0;
      gp32_yscreen=0;
      gp32_width=320;
      gp32_height=240;

      if (game_width>320)
            gp32_xoffset = (game_width-320) / 2;

      if (game_width<320)
            gp32_width = game_width;

      if (game_height>240) {
            gp32_yoffset = (game_height-240) / 2;
            if (game_width==336 && game_height==256)
                        /* For Tetris */
                  gp32_yoffset-=8;
            if (game_width==256 && game_height==512)
                        /*For SlapFight, TigerHeli, TwinCobra*/ {
                  gp32_xoffset+=8;
                  gp32_yoffset+=112;
            }
      }

      if (game_height<240)
```

```
            gp32_height = game_height;

      if (game_width<320)
            gp32_xscreen = (320-game_width) / 2;

      if (game_height<240)
            gp32_yscreen = (240-game_height) / 2;

} else {
      /* Image has to be rotated */
      gp32_rotate=1;
      gp32_xoffset=0;
      gp32_yoffset=0;
      gp32_xscreen=0;
      gp32_yscreen=0;
      gp32_width=240;
      gp32_height=320;

      if (game_width>240)
            gp32_xoffset = (game_width-240) / 2;

      if (game_width<240)
            gp32_width = game_width;

      if (game_height>320) {
            gp32_yoffset = (game_height-320) / 2;
            if (game_width==256 && game_height==512)
                  /*For SlapFight, TigerHeli, Twin Cobra*/ {
                  gp32_xoffset+=8;
                  gp32_yoffset+=96;
            }
      }

      if (game_height<320)
            gp32_height = game_height;

      if (game_width<240)
            gp32_xscreen = (240-game_width) / 2;

      if (game_height<320)
            gp32_yscreen = (320-game_height) / 2;
}
}
```

Entre las funciones de impresión de texto por pantalla, se muestra por ejemplo el contenido de la función **gp32_text_out():**

```
/* Debug information function*/
void gp32_text_out(int x, int y, char *texto) {
GpTextOut( NULL, &gpDraw[nflip],x,y,texto, 0x01 );
}
```

La función **osd_new_bitmap()** reserva memoria para una imagen de tipo bitmap. Internamente se realiza la reserva de memoria para una de mayor tamaño, se deja un margen de 8 píxeles alrededor de la imagen, ya que muchos drivers del MAME no hacen el *clipping* de forma exacta y en caso necesario se podría producir una escritura de vídeo en el bitmap fuera de su tamaño con el consiguiente error de acceso inválido a memoria.

Además se escribe un array de punteros para acceder de forma más sencilla a cada una de las líneas de resolución horizontal de la imagen. Por último también redondea el tamaño de las imágenes para que su tamaño sea múltiplo de cuatro bytes (*int32*), para mejorar el alineamiento de los datos en la memoria.

Toda la información queda recopilada en una estructura *osd_bitmap*.

```
struct osd_bitmap *osd_new_bitmap(int width,int height,int depth)
{
struct osd_bitmap *bitmap;

if (Machine->orientation & ORIENTATION_SWAP_XY) {
      int temp;
      temp = width;
      width = height;
      height = temp;
}

if ((bitmap = gm_zi_malloc(sizeof(struct osd_bitmap)))
            != 0) {
      int i,rowlen,rdwidth;
      unsigned char *bm;
      int safety;

      if (width > 32) safety = 8;
      else safety = 0;
            /* don't create the safety area
             for GfxElement bitmaps */

      if (depth != 8 && depth != 16) depth = 8;

      bitmap->depth = depth;
      bitmap->width = width;
      bitmap->height = height;

      rdwidth = (width + 7) & ~7;
            /* round width to a quadword */
      rowlen =     (rdwidth + 2 * safety) *
                  sizeof(unsigned char);

      if ((bm = gm_zi_malloc((height + 2 * safety) *
                  rowlen)) == 0) {
            gp32_text_log(
                        "osd_new_bitmap(): Out of Memory");
            gm_free(bitmap);
            return 0;
      }

      if ((bitmap->line = gm_zi_malloc(height *
                  sizeof(unsigned char *))) == 0) {
            gp32_text_log(
                        "osd_new_bitmap(): Out of Memory");
```

```
            gm_free(bm);
            gm_free(bitmap);
            return 0;
      }

      for (i = 0;i < height;i++)
            bitmap->line[i] = &bm[(i + safety) * rowlen
                        + safety];
      bitmap->_private = bm;
      osd_clearbitmap(bitmap);
}

if(bitmap==0)
      gp32_text_log("osd_new_bitmap(): Out of Memory");

return bitmap;
}
```

La función que limpia el contenido de un bitmap: **osd_clear_bitmap()**

```
/* set the bitmap to black */
void osd_clearbitmap(struct osd_bitmap *bitmap)
{
int i,j;

for (i = 0;i < bitmap->height;i++)
{
      for (j = 0;j<bitmap->width;j++)
            bitmap->line[i][j]=BACKGROUND;
}
}
```

La función que libera la memoria de un bitmap: **osd_free_bitmap()**

```
void osd_free_bitmap(struct osd_bitmap *bitmap)
{
if (bitmap)
{
      gm_free(bitmap->line);
      gm_free(bitmap->_private);
      gm_free(bitmap);
}
}
```

La función **`osd_create_display()`** inicializa el buffer de pantalla interno del MAME:

- Cambio de orientación de pantalla si así lo requiere el juego.
- Llama a `gp32_adjust_display()` para inicializar una serie de variables necesarias para mostrar la pantalla correctamente en la GP32 (centrado y recorte de imagen).
- Reserva de memoria para el bitmap: `osd_new_bitmap()`.
- Marca la paleta para ser actualizada.
- Llama a `adjust_display()` para actualizar los parámetros internos del MAME en cuanto a centrado y recorte de la imagen.
- Marca la función `update_screen_gp32()` para que sea la función encargada de actualizar la pantalla.
- Devuelve el *bitmap* de pantalla.

```
struct osd_bitmap *osd_create_display(int width,int height,int attributes)
{
   int     i;
   game_width = width;
   game_height = height;

   if (Machine->orientation & ORIENTATION_SWAP_XY)
   {
         int temp;
         temp = game_width;
         game_width = game_height;
         game_height = temp;
   }
   gp32_adjust_display();

   scrbitmap = osd_new_bitmap(width,height,8);
   if (!scrbitmap) return 0;

   for (i = 0;i < 256;i++)
   {
         dirtycolor[i] = 1;
   }
   dirtypalette = 1;

   /* center display based on visible area */
   {
         struct rectangle vis = Machine->drv->visible_area;
         adjust_display (vis.min_x, vis.min_y, vis.max_x,
         vis.max_y);
   }

   update_screen = update_screen_gp32;

   return scrbitmap;
}
```

La función **`osd_close_display()`** libera la memoria de buffer de vídeo interno del MAME:

```
/* shut up the display */
void osd_close_display(void)
{
if (scrbitmap)
{
      osd_free_bitmap(scrbitmap);
      scrbitmap = NULL;
}
}
```

La función **`osd_allocate_colors()`** inicializa la paleta de colores del juego:

```
void osd_allocate_colors(unsigned int totalcolors,const unsigned char
*palette,unsigned short *pens)
{

int i;

/* initialize the palette */
for (i = 0;i < 256;i++)
      current_palette[i][0] = current_palette[i][1] =
                  current_palette[i][2] = 0;

if (totalcolors >= 255)
{
      int bestblack,bestwhite;
      int bestblackscore,bestwhitescore;

      for (i = 0;i < totalcolors;i++)
            pens[i] = i;

      bestblack = bestwhite = 0;
      bestblackscore = 3*255*255;
      bestwhitescore = 0;
      for (i = 0;i < totalcolors;i++)
      {
            int r,g,b,score;

            r = palette[3*i];
            g = palette[3*i+1];
            b = palette[3*i+2];
            score = r*r + g*g + b*b;

            if (score < bestblackscore)
            {
                  bestblack = i;
                  bestblackscore = score;
            }
            if (score > bestwhitescore)
            {
                  bestwhite = i;
                  bestwhitescore = score;
            }
      }
}
else
```

```
{
      /* reserve color 1 for the user interface text */
      current_palette[1][0] = current_palette[1][1] =
                  current_palette[1][2] = 0xff;

      /* fill the palette starting from the end, so we mess
            up badly written */
      /* drivers which don't go through Machine->pens[] */
      for (i = 0;i < totalcolors;i++)
            pens[i] = 255-i;
}

for (i = 0;i < totalcolors;i++)
{
      current_palette[pens[i]][0] = palette[3*i];
      current_palette[pens[i]][1] = palette[3*i+1];
      current_palette[pens[i]][2] = palette[3*i+2];
}

osd_update_display();
osd_update_display();
}
```

Esta función sirve para modificar un determinado color de la paleta de colores: **`osd_modify_pen()`**. Realmente no actualiza la paleta, sólo marca el color para que sea modificado.

```
void osd_modify_pen(int pen,unsigned char red, unsigned char green,
unsigned char blue)
{
if (scrbitmap->depth != 8)
{
      return;
}


if (current_palette[pen][0] != red ||
            current_palette[pen][1] != green ||
            current_palette[pen][2] != blue)
{
      current_palette[pen][0] = red;
      current_palette[pen][1] = green;
      current_palette[pen][2] = blue;

      dirtycolor[pen] = 1;
      dirtypalette = 1;
}
}
```

Esta función consigue un determinado valor de la paleta, como R,G,B: **osd_get_pen()**

```
void osd_get_pen(int pen,unsigned char *red, unsigned char *green,
unsigned char *blue)
{
*red = current_palette[pen][0];
*green = current_palette[pen][1];
*blue = current_palette[pen][2];
}
```

Esta función limpia los *buffers* de vídeo de la consola GP32: **gp32_clear_screen()**

```
void gp32_clear_screen(void)
{
int i;
for(i=0;i<2;i++)
      GpRectFill( NULL, &gpDraw[i],0,0,
                  gpDraw[i].buf_w,gpDraw[i].buf_h,0x0 );
}
```

La función más importante de actualización de pantalla del MAME es **osd_update_display().** Esta función va a ser llamada de forma periódica para:

- Sincronizar la velocidad del emulador: Si el emulador funciona demasiado rápido es necesario realizar una pequeña pausa para realizar una sincronización.
- Actualización de la paleta de colores: Si es necesario, realiza los cambios necesarios en la paleta de colores.
- Actualización de la pantalla: Llama a `gp32_update_display()` para actualizar la pantalla.

```
/* Update the display. */
void osd_update_display(void)
{
   int i;
   int curr;

   /* now wait until it's time to update the screen */
   if (throttle)
   {
         do
         {
               curr = GpTickCountGet();
               __asm("nop;nop;nop;nop;");
         } while ((curr - prev[memory]) < (frameskip+1) *
               gp32_timer/Machine->drv->frames_per_second);
   }
   else curr = GpTickCountGet();

   memory = (memory+1) % MEMORY;
   prev[memory] = curr;

   if (dirtypalette)
   {
         dirtypalette = 0;
```

```
        for (i = 0;i < 256;i++)
        {
              if (dirtycolor[i])
              {
                    int r,g,b;
                    dirtycolor[i] = 0;

                    r = current_palette[i][0];
                    g = current_palette[i][1];
                    b = current_palette[i][2];

                    {
                          GP_PALETTEENTRY colorgp;
                          colorgp = GP_RGB24(r,g,b);
                                      GpPaletteEntryChange(
                                      i,1,&colorgp,
                                      GPC_PAL_ASYNC_REALIZE);
                    }
              }
        }
   }

   /* copy the bitmap to screen memory */
   update_screen();
}
```

## 2.7.6. OPTIMIZACIONES

La optimización del código se puede hacer en múltiples frentes. Desde el punto de vista del procesador ARM9 se listan las más interesantes:

- El acceso a memoria por bytes consume el mismo tiempo de CPU que el acceso mediante enteros dword de 32 bit (4 bytes). Por tanto es interesante acceder a la memoria (lectura ó escritura), es decir, utilizar variables de tipo entero (int = 4 bytes). Todo lo que se pueda realizar en grupos de 4 bytes será mucho más optimizado que las típicas operaciones mediante bytes.
- Mientras que en arquitecturas x86 el acceso a datos no alineados en memoria (por ejemplo enteros no ubicados en direcciones de memoria múltiplo de 4 bytes) es bastante más lento, aunque no provocan reseteos del sistema, en los microprocesadores ARM9 el acceso a un dato no alineado de forma estricta en memoria provoca el reseteo del procesador, ya que la MMU (*Memory Management Unit*) de la consola GP32 lo considera un acceso a memoria ilegal.
- En la consola GP32 por defecto no está habilitada la caché de alineamiento de la CPU. Se puede acelerar el rendimiento de la aplicación si esta caché es activada. Para más detalles comprobar el módulo `src/gp32/enablecache.s`
- Cualquier operación con números reales es muy lenta en la consola GP32, ya que el micro ARM no tiene coprocesador matemático. Por tanto es recomendable utilizar otras alternativas en lugar de utilizar operaciones con reales. Una alternativa válida son las librerías de emulación de reales mediante números enteros con punto fijo.
- Las funciones de cadenas de caracteres y de memoria estándar provocan una merma del rendimiento y diversos problemas aleatorios porque no manejan bien la caché de la MMU. En su lugar es imprescindible utilizar las funciones correspondientes del GPSDK.
- Debido a la arquitectura de la consola GP32, si un programa consume permanentemente toda la CPU y satura el bus de memoria, puede ocurrir que el brillo de la consola empiece a fluctuar y que la actualización de pantalla se ralentice de modo que se vean rayas negras verticales. Para evitar este contratiempo hay varias alternativas:
    - Optimizar el código, por ejemplo sustituyendo operaciones con bytes por otras mediante enteros de 4 bytes (int), por lo que reduciremos bastante el trasiego de información con la memoria.
    - Otra alternativa es hacer justamente lo contrario: Meter operaciones en el código que no accedan a la memoria y que ralenticen el programa, de esta forma el bus de memoria no se congestiona.
    - También se puede reducir el grado de optimización del compilador GCC: Si con –O3 (optimización máxima) nos ocurre, podemos probar a bajar el grado de optimización a –O2 ó –O1. Pero también el programa irá más lento.
    - También se puede reducir este problema cambiando operaciones como `gm_memcpy()` ó `gm_memset()` por bucles que hagan la operación por bytes. Con ello relajaremos el uso del bus de memoria, ya que será más lento que las funciones de memoria del GPSDK.
    - Otra opción puede ser utilizar la opción de compilación –Os que reduce el tamaño del ejecutable y que indirectamente consigue resolverlo de vez en cuando.

- Se puede acelerar el acceso a los campos de las estructuras si estos campos estan alineados en memoria. Para ello se debe intentar que los campos sean de tipo entero ó múltiplo de 4 bytes. Si no es posible se deben de meter campos bytes no usados para asegurar el alineamiento estricto. Por último el compilador también tiene opciones para alinear las estructuras en memoria.

Desde el punto de vista de optimizaciones del código en C no específicas de los procesadores ARM, se pueden realizar las siguientes optimizaciones:

- Reducir el uso de bucles, sustituyéndolos por otros de menos iteraciones si es posible.
- Reducir los saltos entre distintas funciones todo lo posible.
- Utilizar la clave “*register*” para indicar al compilador qué variables son más usadas dentro de una función y que deberían estar en los registros del procesador para acelerar la ejecución del código. Por ejemplo: `register int contador;`
- Funciones ‘inline’: Se pueden definir funciones pequeñas que son llamadas desde muchos sitios para que se genere su código en cada una de las llamadas. Se incrementa el tamaño del ejecutable pero se mejora la velocidad. Ejemplo `static __inline funcion`
- Intentar evitar las operaciones de división, multiplicación y el módulo de la división, que son bastante lentas en la GP32. Además la multiplicación y división por números del tipo 2 elevado a ‘n’ se pueden cambiar por operaciones de desplazamiento de bits. Por ejemplo valor*4 es lo mismo valor<<2 ó por ejemplo valor/128 es lo mismo que valor>>7. En el módulo se puede sustituir por AND a nivel de bits si el divisor es también del tipo 2 elevado a ‘n’. Por ejemplo: valor % 128 es lo mismo que valor & 128.
- Se puede acelerar el acceso incremental a un array si se usan sentencias del tipo `*valor++`.

Desde el punto de vista del compilador, se dispone de varios parámetros que permiten la optimización:

- -On: Donde ‘n’ puede tener los valores 0, 1, 2, 3 ó ‘s’: Cuanto mayor sea el valor más optimizado generará el código, pero más probable es que aparezca el problema de saturación del bus de memoria, ya que se aumenta el tamaño del ejecutable. Además con ‘s’ podemos optimizar el código en tamaño del ejecutable en lugar de su velocidad.
- Podemos optimizar más el código de salida del compilador si usamos las opciones: `-mcpu=arm920 -mtune=arm920`, ya que genera código más óptimo para el procesador ARM de la consola GP32.
- `-fexpensive-optimizations`: Genera optimizaciones de velocidad a costa de aumentar aún más el tamaño del ejecutable.
- `-fstrict-aliasing`: Permite alinear los datos estrictamente en memoria, para acelerar su acceso.
- `-fomit-frame-pointer`: Elimina código de depuración del ejecutable.
- `-finline -finline-functions`: Activa el soporte para funciones ’inline’, para incrementar el tamaño del ejecutable pero mejorar su velocidad.

- `-mstructure-size-boundary=8` ó `-mstructure-size-boundary=32`: Permite alinear los campos de la estructura en direcciones de memoria múltiplos de 8 ó 32 bytes.

Desde el punto de vista del MAME se ha optimizado el funcionamiento fundamentalmente cambiando el emulador de Z80 que venía con el MAME 0.34 (*Z80Em* por *Marcel de Kogel*) por otro de otra versión superior del emulador (*Portable Z80 Emulator* por *Juergen Buchmueller*). El problema fundamental de antiguo emulador es que saturaba el bus de la GP32 debido a que tenía un exceso de saltos entre funciones y operaciones a nivel de bytes. El nuevo ‘core’ está más optimizado en este sentido.

## 2.7.7. FICHEROS “MAKEFILE”

En total hay 13 ficheros Makefile para compilar los 13 distintos ejecutables del MAME para GP32. La diferencia entre estos fichero Makefile son muy pequeñas, solamente las siguientes:

- Cada Makefile es compilado con una lista de drivers de juegos diferentes. Las distintas colecciones de drivers están en ficheros `Drivers*.c`, cada uno de los ficheros de drivers se corresponde con cada uno de los Makefile.
- Logicamente cada Makefile genera un ejecutable distinto.

Se muestra como ejemplo el contenido del fichero **Makefile_capcom**:

```
PREFIX := arm-elf-
CC     := $(PREFIX)gcc
LD     := $(PREFIX)gcc
AS     := $(PREFIX)as
AR     := $(PREFIX)ar
OBJCOPY := $(PREFIX)objcopy

GPLIBS := -LC:/DEVKITARM/arm-elf/lib/ \
-LC:/DEVKITARM/gamepark_sdk/lib/ -LC:/FRAN/GP32_MAME/GP32_MAME/OBJ/

INCLUDES := -IC:/DEVKITARM/arm-elf/include \
-IC:/DEVKITARM/gamepark_sdk/include

CFLAGS := -W -Wall -Wno-sign-compare -Wno-unused -Wpointer-arith \
-Wbad-function-cast -Wcast-align \
-Waggregate-return -Wshadow -Wstrict-prototypes \
-mcpu=arm920 -mtune=arm920 -fexpensive-optimizations -O \
-fstrict-aliasing -falign-functions=4 \
-fomit-frame-pointer -ffast-math -finline -finline-functions \
-fno-builtin -fno-common \
-fno-exceptions -ffreestanding -fshort-enums \
-mapcs-32 -mstructure-size-boundary=8 -mno-thumb-interwork \
-ansi -pedantic -fsigned-char \
-Isrc -Isrc/gp32 $(INCLUDES) -DLITTLE_ENDIAN -DGP32 -DLSB_FIRST \
-DSIGNED_SAMPLES -DACORN -DINLINE="static __inline" -DUSE_GP_MEM

VPATH=src src/Z80 src/M6502 src/I86 src/M6809

OBJS   =    obj/gp32/gpstart.o obj/gp32/gp32.o obj/gp32/vídeo.o \
            obj/gp32/gp32_menu.o \
            obj/gp32/sound.o obj/gp32/gpsoundbuf.o \
            obj/gp32/input.o obj/gp32/fileio.o obj/gp32/gp32_fileio.o
            obj/gp32/config.o \
            src/gp32/enablecache.o \
            obj/mame.o obj/common.o obj/usrintrf.o \
            obj/driver_capcom.o \
            obj/cpuintrf.o obj/memory.o obj/timer.o obj/palette.o \
            obj/inptport.o obj/unzip.o obj/inflate.o \
            obj/sndhrdw/adpcm.o \
            obj/sndhrdw/ay8910.o obj/sndhrdw/psgintf.o \
            obj/sndhrdw/2151intf.o obj/sndhrdw/fm.o \
            obj/sndhrdw/ym2151.o obj/sndhrdw/ym2413.o \
            obj/sndhrdw/2610intf.o \
            obj/sndhrdw/ym3812.o obj/sndhrdw/3812intf.o \
            obj/sndhrdw/tms5220.o obj/sndhrdw/5220intf.o \
            obj/sndhrdw/vlm5030.o \
```

```
            obj/sndhrdw/pokey.o obj/sndhrdw/sn76496.o \
            obj/sndhrdw/nes.o obj/sndhrdw/nesintf.o \
            obj/sndhrdw/astrocde.o \
            obj/sndhrdw/votrax.o obj/sndhrdw/dac.o \
            obj/sndhrdw/samples.o \
            obj/sndhrdw/streams.o \
            obj/machine/z80fmly.o obj/machine/6821pia.o \
            obj/vidhrdw/generic.o obj/sndhrdw/generic.o \
            obj/sndhrdw/namco.o \
            obj/machine/segacrpt.o \
            obj/machine/atarigen.o \
            obj/machine/slapstic.o \
            obj/machine/ticket.o \
            obj/Z80/Z80.o obj/M6502/m6502.o obj/I86/I86.o \
            obj/I8039/I8039.o obj/I8085/I8085.o \
            obj/M6809/m6809.o obj/M6805/m6805.o \
            obj/S2650/s2650.o obj/T11/t11.o \
            obj/m6808/m6808.o \
            obj/M68000/opcode0.o obj/M68000/opcode1.o \
            obj/M68000/opcode2.o obj/M68000/opcode3.o \
            obj/M68000/opcode4.o obj/M68000/opcode5.o \
            obj/M68000/opcode6.o obj/M68000/opcode7.o \
            obj/M68000/opcode8.o obj/M68000/opcode9.o \
            obj/M68000/opcodeb.o obj/M68000/opcodec.o \
            obj/M68000/opcoded.o obj/M68000/opcodee.o \
            obj/M68000/mc68kmem.o obj/M68000/cpufunc.o

LIBS =  obj/pacman.a obj/galaxian.a obj/scramble.a obj/cclimber.a \
       obj/phoenix.a obj/namco.a obj/univers.a obj/nintendo.a \
       obj/midw8080.a obj/midwz80.a obj/meadows.a obj/astrocde.a \
       obj/irem.a obj/oldtaito.a \
       obj/qixtaito.a obj/taito.a obj/taito2.a obj/williams.a \
       obj/capcom.a obj/gremlin.a obj/vicdual.a \
       obj/segar.a obj/zaxxon.a obj/system8.a \
       obj/btime.a obj/dataeast.a obj/dec8.a \
       obj/dec0.a obj/tehkan.a obj/konami.a obj/nemesis.a \
       obj/exidy.a \
       obj/kangaroo.a obj/missile.a obj/ataribw.a obj/atarisy1.a \
       obj/atari.a obj/rockola.a obj/technos.a \
       obj/berzerk.a obj/gameplan.a obj/stratvox.a obj/zaccaria.a \
       obj/upl.a obj/cinemar.a obj/thepit.a obj/valadon.a \
       obj/seibu.a obj/nichibut.a obj/other.a

all: mcapcom.gxb
      b2fxec -t "MAME GP32 1.1 (CAPCOM)" -b "mcapcom.bmp" \
      -a "Franxis" mcapcom.gxb mcapcom.fxe

mcapcom.gxb: mcapcom.elf
      $(OBJCOPY) -O binary mcapcom.elf mcapcom.gxb

mcapcom.elf: $(OBJS) $(LIBS)
      $(LD) -s -specs=gp32_gpsdk.specs $(OBJS) $(LIBS) \
      -o mcapcom.elf $(GPLIBS) -lgpgraphic -lgpmem -lgpos -lgpstdlib \
      -lgpstdio -lgpsound -lgpfont -lgpg_ex01 -lm

obj/%.o: src/%.c mame.h driver.h
      $(CC) $(CFLAGS) -o $@ -c $<

# dependencies
obj/M6502/m6502.o: m6502.c m6502.h m6502ops.h tbl6502.c \
```

```
      tbl65c02.c tbl6510.c
obj/I86/I86.o:  I86.c I86.h I86intrf.h ea.h host.h instr.h modrm.h
obj/M6809/m6809.o:  m6809.c m6809.h 6809ops.c
obj/M6808/M6808.o:  m6808.c m6808.h

obj/%.a:
       $(AR) -r $@ $^

obj/pacman.a: \
      obj/machine/pacman.o obj/drivers/pacman.o \
      obj/machine/pacplus.o \
      obj/machine/theglob.o \
      obj/drivers/maketrax.o \
      obj/machine/jrpacman.o obj/drivers/jrpacman.o \
      obj/vidhrdw/jrpacman.o \
      obj/vidhrdw/pengo.o obj/drivers/pengo.o

obj/galaxian.a: \
      obj/vidhrdw/galaxian.o obj/drivers/galaxian.o \
      obj/sndhrdw/mooncrst.o obj/drivers/mooncrst.o \

obj/scramble.a: \
      obj/machine/scramble.o obj/sndhrdw/scramble.o \
      obj/drivers/scramble.o \
      obj/vidhrdw/frogger.o obj/sndhrdw/frogger.o \
      obj/drivers/frogger.o \
      obj/drivers/ckongs.o \
      obj/drivers/scobra.o \
      obj/vidhrdw/amidar.o obj/drivers/amidar.o \
      obj/vidhrdw/jumpbug.o obj/drivers/jumpbug.o \
      obj/vidhrdw/fastfred.o obj/drivers/fastfred.o

obj/cclimber.a: \
      obj/vidhrdw/cclimber.o obj/sndhrdw/cclimber.o \
      obj/drivers/cclimber.o

obj/phoenix.a: \
      obj/vidhrdw/phoenix.o obj/sndhrdw/phoenix.o \
      obj/drivers/phoenix.o \
      obj/sndhrdw/pleiads.o \
      obj/vidhrdw/naughtyb.o obj/drivers/naughtyb.o

obj/namco.a: \
      obj/vidhrdw/rallyx.o obj/drivers/rallyx.o \
      obj/drivers/locomotn.o \
      obj/machine/bosco.o obj/sndhrdw/bosco.o obj/vidhrdw/bosco.o \
      obj/drivers/bosco.o \
      obj/machine/galaga.o obj/vidhrdw/galaga.o \
      obj/drivers/galaga.o \
      obj/machine/digdug.o obj/vidhrdw/digdug.o \
      obj/drivers/digdug.o \
      obj/vidhrdw/xevious.o obj/machine/xevious.o \
      obj/drivers/xevious.o \
      obj/machine/superpac.o obj/vidhrdw/superpac.o \
      obj/drivers/superpac.o \
      obj/machine/mappy.o obj/vidhrdw/mappy.o obj/drivers/mappy.o \
      obj/vidhrdw/pacland.o obj/drivers/pacland.o

obj/univers.a: \
      obj/vidhrdw/cosmica.o obj/drivers/cosmica.o \
```

```
	obj/vidhrdw/cheekyms.o obj/drivers/cheekyms.o \
	obj/machine/panic.o obj/vidhrdw/panic.o obj/drivers/panic.o \
	obj/vidhrdw/ladybug.o obj/drivers/ladybug.o \
	obj/vidhrdw/mrdo.o obj/drivers/mrdo.o \
	obj/machine/docastle.o obj/vidhrdw/docastle.o \
	obj/drivers/docastle.o \
	obj/drivers/dowild.o

obj/nintendo.a: \
	obj/vidhrdw/dkong.o obj/sndhrdw/dkong.o obj/drivers/dkong.o \
	obj/vidhrdw/mario.o obj/sndhrdw/mario.o obj/drivers/mario.o

obj/midw8080.a: \
	obj/machine/8080bw.o obj/vidhrdw/8080bw.o \
	obj/sndhrdw/8080bw.o obj/drivers/8080bw.o \
	obj/vidhrdw/m79amb.o obj/drivers/m79amb.o \

obj/midwz80.a: \
	obj/machine/z80bw.o obj/vidhrdw/z80bw.o obj/sndhrdw/z80bw.o \
	obj/drivers/z80bw.o

obj/meadows.a: \
	obj/drivers/lazercmd.o obj/vidhrdw/lazercmd.o \
	obj/drivers/meadows.o obj/sndhrdw/meadows.o \
	obj/vidhrdw/meadows.o \
	obj/drivers/medlanes.o obj/vidhrdw/medlanes.o \

obj/astrocde.a: \
	obj/machine/wow.o obj/vidhrdw/wow.o obj/sndhrdw/wow.o \
	obj/drivers/wow.o \
	obj/sndhrdw/gorf.o

obj/irem.a: \
	obj/sndhrdw/irem.o \
	obj/vidhrdw/mpatrol.o obj/drivers/mpatrol.o \
	obj/vidhrdw/yard.o obj/drivers/yard.o \
	obj/vidhrdw/kungfum.o obj/drivers/kungfum.o \
	obj/vidhrdw/travrusa.o obj/drivers/travrusa.o

obj/oldtaito.a: \
	obj/vidhrdw/crbaloon.o obj/drivers/crbaloon.o

obj/qixtaito.a: \
	obj/machine/qix.o obj/vidhrdw/qix.o obj/drivers/qix.o

obj/taito.a: \
	obj/machine/taito.o obj/vidhrdw/taito.o obj/drivers/taito.o

obj/taito2.a: \
	obj/vidhrdw/gsword.o obj/drivers/gsword.o \
	obj/vidhrdw/gladiatr.o obj/drivers/gladiatr.o \
	obj/vidhrdw/tokio.o obj/drivers/tokio.o \
	obj/machine/bublbobl.o obj/vidhrdw/bublbobl.o \
	obj/drivers/bublbobl.o \
	obj/vidhrdw/rastan.o obj/sndhrdw/rastan.o \
	obj/drivers/rastan.o \
	obj/machine/rainbow.o obj/drivers/rainbow.o \
	obj/machine/arkanoid.o obj/vidhrdw/arkanoid.o \
	obj/drivers/arkanoid.o \
	obj/vidhrdw/superqix.o obj/drivers/superqix.o \
```

```
	obj/vidhrdw/twincobr.o obj/drivers/twincobr.o \
	obj/machine/tnzs.o obj/vidhrdw/tnzs.o obj/drivers/tnzs.o \
	obj/drivers/arkanoi2.o \
	obj/machine/slapfght.o obj/vidhrdw/slapfght.o \
	obj/drivers/slapfght.o

obj/williams.a: \
	obj/machine/williams.o obj/vidhrdw/williams.o \
	obj/drivers/williams.o

obj/capcom.a: \
	obj/vidhrdw/vulgus.o obj/drivers/vulgus.o \
	obj/vidhrdw/sonson.o obj/drivers/sonson.o \
	obj/vidhrdw/higemaru.o obj/drivers/higemaru.o \
	obj/vidhrdw/1942.o obj/drivers/1942.o \
	obj/vidhrdw/exedexes.o obj/drivers/exedexes.o \
	obj/vidhrdw/commando.o obj/drivers/commando.o \
	obj/vidhrdw/gng.o obj/drivers/gng.o \
	obj/vidhrdw/gunsmoke.o obj/drivers/gunsmoke.o \
	obj/vidhrdw/srumbler.o obj/drivers/srumbler.o \
	obj/machine/lwings.o obj/vidhrdw/lwings.o \
	obj/drivers/lwings.o \
	obj/vidhrdw/sidearms.o obj/drivers/sidearms.o \
	obj/vidhrdw/bionicc.o obj/drivers/bionicc.o \
	obj/vidhrdw/1943.o obj/drivers/1943.o \
	obj/vidhrdw/blktiger.o obj/drivers/blktiger.o \
	obj/vidhrdw/tigeroad.o obj/drivers/tigeroad.o \
	obj/vidhrdw/lastduel.o obj/drivers/lastduel.o

obj/gremlin.a: \
	obj/vidhrdw/blockade.o obj/drivers/blockade.o

obj/vicdual.a: \
	obj/vidhrdw/vicdual.o obj/sndhrdw/vicdual.o \
	obj/drivers/vicdual.o

obj/segar.a: \
	obj/vidhrdw/segar.o obj/sndhrdw/segar.o obj/machine/segar.o \
	obj/drivers/segar.o \
	obj/sndhrdw/monsterb.o

obj/zaxxon.a: \
	obj/vidhrdw/zaxxon.o obj/sndhrdw/zaxxon.o \
	obj/drivers/zaxxon.o \
	obj/sndhrdw/congo.o obj/drivers/congo.o

obj/system8.a: \
	obj/vidhrdw/system8.o obj/drivers/system8.o

obj/btime.a: \
	obj/vidhrdw/btime.o obj/drivers/btime.o \
	obj/vidhrdw/tagteam.o obj/drivers/tagteam.o

obj/dataeast.a: \
	obj/vidhrdw/astrof.o obj/sndhrdw/astrof.o \
	obj/drivers/astrof.o \
	obj/vidhrdw/kchamp.o obj/drivers/kchamp.o \
	obj/vidhrdw/firetrap.o obj/drivers/firetrap.o \
	obj/vidhrdw/brkthru.o obj/drivers/brkthru.o \
	obj/vidhrdw/shootout.o obj/drivers/shootout.o \
```

```
	obj/vidhrdw/sidepckt.o obj/drivers/sidepckt.o \
	obj/vidhrdw/exprraid.o obj/drivers/exprraid.o

obj/dec8.a: \
	obj/vidhrdw/dec8.o obj/drivers/dec8.o

obj/dec0.a: \
	obj/vidhrdw/karnov.o obj/drivers/karnov.o \
	obj/machine/dec0.o obj/vidhrdw/dec0.o obj/drivers/dec0.o

obj/tehkan.a: \
	obj/vidhrdw/bombjack.o obj/drivers/bombjack.o \
	obj/sndhrdw/starforc.o obj/vidhrdw/starforc.o \
	obj/drivers/starforc.o \
	obj/vidhrdw/pbaction.o obj/drivers/pbaction.o \
	obj/vidhrdw/tehkanwc.o obj/drivers/tehkanwc.o \
	obj/vidhrdw/solomon.o obj/drivers/solomon.o \
	obj/vidhrdw/tecmo.o obj/drivers/tecmo.o \
	obj/vidhrdw/wc90.o obj/drivers/wc90.o \
	obj/vidhrdw/wc90b.o obj/drivers/wc90b.o

obj/konami.a: \
	obj/vidhrdw/pooyan.o obj/drivers/pooyan.o \
	obj/vidhrdw/timeplt.o obj/drivers/timeplt.o \
	obj/vidhrdw/rocnrope.o obj/drivers/rocnrope.o \
	obj/sndhrdw/gyruss.o obj/vidhrdw/gyruss.o \
	obj/drivers/gyruss.o \
	obj/machine/konami.o obj/vidhrdw/trackfld.o \
	obj/sndhrdw/trackfld.o obj/drivers/trackfld.o \
	obj/vidhrdw/circusc.o obj/drivers/circusc.o \
	obj/machine/tp84.o obj/vidhrdw/tp84.o obj/drivers/tp84.o \
	obj/vidhrdw/hyperspt.o obj/drivers/hyperspt.o \
	obj/vidhrdw/sbasketb.o obj/drivers/sbasketb.o \
	obj/vidhrdw/mikie.o obj/drivers/mikie.o \
	obj/vidhrdw/yiear.o obj/drivers/yiear.o \
	obj/vidhrdw/shaolins.o obj/drivers/shaolins.o \
	obj/vidhrdw/pingpong.o obj/drivers/pingpong.o \
	obj/vidhrdw/gberet.o obj/drivers/gberet.o \
	obj/vidhrdw/jailbrek.o obj/drivers/jailbrek.o \
	obj/vidhrdw/ironhors.o obj/drivers/ironhors.o \
	obj/machine/jackal.o obj/vidhrdw/jackal.o \
	obj/drivers/jackal.o \
	obj/vidhrdw/contra.o obj/drivers/contra.o \
	obj/vidhrdw/mainevt.o obj/drivers/mainevt.o

obj/nemesis.a: \
	obj/vidhrdw/nemesis.o obj/drivers/nemesis.o

obj/exidy.a: \
	obj/machine/exidy.o obj/vidhrdw/exidy.o \
	obj/sndhrdw/exidy.o obj/drivers/exidy.o \
	obj/sndhrdw/targ.o \
	obj/vidhrdw/circus.o obj/drivers/circus.o

obj/kangaroo.a: \
	obj/machine/kangaroo.o obj/vidhrdw/kangaroo.o \
	obj/drivers/kangaroo.o \
	obj/machine/arabian.o obj/vidhrdw/arabian.o \
	obj/drivers/arabian.o
```

```
obj/missile.a: \
      obj/machine/missile.o obj/vidhrdw/missile.o \
      obj/drivers/missile.o

obj/ataribw.a: \
      obj/machine/sprint2.o obj/vidhrdw/sprint2.o \
      obj/drivers/sprint2.o \
      obj/machine/sbrkout.o obj/vidhrdw/sbrkout.o \
      obj/drivers/sbrkout.o \
      obj/machine/dominos.o obj/vidhrdw/dominos.o \
      obj/drivers/dominos.o \
      obj/vidhrdw/nitedrvr.o obj/machine/nitedrvr.o \
      obj/drivers/nitedrvr.o \
      obj/vidhrdw/bsktball.o obj/machine/bsktball.o \
      obj/drivers/bsktball.o \
      obj/vidhrdw/copsnrob.o obj/machine/copsnrob.o \
      obj/drivers/copsnrob.o \
      obj/machine/avalnche.o obj/vidhrdw/avalnche.o \
      obj/drivers/avalnche.o \
      obj/machine/subs.o obj/vidhrdw/subs.o obj/drivers/subs.o

obj/atarisy1.a: \
      obj/machine/atarisy1.o obj/vidhrdw/atarisy1.o \
      obj/drivers/atarisy1.o

obj/atari.a: \
      obj/machine/gauntlet.o obj/vidhrdw/gauntlet.o \
      obj/drivers/gauntlet.o \
      obj/vidhrdw/atetris.o obj/drivers/atetris.o \
      obj/vidhrdw/vindictr.o obj/drivers/vindictr.o \
      obj/vidhrdw/klax.o obj/drivers/klax.o \
      obj/vidhrdw/eprom.o obj/drivers/eprom.o \
      obj/vidhrdw/xybots.o obj/drivers/xybots.o

obj/rockola.a: \
      obj/vidhrdw/rockola.o obj/sndhrdw/rockola.o \
      obj/drivers/rockola.o \
      obj/vidhrdw/warpwarp.o obj/drivers/warpwarp.o

obj/technos.a: \
      obj/vidhrdw/mystston.o obj/drivers/mystston.o \
      obj/vidhrdw/matmania.o obj/drivers/matmania.o \
      obj/vidhrdw/renegade.o obj/drivers/renegade.o \
      obj/vidhrdw/xain.o obj/drivers/xain.o \
      obj/vidhrdw/ddragon.o obj/drivers/ddragon.o \
      obj/vidhrdw/blockout.o obj/drivers/blockout.o

obj/berzerk.a: \
      obj/machine/berzerk.o obj/vidhrdw/berzerk.o \
      obj/sndhrdw/berzerk.o obj/drivers/berzerk.o

obj/gameplan.a: \
      obj/vidhrdw/gameplan.o obj/drivers/gameplan.o

obj/stratvox.a: \
      obj/vidhrdw/route16.o obj/drivers/route16.o

obj/zaccaria.a: \
      obj/vidhrdw/zaccaria.o obj/drivers/zaccaria.o
```

```
obj/upl.a: \
      obj/vidhrdw/nova2001.o obj/drivers/nova2001.o \
      obj/vidhrdw/pkunwar.o obj/drivers/pkunwar.o \
      obj/vidhrdw/ninjakd2.o obj/drivers/ninjakd2.o

obj/cinemar.a: \
      obj/vidhrdw/jack.o obj/drivers/jack.o

obj/thepit.a: \
      obj/vidhrdw/thepit.o obj/drivers/thepit.o

obj/valadon.a: \
      obj/machine/bagman.o obj/vidhrdw/bagman.o obj/drivers/bagman.o

obj/seibu.a: \
      obj/vidhrdw/wiz.o obj/drivers/wiz.o

obj/nichibut.a: \
      obj/vidhrdw/cop01.o obj/drivers/cop01.o \
      obj/vidhrdw/terracre.o obj/drivers/terracre.o \
      obj/vidhrdw/galivan.o obj/drivers/galivan.o

obj/other.a: \
      obj/machine/spacefb.o obj/vidhrdw/spacefb.o \
      obj/sndhrdw/spacefb.o obj/drivers/spacefb.o \
      obj/vidhrdw/tutankhm.o obj/drivers/tutankhm.o \
      obj/drivers/junofrst.o \
      obj/vidhrdw/ccastles.o obj/drivers/ccastles.o \
      obj/vidhrdw/blueprnt.o obj/drivers/blueprnt.o \
      obj/vidhrdw/bankp.o obj/drivers/bankp.o \
      obj/machine/espial.o obj/vidhrdw/espial.o obj/drivers/espial.o \
      obj/machine/cloak.o obj/vidhrdw/cloak.o obj/drivers/cloak.o \
      obj/vidhrdw/champbas.o obj/drivers/champbas.o \
      obj/drivers/sinbadm.o \
      obj/vidhrdw/exerion.o obj/drivers/exerion.o \
      obj/machine/foodf.o obj/vidhrdw/foodf.o obj/drivers/foodf.o \
      obj/vidhrdw/jack.o obj/drivers/jack.o \
      obj/machine/vastar.o obj/vidhrdw/vastar.o \
      obj/drivers/vastar.o \
      obj/vidhrdw/citycon.o obj/drivers/citycon.o \
      obj/vidhrdw/psychic5.o obj/drivers/psychic5.o \
      obj/machine/jedi.o obj/vidhrdw/jedi.o obj/sndhrdw/jedi.o \
      obj/drivers/jedi.o \
      obj/vidhrdw/tankbatt.o obj/drivers/tankbatt.o \
      obj/vidhrdw/dday.o obj/sndhrdw/dday.o obj/drivers/dday.o \
      obj/vidhrdw/toki.o obj/drivers/toki.o \
      obj/vidhrdw/snowbros.o obj/drivers/snowbros.o \
      obj/vidhrdw/gundealr.o obj/drivers/gundealr.o \
      obj/machine/leprechn.o obj/vidhrdw/leprechn.o \
      obj/drivers/leprechn.o \
      obj/vidhrdw/hexa.o obj/drivers/hexa.o \
      obj/vidhrdw/redalert.o obj/sndhrdw/redalert.o \
      obj/drivers/redalert.o \
      obj/machine/spiders.o obj/vidhrdw/crtc6845.o \
      obj/vidhrdw/spiders.o obj/drivers/spiders.o \
      obj/machine/stactics.o obj/vidhrdw/stactics.o \
      obj/drivers/stactics.o \
      obj/vidhrdw/goldstar.o obj/drivers/goldstar.o \
      obj/vidhrdw/vigilant.o obj/drivers/vigilant.o \
      obj/vidhrdw/sharkatt.o obj/drivers/sharkatt.o \
```

```
obj/vidhrdw/kingobox.o obj/drivers/kingobox.o \
obj/machine/exctsccr.o obj/vidhrdw/exctsccr.o \
obj/drivers/exctsccr.o \
obj/vidhrdw/speedbal.o obj/drivers/speedbal.o \
obj/vidhrdw/sauro.o obj/drivers/sauro.o \
obj/vidhrdw/pow.o obj/drivers/pow.o
```

# 3. PLANOS

Por la naturaleza del proyecto, los planos que en este caso serían los programas, se han incluido y explicado detalladamente en la memoria.

# 4. PLIEGO DE CONDICIONES

En este capítulo se describen los componentes necesarios para la realización del proyecto, así como los requisitos mínimos necesarios para hacer funcionar la aplicación:

## 4.1. COMPONENTES NECESARIOS

A continuación se muestran los componentes hardware y software utilizados para la elaboración de este trabajo fin de carrera.

Componentes hardware:

- Ordenador personal.
- Conexión a Internet.
- Consola portátil GP32 modelo BLU + tarjeta de memoria SmartMediaCard (SMC) de 128 Mb.

Componentes software:

- Sistema operativo Windows XP Professional SP2 en español.
- Microsoft Office XP Professional.
- Entorno de desarrollo DevKitARM r11 (GCC 3.4.3).
- Librerias GPSDK 2.1.0.
- B2FXEC 0.6a.
- Geepee32 0.40.
- GP32Converter 1.3.
- MakeSMC128 1.0.

## 4.2. REQUISITOS DE UTILIZACIÓN DEL PROYECTO

Los programas ejecutables generados en el proyecto están destinados para funcionar en el siguiente entorno hardware y software:

Requisitos hardware:

- Consola GP32 coreana ó europea de cualquiera de los distintos modelos existentes: GP32 NLU, GP32 FLU, GP32 BLU ó GP32 BLU+.
- Velocidad de reloj de la consola de al menos 133 MHz. En teoría todas las consolas GP32 llegan al menos a esta velocidad de reloj.
- Tarjeta de memoria SmartMediaCard (SMC) de al menos 16 Mb (hasta un máximo de 128 Mb) con un mínimo de 7 Mb libres + el espacio necesario para almacenar las ROMs de los juegos.

Requisitos software:

- Firmware de la GP32 con posibilidad de ejecutar ficheros con extensión FXE (Free Executables). Las consolas GP32 más antiguas no tienen esta característica, por que sería necesario actualizar su firmware ó bien utilizar la aplicación gratuita Free Launcher para lanzar los programas desde la consola.
- Las ROMs de los juegos: Son los volcados sobre ficheros de las ROMs de las antiguas recreativas.

Condiciones de utilización de la aplicación:

- El usuario debe utilizar las ROMs de los juegos respetando la legalidad y el copyright de los creadores de las máquinas recreativas originales.

# 5. PRESUPUESTO

## 5.1. PARTES DEL PRESUPUESTO

Se compone de los siguientes apartados:

- Coste de uso de equipos.
- Coste personal.
- Coste de materiales.
- Gastos generales, beneficio industrial e IVA.

La duración estimada del proyecto es de 8 meses, incluyendo el mecanografiado y el tratamiento de textos.

La elaboración de este proyecto ha sido realizada por un ingeniero técnico de telecomunicación.

Todas las cantidades seran expresadas en euros.

## 5.2. PRESUPUESTO DE EJECUCIÓN MATERIAL (PEM)

El presupuesto de ejecución material se calcula a partir del coste de los equipos utilizados en el laboratorio, software y material de oficina, junto con el coste personal.

### 5.2.1. COSTE DE USO DE EQUIPOS

Aquí se incluyen los gastos de amortización de los equipos informáticos y equipos del laboratorio empleados, además del material software y el coste por material de oficina empleado.

- **Coste de equipos informáticos:**

| Concepto | Precio | Amortización | Uso | Total |
|---|---|---|---|---|
| Ordenador personal | 1099.00 | 4 años | 8 meses | 183.17 |
| Impresora HP LaserJet Color | 599.00 | 4 años | 8 meses | 199.66 |
| Modem V92 | 89.00 | 4 años | 8 meses | 29.66 |
| Línea telef´ónica | 300.00 | 4 años | 8 meses | 100.00 |
| | | | **TOTAL:** | **512.49** |

- **Coste del entorno hardware de la consola:**

| Concepto<br>Precio | Precio por unidad | Unidades | Total |
|---|---|---|---|
| Consola GP32 modelo BLU | 115.00 | 1 | 115.00 |
| Tarjeta SMC 128 Mb | 30.00 | 1 | 30.00 |
| | | **TOTAL:** | **145.00** |

- **Coste por material software:**

| Concepto | Precio por unidad | Unidades | Total |
|---|---|---|---|
| Microsoft Windows XP Professional | 133.62 | 1 | 133.62 |
| Microsoft Office XP Professional | 318,97 | 1 | 318.97 |
| | | **TOTAL:** | **452.59** |

- **Coste por material de oficina:**

| Concepto | Precio por unidad | Unidades | Total |
|---|---|---|---|
| CDs regrabables | 0.90 | 10 | 9.00 |
| Cartucho de tinta | 24.00 | 3 | 72.00 |
| Papel | 14.00 | 1 | 14.00 |
| Encuadernado | 18.00 | 4 | 72.00 |
| | | **TOTAL:** | **153.00** |

Total de coste por equipos..... 1263.08 euros

## 5.2.2. COSTE POR TIEMPO DE EMPLEADO

| Función | Nº de horas | Euros / Hora | Total |
|---|---|---|---|
| Ingeniero Técnico de Telecomunicación | 680.00 | 60.00 | 40800.00 |
| Mecanógrafo | 176.00 | 50.00 | 8800.00 |
| | | **TOTAL:** | **49600.00** |

### 5.2.3 COSTE TOTAL DEL PRESUPUESTO DE EJECUCIÓN MATERIAL (PEM)

Total de coste por equipos..... 1263.08 euros
Total coste por tiempo de empleado..... 49600.00 euros
Total de coste de ejecución material (PEM)..... 50863.08 euros

## 5.3. PRESUPUESTO DE EJECUCIÓN POR CONTRATA (PC)

El presupuesto de ejecución por contrata (PC) incluye el coste de ejecución material junto con los gastos generales, así como el beneficio industrial y los horarios de dirección y redacción.

Los gastos generales son los obligados por disponer de las instalaciones donde llevar a cabo los trabajos, así como las amortizaciones, etc.

| Concepto | Valor | Total |
|---|---|---|
| Presupuesto de ejecución material (PEM) | 100% | 50863.08 |
| Gastos generales y beneficio industrial | 15% PEM | 7629.46 |
| Horarios de dirección | 7% PEM | 3560.42 |
| Horarios de secretaría | 7% PEM | 3560.42 |
| | **TOTAL:** | **65613.38** |

## 5.4. PRESUPUESTO TOTAL

| Concepto | Valor | Total |
|---|---|---|
| Presupuesto de ejecución por contrata (PC) | 100% | 65613.38 |
| IVA | 16% PC | 10498.14 |
| | **TOTAL:** | **76111.52** |

El presupuesto total del proyecto asciende a la cantidad de **setenta y seis mil ciento once euros** con **cincuenta y dos céntimos** de euro.

En Alcalá de Henares, a 23 de mayo de 2005.

Fdo: Francisco Javier Martínez Romo.
Ingeniero Técnico de Telecomunicación.

# 6. MANUAL DE USUARIO

## 6.1. INTRODUCCIÓN

Se trata de un *port* para la consola portátil GP32 del emulador MAME 0.34 de *Nicola Salmoria*. Emula todas las recreativas soportadas por el MAME original, a excepción de:

- SNK Neo·Geo
- Sega System 16
- CPS-1
- Juegos vectoriales
- Otros juegos que no caben en la memoria de la GP32
- Otros juegos diversos que no funcionan todavía por distintas razones

Esta versión emula 706 romsets distintos.

Página web oficial del proyecto: http://www.talfi.net/gp32_franxis/


**Figura 34: Logotipo de MAME GP32**

El emulador viene en un fichero comprimido .RAR con los siguientes archivos:

- `GPMM\MAME034\` -> Directorio donde colocar las ROMs
- `GPMM\*.fxe` -> Cada uno de los 13 distintos ejecutables
- `clrmame.dat` -> Fichero de datos para ClrMAME Pro
- `gamelist.txt` -> Listado de juegos soportados
- `leeme.txt` -> Manual de usuario en idioma español
- `readme.txt` -> Manual de usuario en idioma inglés
- `whatsnew.txt` -> Historial de versiones

## 6.2. CONTROLES

Los controles del emulador son:

- **Joystick**: Movimiento de pad, ratón y control analógico de joysticks 1 y 2.
- **Botones A,B,Select,R,L,Start**: Botones A,B,C,D,E,F.
- **Botón L**: Insertar créditos.
- **Botón R**: Cambio de orientación de pantalla.
- **Botón R prolongadamente durante un segundo**: Pausa.
- **Botón START**: Jugar.
- **Botones L+R simultaneamente**: Salir del juego para elegir otro de los disponibles.
- **Botón START+SELECT simultaneamente**: Resetear el juego.

## 6.3. OPCIONES DEL EMULADOR

Primero es necesario seleccionar un juego de la lista:


**Figura 35: Imagen del menú de selección de juegos**

Después de seleccionar el juego aparecen las siguientes opciones:

- **Pulsa L para cambiar la frecuencia de la GP32**: 133 a 166 MHz funcionan bien en la mayoría de las GP32 modelo BLU. En las de modelo FLU se pueden conseguir frecuencias más altas (168 a 256 MHz). **AVISO**: Es peligroso usar frecuencias mayores de 166 MHz, usa bajo tu responsabilidad.

- **Pulsa R para cambiar el *frame-skip***: Se puede seleccionar de 0 a 5 de frame-skip. A menor frame-skip más frames de vídeo por segundo y el juego se ve mejor, pero el emulador se relentiza. Un valor entre 1 y 2 dependiendo del juego es el adecuado en principio.

- **Pulsa START o SELECT para cambiar las opciones de sonido**, que son las siguientes:
  - **Sound ON**: El sonido del MAME activado.
  - **Sound OFF**: El sonido esta internamente emulado por el MAME, pero no se reproduce en la GP32. El emulador funciona más rápido que con sonido.

- **Not Emulated**: El sonido no está emulado internamente y está desactivado. El emulador así es lo más rápido posible por ahora, pero algunos juegos no funcionan en este modo (por ejemplo los juegos de Capcom).

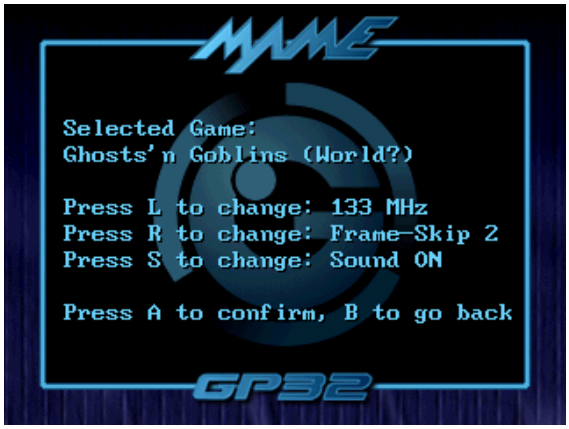



**Figura 36: Imagen de la pantalla de opciones**

- **Pulsa A para jugar al juego seleccionado o B para volver al menú inicial de selección de juegos.**

- En el frontend si se **pulsa START+SELECT simultaneamente, la GP32 se resetea**.

## 6.4. INSTALACIÓN

Hay disponibles 13 ejecutables distintos, a copiar en \GPMM de la SMC:

- **mclassic.fxe**: Más de 500 clásicos: Pacman, Frogger, Donkey Kong, etc...
- **matari.fxe**: Juegos de Atari: Tetris, Klax, Gauntlet, etc...
- **mcapcom.fxe**: Capcom: Ghosts'n Goblins, Commando, etc...
- **mdataeas.fxe**: Data East: Shootout, Oscar, Karnov, etc...
- **mirem.fxe**: Irem: Vigilante, Kung Fu Master, etc...
- **mkonami.fxe**: Konami: Contra, Yie Ar Kungfu, Track & Field, etc...
- **mnamco.fxe**: Namco: Motos, Xevious, etc...
- **msega.fxe**: Sega: Zaxxon, Wonder Boy, Pitfall 2, etc...
- **mtaito.fxe**: Taito: Bubble Bobble, Arkanoid, Tokio, etc...
- **mtechnos.fxe**: Technos Japan: Double Dragon, Renegade, etc...
- **mtecmo.fxe**: Tehkan/Tecmo: Bombjack, Rygar, etc...
- **mwilliam.fxe**: Williams: Blaster, Defender, Joust, Robotron, etc...
- **mothers.fxe**: Otros juegos diversos: Snow Bros, Terra Cresta, Ninja Kid 2, etc...

Las ROMs de las recreativas hay que copiarlas en el directorio `\GPMM\MAME034`, bien en ficheros ZIP o bien en subcarpetas con las ROMs descomprimidas. Para más información ver más adelante...

Al lanzar uno de los juegos desde el emulador, aparecen diversos mensajes durante la carga e inicialización del emulador:


**Figura 37: Imagen del progreso de inicialización**

Si ocurre algún error durante la inicialización (por ejemplo por algún problema al cargar alguna de las ROMs requeridas por el juego seleccionado), aparecerá un mensaje de error y después se volverá al menú de selección de juegos. En caso contrario, a los pocos segundos arrancará el juego:


**Figura 38: Ghosts'n Goblins**

## 6.5. JUEGOS SOPORTADOS

En total son **706 juegos soportados.**

Los juegos deben de ir dentro de la carpeta `\GPMM\MAME034` de la SMC.

La lista de juegos soportados se muestra en la siguiente tabla:

| ***Identificador*** | ***Fichero ZIP*** | ***Nombre del juego*** |
|---|---|---|
| 005 | 005.zip | "005" |
| 1942 | 1942.zip | "1942 (set 1)" |
| 1942a | 1942a.zip | "1942 (set 2)" |
| 1942b | 1942b.zip | "1942 (set 3)" |
| 1943 | 1943.zip | "1943 (US)" |
| 1943jap | 1943jap.zip | "1943 (Japan)" |
| 1943kai | 1943kai.zip | "1943 Kai" |
| 280zzzap | 280zzzap.zip | "280-ZZZAP" |
| 3stooges | 3stooges.zip | "Three Stooges" |
| 4dwarrio | 4dwarrio.zip | "4D Warriors" |
| 600 | 600.zip | "600" |
| alpine | alpine.zip | "Alpine Ski (set 1)" |
| alpinea | alpinea.zip | "Alpine Ski (set 2)" |
| amidar | amidar.zip | "Amidar (US)" |
| amidarjp | amidarjp.zip | "Amidar (Japan)" |
| amidars | amidars.zip | "Amidar (Scramble hardware)" |
| amigo | amigo.zip | "Amigo" |
| anteater | anteater.zip | "Ant Eater" |
| arabian | arabian.zip | "Arabian" |
| ark2us | ark2us.zip | "Arkanoid 2 - Revenge of DOH (Romstar)" |
| arkanoi2 | arkanoi2.zip | "Arkanoid 2 - Revenge of DOH" |
| arkanoid | arkanoid.zip | "Arkanoid (World)" |
| arkatayt | arkatayt.zip | "Arkanoid (Tayto bootleg, Japanese)" |
| arknoidu | arknoidu.zip | "Arkanoid (USA)" |
| armorca2 | armorca2.zip | "Armored Car (set 2)" |
| armorcar | armorcar.zip | "Armored Car (set 1)" |
| astdelux | astdelux.zip | "Asteroids Deluxe (rev 2)" |
| asteroi1 | asteroi1.zip | "Asteroids (rev 1)" |
| asteroid | asteroid.zip | "Asteroids (rev 2)" |
| astinvad | astinvad.zip | "Astro Invader" |
| astlaser | astlaser.zip | "Astro Laser" |
| astrob | astrob.zip | "Astro Blaster (version 2)" |
| astrob1 | astrob1.zip | "Astro Blaster (version 1)" |
| astrof | astrof.zip | "Astro Fighter" |
| astrof2 | astrof2.zip | "Astro Fighter (set 2)" |
| astrof3 | astrof3.zip | "Astro Fighter (set 3)" |
| atetckt2 | atetckt2.zip | "Tetris (Cocktail set 2)" |
| atetcktl | atetcktl.zip | "Tetris (Cocktail set 1)" |
| atetris | atetris.zip | "Tetris (set 1)" |
| atetrisa | atetrisa.zip | "Tetris (set 2)" |
| atetrisb | atetrisb.zip | "Tetris (bootleg)" |

| ***Identificador*** | ***Fichero ZIP*** | ***Nombre del juego*** |
|---|---|---|
| atlantis | atlantis.zip | "Battle of Atlantis" |
| avalnche | avalnche.zip | "Avalanche" |
| bagman | bagman.zip | "Bagman" |
| bagmans | bagmans.zip | "Bagman (Stern set 1)" |
| bagmans2 | bagmans2.zip | "Bagman (Stern set 2)" |
| bandido | bandido.zip | "Bandido" |
| bankp | bankp.zip | "Bank Panic" |
| beastf | beastf.zip | "Beastie Feastie (Pac Man hardware)" |
| berzerk | berzerk.zip | "Berzerk" |
| berzerk1 | berzerk1.zip | "Berzerk (version 1)" |
| billiard | billiard.zip | "The Billiards" |
| bioatack | bioatack.zip | "Bio Attack" |
| bionicc | bionicc.zip | "Bionic Commando (set 1)" |
| bionicc2 | bionicc2.zip | "Bionic Commando (set 2)" |
| blaster | blaster.zip | "Blaster" |
| blasto | blasto.zip | "Blasto" |
| blockade | blockade.zip | "Blockade" |
| blockout | blockout.zip | "Block Out" |
| blueprnt | blueprnt.zip | "Blue Print" |
| blueshrk | blueshrk.zip | "Blue Shark" |
| bnj | bnj.zip | "Bump 'n' Jump" |
| boblbobl | boblbobl.zip | "Bobble Bobble" |
| bombjack | bombjack.zip | "Bomb Jack" |
| boothill | boothill.zip | "Boot Hill" |
| bosco | bosco.zip | "Bosconian (Midway)" |
| bosconm | bosconm.zip | "Bosconian (Namco)" |
| bowler | bowler.zip | "4 Player Bowling" |
| brain | brain.zip | "Brain" |
| brix | brix.zip | "Brix" |
| brkthru | brkthru.zip | "Break Thru" |
| brubber | brubber.zip | "Burnin' Rubber" |
| bsktball | bsktball.zip | "Basketball" |
| btime | btime.zip | "Burger Time (Midway)" |
| btimea | btimea.zip | "Burger Time (Data East)" |
| bubbles | bubbles.zip | "Bubbles" |
| bubblesr | bubblesr.zip | "Bubbles (Solid Red label)" |
| bublbobl | bublbobl.zip | "Bubble Bobble" |
| bullfgtj | bullfgtj.zip | "Bull Fight (Japan)" |
| bwidow | bwidow.zip | "Black Widow" |
| bzone | bzone.zip | "Battle Zone" |
| bzone2 | bzone2.zip | "Battle Zone (alternate version)" |
| calipso | calipso.zip | "Calipso" |
| caractn | caractn.zip | "Car Action" |
| carnival | carnival.zip | "Carnival" |
| cavenger | cavenger.zip | "Cosmic Avenger" |
| ccboot | ccboot.zip | "Crazy Climber (bootleg)" |
| ccjap | ccjap.zip | "Crazy Climber (Japan)" |
| cclimber | cclimber.zip | "Crazy Climber (US)" |
| challeng | challeng.zip | "Challenger" |
| champbas | champbas.zip | "Champion Baseball" |

| ***Identificador*** | ***Fichero ZIP*** | ***Nombre del juego*** |
|---|---|---|
| checkman | checkman.zip | "Checkman" |
| checkmat | checkmat.zip | "Checkmate" |
| cheekyms | cheekyms.zip | "Cheeky Mouse" |
| chelnov | chelnov.zip | "Chelnov - Atomic Runner (US)" |
| chelnovj | chelnovj.zip | "Chelnov - Atomic Runner (Japan)" |
| chplft | chplft.zip | "Choplifter" |
| chplftb | chplftb.zip | "Choplifter (alternate)" |
| chplftbl | chplftbl.zip | "Choplifter (bootleg)" |
| circus | circus.zip | "Circus" |
| circusc | circusc.zip | "Circus Charlie" |
| circusc2 | circusc2.zip | "Circus Charlie (level select)" |
| citycon | citycon.zip | "City Connection" |
| ckong | ckong.zip | "Crazy Kong (set 1)" |
| ckonga | ckonga.zip | "Crazy Kong (set 2)" |
| ckongalc | ckongalc.zip | "Crazy Kong (Alca bootleg)" |
| ckongjeu | ckongjeu.zip | "Crazy Kong (Jeutel bootleg)" |
| ckongs | ckongs.zip | "Crazy Kong (Scramble Hardware)" |
| cloak | cloak.zip | "Cloak & Dagger" |
| clowns | clowns.zip | "Clowns" |
| cobracom | cobracom.zip | "Cobra Command" |
| colony7 | colony7.zip | "Colony 7 (set 1)" |
| colony7a | colony7a.zip | "Colony 7 (set 2)" |
| commandj | commandj.zip | "Senjo no Ookami" |
| commando | commando.zip | "Commando (World)" |
| commandu | commandu.zip | "Commando (US)" |
| commsega | commsega.zip | "Commando (Sega)" |
| comotion | comotion.zip | "Comotion" |
| congo | congo.zip | "Congo Bongo" |
| contra | contra.zip | "Contra (US)" |
| contrab | contrab.zip | "Contra (US bootleg)" |
| cookrace | cookrace.zip | "Cook Race" |
| cop01 | cop01.zip | "Cop 01 (set 1)" |
| cop01a | cop01a.zip | "Cop 01 (set 2)" |
| copsnrob | copsnrob.zip | "Cops'n Robbers" |
| cosmica | cosmica.zip | "Cosmic Alien" |
| cosmicmo | cosmicmo.zip | "Cosmic Monsters" |
| crash | crash.zip | "Crash" |
| crbalon2 | crbalon2.zip | "Crazy Balloon (set 2)" |
| crbaloon | crbaloon.zip | "Crazy Balloon (set 1)" |
| crush | crush.zip | "Crush Roller" |
| csprint | csprint.zip | "Championship Sprint" |
| curvebal | curvebal.zip | "Curve Ball" |
| darwin | darwin.zip | "Darwin 4078 (Japan)" |
| dday | dday.zip | "D-Day" |
| ddragon | ddragon.zip | "Double Dragon" |
| ddragon2 | ddragon2.zip | "Double Dragon 2" |
| ddragonb | ddragonb.zip | "Double Dragon (bootleg)" |
| deadeye | deadeye.zip | "Dead Eye" |
| defcmnd | defcmnd.zip | "Defense Command" |
| defence | defence.zip | "Defence Command" |

| *Identificador* | *Fichero ZIP* | *Nombre del juego* |
|---|---|---|
| defender | defender.zip | "Defender" |
| depthch | depthch.zip | "Depth Charge" |
| desterth | desterth.zip | "Destination Earth" |
| diamond | diamond.zip | "Diamond Run" |
| digdug | digdug.zip | "Dig Dug (Atari)" |
| digdug2 | digdug2.zip | "Dig Dug 2" |
| digdugnm | digdugnm.zip | "Dig Dug (Namco)" |
| dkjrbl | dkjrbl.zip | "Donkey Kong Junior (bootleg?)" |
| dkjrjp | dkjrjp.zip | "Donkey Kong Junior (Japan)" |
| dkngjrjp | dkngjrjp.zip | "Donkey Kong Jr. (Original Japanese)" |
| dkong | dkong.zip | "Donkey Kong (US)" |
| dkong3 | dkong3.zip | "Donkey Kong 3" |
| dkongjp | dkongjp.zip | "Donkey Kong (Japan)" |
| dkongjr | dkongjr.zip | "Donkey Kong Junior (US)" |
| docastl2 | docastl2.zip | "Mr. Do's Castle (set 2)" |
| docastle | docastle.zip | "Mr. Do's Castle (set 1)" |
| dogpatch | dogpatch.zip | "Dog Patch" |
| dominos | dominos.zip | "Dominos" |
| dorunrun | dorunrun.zip | "Mr. Do! Run Run" |
| douni | douni.zip | "Mr. Do vs. Unicorns" |
| dowild | dowild.zip | "Mr. Do's Wild Ride" |
| dplay | dplay.zip | "Double Play" |
| eagle | eagle.zip | "Eagle" |
| earthinv | earthinv.zip | "Super Earth Invasion" |
| eggs | eggs.zip | "Eggs" |
| einnings | einnings.zip | "Extra Innings" |
| elevatob | elevatob.zip | "Elevator Action (bootleg)" |
| elevator | elevator.zip | "Elevator Action" |
| elim2 | elim2.zip | "Eliminator (2 Players)" |
| elim4 | elim4.zip | "Eliminator (4 Players)" |
| espial | espial.zip | "Espial (US?)" |
| espiale | espiale.zip | "Espial (Europe)" |
| excthour | excthour.zip | "Exciting Hour" |
| exctsccb | exctsccb.zip | "Exciting Soccer (bootleg)" |
| exctsccr | exctsccr.zip | "Exciting Soccer" |
| exedexes | exedexes.zip | "Exed Exes" |
| exerion | exerion.zip | "Exerion" |
| exerionb | exerionb.zip | "Exerion (bootleg)" |
| exeriont | exeriont.zip | "Exerion (Taito)" |
| exprraid | exprraid.zip | "Express Raider" |
| eyes | eyes.zip | "Eyes" |
| fantasy | fantasy.zip | "Fantasy" |
| fantazia | fantazia.zip | "Fantazia" |
| fantzone | fastfred.zip | "Fantasy Zone" |
| fastfred | fax.zip | "Fast Freddie" |
| fax | firetpbl.zip | "Fax" |
| firetpbl | firetrap.zip | "Fire Trap (Japan bootleg)" |
| flicky | flicky.zip | "Flicky (set 1)" |
| flicky2 | flicky2.zip | "Flicky (set 2)" |
| flyboy | flyboy.zip | "Fly-Boy (bootleg?)" |

| ***Identificador*** | ***Fichero ZIP*** | ***Nombre del juego*** |
|---|---|---|
| foodf | foodf.zip | "Food Fight" |
| frenzy | frenzy.zip | "Frenzy" |
| frenzy1 | frenzy1.zip | "Frenzy (version 1)" |
| frogger | frogger.zip | "Frogger (set 1)" |
| frogger2 | frogger2.zip | "Frogger (modified Moon Cresta hardware)" |
| froggers | froggers.zip | "Frog" |
| frogs | frogs.zip | "Frogs" |
| frogsega | frogsega.zip | "Frogger (set 2)" |
| frontlin | frontlin.zip | "Front Line" |
| futspy | futspy.zip | "Future Spy" |
| galaga | galaga.zip | "Galaga (Namco)" |
| galagab2 | galagab2.zip | "Galaga (bootleg)" |
| galagads | galagads.zip | "Galaga (fast shoot)" |
| galagamw | galagamw.zip | "Galaga (Midway)" |
| galap1 | galap1.zip | "Space Invaders Galactica" |
| galap4 | galap4.zip | "Galaxian Part 4" |
| galapx | galapx.zip | "Galaxian Part X" |
| galaxian | galaxian.zip | "Galaxian (Namco)" |
| gallag | gallag.zip | "Gallag" |
| galmidw | galmidw.zip | "Galaxian (Midway)" |
| galnamco | galnamco.zip | "Galaxian (Namco, modified)" |
| galturbo | galturbo.zip | "Galaxian Turbo" |
| galxwars | galxwars.zip | "Galaxy Wars" |
| gardia | gardia.zip | "Gardia" |
| gaunt2 | gaunt2.zip | "Gauntlet 2" |
| gaunt2p | gaunt2p.zip | "Gauntlet (2 Players)" |
| gauntir1 | gauntir1.zip | "Gauntlet (Intermediate Release 1)" |
| gauntir2 | gauntir2.zip | "Gauntlet (Intermediate Release 2)" |
| gauntlet | gauntlet.zip | "Gauntlet" |
| gberet | gberet.zip | "Green Beret" |
| gcastle | gcastle.zip | "Golden Castle" |
| gemini | gemini.zip | "Gemini Wing" |
| getstar | getstar.zip | "Get Star (bootleg)" |
| ghostb | ghostb.zip | "The Real Ghostbusters" |
| gladiatr | gladiatr.zip | "Gladiator" |
| gmissile | gmissile.zip | "Guided Missile" |
| gng | gng.zip | "Ghosts'n Goblins (World?)" |
| gngcross | gngcross.zip | "Ghosts'n Goblins (Cross)" |
| gngjap | gngjap.zip | "Makai-Mura" |
| gngt | gngt.zip | "Ghosts'n Goblins (US)" |
| goldstar | goldstar.zip | "Golden Star" |
| goldstbl | goldstbl.zip | "Golden Star (Blue version)" |
| gorf | gorf.zip | "Gorf" |
| gorfpgm1 | gorfpgm1.zip | "Gorf (Program 1)" |
| gradius | gradius.zip | "Gradius" |
| gravitar | gravitar.zip | "Gravitar" |
| grescue | grescue.zip | "Galaxy Rescue (bootleg?)" |
| gridiron | gridiron.zip | "Gridiron Fight" |
| gryzorb | gryzorb.zip | "Contra (Japan bootleg)" |
| gsword | gsword.zip | "Great Swordsman" |

| ***Identificador*** | ***Fichero ZIP*** | ***Nombre del juego*** |
|---|---|---|
| gundeala | gundeala.zip | "Gun Dealer (set 2)" |
| gundealr | gundealr.zip | "Gun Dealer (set 1)" |
| gunfight | gunfight.zip | "Gunfight" |
| gunsmoka | gunsmoka.zip | "Gunsmoke (US set 2)" |
| gunsmoke | gunsmoke.zip | "Gunsmoke (World)" |
| gunsmokj | gunsmokj.zip | "Gunsmoke (Japan)" |
| gunsmrom | gunsmrom.zip | "Gunsmoke (US set 1)" |
| guzzler | guzzler.zip | "Guzzler" |
| gwarrior | gwarrior.zip | "Galactic Warriors" |
| gypsyjug | gypsyjug.zip | "Gypsy Juggler" |
| gyruss | gyruss.zip | "Gyruss (Konami)" |
| gyrussce | gyrussce.zip | "Gyruss (Centuri)" |
| hangly | hangly.zip | "Hangly Man" |
| hardhat | hardhat.zip | "Hard Hat" |
| headon | headon.zip | "Head On" |
| heiankyo | heiankyo.zip | "Heiankyo Alien" |
| helifira | helifira.zip | "Heli Fire (revision A)" |
| helifire | helifire.zip | "Heli Fire (revision B)" |
| hexa | hexa.zip | "Hexa" |
| higemaru | higemaru.zip | "HigeMaru" |
| hustle | hustle.zip | "Hustle" |
| hustler | hustler.zip | "Vídeo Hustler" |
| hvymetal | hvymetal.zip | "Heavy Metal" |
| hyperspt | hyperspt.zip | "HyperSports" |
| hyprolyb | hyprolyb.zip | "Hyper Olympic (bootleg)" |
| hyprolym | hyprolym.zip | "Hyper Olympic" |
| imsorry | imsorry.zip | "I'm Sorry (US)" |
| imsorryj | imsorryj.zip | "I'm Sorry (Japan)" |
| intrepid | intrepid.zip | "Intrepid" |
| intruder | intruder.zip | "Intruder" |
| invaders | invaders.zip | "Space Invaders" |
| invadpt2 | invadpt2.zip | "Space Invaders Part II (Taito)" |
| invdelux | invdelux.zip | "Space Invaders Part II (Midway)" |
| invds | invds.zip | "Invinco / Deep Scan" |
| invho2 | invho2.zip | "Invinco / Head On 2" |
| invinco | invinco.zip | "Invinco" |
| invrvnga | invrvnga.zip | "Invader's Revenge (Dutchford)" |
| invrvnge | invrvnge.zip | "Invader's Revenge" |
| ironhors | ironhors.zip | "Iron Horse" |
| jack | jack.zip | "Jack the Giantkiller (set 1)" |
| jacka | jacka.zip | "Jack the Giantkiller (set 2)" |
| jackal | jackal.zip | "Jackal" |
| jackrab2 | jackrab2.zip | "Jack Rabbit (set 2)" |
| jackrabs | jackrabs.zip | "Jack Rabbit (special)" |
| jackrabt | jackrabt.zip | "Jack Rabbit (set 1)" |
| jailbrek | jailbrek.zip | "Jail Break" |
| japirem | japirem.zip | "Gingateikoku No Gyakushu" |
| jbugsega | jbugsega.zip | "Jump Bug (bootleg)" |
| jedi | jedi.zip | "Return of the Jedi" |
| jhunt | jhunt.zip | "Jungle Hunt" |

| *Identificador* | *Fichero ZIP* | *Nombre del juego* |
|---|---|---|
| jjack | jjack.zip | "Jumping Jack" |
| joust | joust.zip | "Joust (White/Green label)" |
| joustr | joustr.zip | "Joust (Solid Red label)" |
| joustwr | joustwr.zip | "Joust (White/Red label)" |
| jrpacman | jrpacman.zip | "Jr. Pac Man" |
| jumpbug | jumpbug.zip | "Jump Bug" |
| jumpcoas | jumpcoas.zip | "Jump Coaster" |
| junglek | junglek.zip | "Jungle King" |
| jungler | jungler.zip | "Jungler (Konami)" |
| junglers | junglers.zip | "Jungler (Stern)" |
| junofrst | junofrst.zip | "Juno First" |
| kamikaze | kamikaze.zip | "Kamikaze" |
| kangarob | kangarob.zip | "Kangaroo (bootleg)" |
| kangaroo | kangaroo.zip | "Kangaroo" |
| kaos | kaos.zip | "Kaos" |
| karatedo | karatedo.zip | "Taisen Karate Dou (VS version)" |
| karnov | karnov.zip | "Karnov (US)" |
| karnovj | karnovj.zip | "Karnov (Japan)" |
| kchamp | kchamp.zip | "Karate Champ" |
| kchampvs | kchampvs.zip | "Karate Champ (VS version)" |
| kicker | kicker.zip | "Kicker" |
| kickridr | kickridr.zip | "Kick Rider" |
| killcom | killcom.zip | "Killer Comet" |
| kingball | kingball.zip | "King & Balloon" |
| kingofb | kingofb.zip | "King of Boxer (English)" |
| kingofbj | kingofbj.zip | "King of Boxer (Japanese?)" |
| klax | klax.zip | "Klax (set 1)" |
| klax2 | klax2.zip | "Klax (set 2)" |
| klax3 | klax3.zip | "Klax (set 3)" |
| konamigt | konamigt.zip | "Konami GT" |
| kram | kram.zip | "Kram (set 1)" |
| kram2 | kram2.zip | "Kram (set 2)" |
| krull | krull.zip | "Krull" |
| ktiger | ktiger.zip | "Kyukyoku Tiger" |
| kungfub | kungfub.zip | "Kung Fu Master (bootleg set 1)" |
| kungfub2 | kungfub2.zip | "Kung Fu Master (bootleg set 2)" |
| kungfud | kungfud.zip | "Kung Fu Master (Data East)" |
| kungfum | kungfum.zip | "Kung Fu Master" |
| kuniokub | kuniokub.zip | "Nekketsu Kouha Kunio Kun (Jap bootleg)" |
| ladybug | ladybug.zip | "Lady Bug" |
| ladybugb | ladybugb.zip | "Lady Bug (bootleg)" |
| lagunar | lagunar.zip | "Laguna Racer" |
| lazercmd | lazercmd.zip | "Lazer Command" |
| leprechn | leprechn.zip | "Leprechaun" |
| liberatr | liberatr.zip | "Liberator" |
| lizwiz | lizwiz.zip | "Lizard Wizard" |
| llander | llander.zip | "Lunar Lander (rev 2)" |
| llander1 | llander1.zip | "Lunar Lander (rev 1)" |
| lnc | lnc.zip | "Lock'n'Chase" |
| locomotn | locomotn.zip | "Loco-Motion" |

| *Identificador* | *Fichero ZIP* | *Nombre del juego* |
|---|---|---|
| losttmbh | losttmbh.zip | "Lost Tomb (hard)" |
| losttomb | losttomb.zip | "Lost Tomb (easy)" |
| lottofun | lottofun.zip | "Lotto Fun" |
| lrescue | lrescue.zip | "Lunar Rescue" |
| lupin3 | lupin3.zip | "Lupin III" |
| lwings | lwings.zip | "Legendary Wings (US set 1)" |
| lwings2 | lwings2.zip | "Legendary Wings (US set 2)" |
| lwingsjp | lwingsjp.zip | "Legendary Wings (Japan)" |
| m4 | m4.zip | "M-4" |
| m79amb | m79amb.zip | "M79 Ambush" |
| maketrax | maketrax.zip | "Make Trax" |
| mappy | mappy.zip | "Mappy (US)" |
| mappyjp | mappyjp.zip | "Mappy (Japan)" |
| marble | marble.zip | "Marble Madness (set 1)" |
| marble2 | marble2.zip | "Marble Madness (set 2)" |
| marblea | marblea.zip | "Marble Madness (set 3)" |
| mario | mario.zip | "Mario Bros." |
| masao | masao.zip | "Masao" |
| matmania | matmania.zip | "Mat Mania" |
| maze | maze.zip | "Amazing Maze" |
| mazeh | mazeh.zip | "Maze Hunter (Japan)" |
| medlanes | medlanes.zip | "Meadows Lanes" |
| megaforc | megaforc.zip | "Mega Force" |
| megatack | megatack.zip | "MegaTack" |
| mhavoc | mhavoc.zip | "Major Havoc (rev 3)" |
| mhavoc2 | mhavoc2.zip | "Major Havoc (rev 2)" |
| mhavocrv | mhavocrv.zip | "Major Havoc (Return to Vax)" |
| mikie | mikie.zip | "Mikie" |
| mikiehs | mikiehs.zip | "Mikie (High School Graffiti)" |
| mikiej | mikiej.zip | "Shinnyuushain Tooru-kun" |
| milliped | milliped.zip | "Millipede" |
| minefld | minefld.zip | "Minefield" |
| missile | missile.zip | "Missile Command (set 1)" |
| missile2 | missile2.zip | "Missile Command (set 2)" |
| monkeyd | monkeyd.zip | "Monkey Donkey" |
| monsterb | monsterb.zip | "Monster Bash" |
| monwar2a | monwar2a.zip | "Moon War II (set 2)" |
| monymony | monymony.zip | "Money Money" |
| moonal2 | moonal2.zip | "Moon Alien Part 2" |
| moonal2b | moonal2b.zip | "Moon Alien Part 2 (older version)" |
| mooncrgx | mooncrgx.zip | "Moon Cresta (bootleg on Galaxian hardware)" |
| mooncrsb | mooncrsb.zip | "Moon Cresta (bootleg)" |
| mooncrsg | mooncrsg.zip | "Moon Cresta (Gremlin)" |
| mooncrst | mooncrst.zip | "Moon Cresta (Nichibutsu)" |
| moonqsr | moonqsr.zip | "Moon Quasar" |
| moonwar2 | moonwar2.zip | "Moon War II (set 1)" |
| motorace | motorace.zip | "MotoRace USA" |
| motos | motos.zip | "Motos" |
| mpatrol | mpatrol.zip | "Moon Patrol" |
| mpatrolw | mpatrolw.zip | "Moon Patrol (Williams)" |

| *Identificador* | *Fichero ZIP* | *Nombre del juego* |
|---|---|---|
| mplanets | mplanets.zip | "Mad Planets" |
| mranger | mranger.zip | "Moon Ranger" |
| mrdo | mrdo.zip | "Mr. Do! (Universal)" |
| mrdofix | mrdofix.zip | "Mr. Do! (bugfixed)" |
| mrdot | mrdot.zip | "Mr. Do! (Taito)" |
| mrdoy | mrdoy.zip | "Mr. Do! (Yukidaruma)" |
| mrdu | mrdu.zip | "Mr. Du!" |
| mrlo | mrlo.zip | "Mr. Lo!" |
| mrtnt | mrtnt.zip | "Mr. TNT" |
| mrviking | mrviking.zip | "Mister Viking" |
| mspacatk | mspacatk.zip | "Miss Pac Plus" |
| mspacman | mspacman.zip | "Ms. Pac Man" |
| mtrap | mtrap.zip | "Mouse Trap" |
| myhero | myhero.zip | "My Hero" |
| myheroj | myheroj.zip | "Seishun Scandal" |
| mystston | mystston.zip | "Mysterious Stones" |
| namcopac | namcopac.zip | "Pac Man (Namco)" |
| naughtyb | naughtyb.zip | "Naughty Boy" |
| nemesis | nemesis.zip | "Nemesis (hacked?)" |
| nemesuk | nemesuk.zip | "Nemesis (UK)" |
| nibbler | nibbler.zip | "Nibbler (set 1)" |
| nibblera | nibblera.zip | "Nibbler (set 2)" |
| ninjak2a | ninjak2a.zip | "Ninja Kid II (alternate)" |
| ninjakd2 | ninjakd2.zip | "Ninja Kid II" |
| nitedrvr | nitedrvr.zip | "Night Driver" |
| nova2001 | nova2001.zip | "Nova 2001" |
| nprinceb | nprinceb.zip | "Ninja Princess (bootleg)" |
| nprinces | nprinces.zip | "Ninja Princess" |
| nrallyx | nrallyx.zip | "New Rally X" |
| ogonsiro | ogonsiro.zip | "Ohgon no Siro" |
| omegrace | omegrace.zip | "Omega Race" |
| oscar | oscar.zip | "Psycho-Nics Oscar" |
| oscarj | oscarj.zip | "Psycho-Nics Oscar (Japan)" |
| ozmawars | ozmawars.zip | "Ozma Wars" |
| pacland | pacland.zip | "Pac-Land (set 1)" |
| pacland2 | pacland2.zip | "Pac-Land (set 2)" |
| pacland3 | pacland3.zip | "Pac-Land (set 3)" |
| paclandm | paclandm.zip | "Pac-Land (Midway)" |
| pacman | pacman.zip | "Pac Man (Midway)" |
| pacmanbl | pacmanbl.zip | "Pac Man (bootleg on Pisces hardware)" |
| pacmanjp | pacmanjp.zip | "Pac Man (Namco, alternate)" |
| pacmod | pacmod.zip | "Pac Man (harder)" |
| pacnpal | pacnpal.zip | "Pac & Pal" |
| pacplus | pacplus.zip | "Pac Man Plus" |
| panic | panic.zip | "Space Panic (set 1)" |
| panica | panica.zip | "Space Panic (set 2)" |
| panicger | panicger.zip | "Space Panic (German)" |
| paperboy | paperboy.zip | "Paperboy" |
| pbactio2 | pbactio2.zip | "Pinball Action (set 2)" |
| pbaction | pbaction.zip | "Pinball Action (set 1)" |

| ***Identificador*** | ***Fichero ZIP*** | ***Nombre del juego*** |
|---|---|---|
| pballoon | pballoon.zip | "Pioneer Balloon" |
| pengo | pengo.zip | "Pengo (set 1)" |
| pengo2 | pengo2.zip | "Pengo (set 2)" |
| pengo2u | pengo2u.zip | "Pengo (set 2 not encrypted)" |
| penta | penta.zip | "Penta" |
| pepper2 | pepper2.zip | "Pepper II" |
| phantom2 | phantom2.zip | "Phantom II" |
| phoenix | phoenix.zip | "Phoenix (Amstar)" |
| phoenix3 | phoenix3.zip | "Phoenix (T.P.N.)" |
| phoenixc | phoenixc.zip | "Phoenix (IRECSA, G.G.I Corp)" |
| phoenixt | phoenixt.zip | "Phoenix (Taito)" |
| pickin | pickin.zip | "Pickin'" |
| pingpong | pingpong.zip | "Ping Pong" |
| piranha | piranha.zip | "Piranha" |
| pisces | pisces.zip | "Pisces" |
| pitfall2 | pitfall2.zip | "Pitfall II" |
| pitfallu | pitfallu.zip | "Pitfall II (not encrypted)" |
| pkunwar | pkunwar.zip | "Penguin-Kun Wars (US)" |
| pkunwarj | pkunwarj.zip | "Penguin-Kun Wars (Japan)" |
| pleiadce | pleiadce.zip | "Pleiads (Centuri)" |
| pleiads | pleiads.zip | "Pleiads (Tehkan)" |
| polaris | polaris.zip | "Polaris (set 1)" |
| polarisa | polarisa.zip | "Polaris (set 2)" |
| ponpoko | ponpoko.zip | "Ponpoko" |
| pootan | pootan.zip | "Pootan" |
| pooyan | pooyan.zip | "Pooyan (Konami)" |
| pooyans | pooyans.zip | "Pooyan (Stern)" |
| popflame | popflame.zip | "Pop Flamer" |
| portman | portman.zip | "Port Man" |
| potogold | potogold.zip | "Pot of Gold" |
| psychic5 | psychic5.zip | "Psychic 5" |
| puckman | puckman.zip | "Puck Man" |
| pulsar | pulsar.zip | "Pulsar" |
| qix | qix.zip | "Qix" |
| qix2 | qix2.zip | "Qix II (Tournament)" |
| quantum | quantum.zip | "Quantum (rev 2)" |
| quantum1 | quantum1.zip | "Quantum (rev 1)" |
| qwakprot | qwakprot.zip | "Qwak (prototype)" |
| radarscp | radarscp.zip | "Radar Scope" |
| rallyx | rallyx.zip | "Rally X" |
| reactor | reactor.zip | "Reactor" |
| redalert | redalert.zip | "Red Alert" |
| redbaron | redbaron.zip | "Red Baron" |
| redufo | redufo.zip | "Defend the Terra Attack on the Red UFO" |
| regulus | regulus.zip | "Regulus" |
| regulusu | regulusu.zip | "Regulus (not encrypted)" |
| renegade | renegade.zip | "Renegade (US)" |
| rescue | rescue.zip | "Rescue" |
| rf2 | rf2.zip | "RF2" |
| ringkin2 | ringkin2.zip | "Ring King (set 2)" |

| *Identificador* | *Fichero ZIP* | *Nombre del juego* |
|---|---|---|
| ringking | ringking.zip | "Ring King (set 1)" |
| roadf | roadf.zip | "Road Fighter (set 1)" |
| roadf2 | roadf2.zip | "Road Fighter (set 2)" |
| robby | robby.zip | "Robby Roto" |
| robotbwl | robotbwl.zip | "Robot Bowl" |
| robotron | robotron.zip | "Robotron (Solid Blue label)" |
| robotryo | robotryo.zip | "Robotron (Yellow/Orange label)" |
| rocnrope | rocnrope.zip | "Roc'n Rope" |
| rollingc | rollingc.zip | "Rolling Crash / Moon Base" |
| roundup | roundup.zip | "Round-Up" |
| route16 | route16.zip | "Route 16" |
| rpatrolb | rpatrolb.zip | "River Patrol (bootleg)" |
| rushatck | rushatck.zip | "Rush'n Attack" |
| rygar | rygar.zip | "Rygar (US set 1)" |
| rygar2 | rygar2.zip | "Rygar (US set 2)" |
| rygarj | rygarj.zip | "Rygar (Japan)" |
| safari | safari.zip | "Safari" |
| samurai | samurai.zip | "Samurai" |
| sasuke | sasuke.zip | "Sasuke vs. Commander" |
| satansat | satansat.zip | "Satan of Saturn" |
| sauro | sauro.zip | "Sauro" |
| savgbees | savgbees.zip | "Savage Bees" |
| sbagman | sbagman.zip | "Super Bagman" |
| sbagmans | sbagmans.zip | "Super Bagman (Stern)" |
| sbasketb | sbasketb.zip | "Super Basketball" |
| sboblbob | sboblbob.zip | "Super Bobble Bobble" |
| sbrkout | sbrkout.zip | "Super Breakout" |
| schaser | schaser.zip | "Space Chaser" |
| scion | scion.zip | "Scion" |
| scionc | scionc.zip | "Scion (Cinematronics)" |
| scobra | scobra.zip | "Super Cobra (Stern)" |
| scobrab | scobrab.zip | "Super Cobra (bootleg)" |
| scobrak | scobrak.zip | "Super Cobra (Konami)" |
| scramble | scramble.zip | "Scramble" |
| scregg | scregg.zip | "Scrambled Egg" |
| sdungeon | sdungeon.zip | "Space Dungeon" |
| seawolf | seawolf.zip | "Sea Wolf" |
| seawolf2 | seawolf2.zip | "Sea Wolf II" |
| sectionz | sectionz.zip | "Section Z" |
| seganinj | seganinj.zip | "Sega Ninja" |
| seganinu | seganinu.zip | "Sega Ninja (not encrypted)" |
| sfeverbw | sfeverbw.zip | "Space Fever (B&W)" |
| shaolins | shaolins.zip | "Shao-Lin's Road" |
| sharkatt | sharkatt.zip | "Shark Attack" |
| shootout | shootout.zip | "Shoot Out" |
| sidearjp | sidearjp.zip | "Sidearms (Japan)" |
| sidearmr | sidearmr.zip | "Sidearms (US)" |
| sidearms | sidearms.zip | "Sidearms (World)" |
| sidepckt | sidepckt.zip | "Side Pocket" |
| sidetrac | sidetrac.zip | "Side Track" |

| *Identificador* | *Fichero ZIP* | *Nombre del juego* |
|---|---|---|
| silkworm | silkworm.zip | "Silkworm (set 1)" |
| silkwrm2 | silkwrm2.zip | "Silkworm (set 2)" |
| silvland | silvland.zip | "Silver Land" |
| sinista1 | sinista1.zip | "Sinistar (prototype version)" |
| sinista2 | sinista2.zip | "Sinistar (revision 2)" |
| sinistar | sinistar.zip | "Sinistar (revision 3)" |
| sinvemag | sinvemag.zip | "Super Invaders" |
| slapbtjp | slapbtjp.zip | "Slap Fight (Japanese bootleg)" |
| slapbtuk | slapbtuk.zip | "Slap Fight (English bootleg)" |
| slapfigh | slapfigh.zip | "Slap Fight" |
| snapjack | snapjack.zip | "Snap Jack" |
| snowbro2 | snowbro2.zip | "Snow Bros (set 2)" |
| snowbros | snowbros.zip | "Snow Bros (set 1)" |
| solarwar | solarwar.zip | "Solar Warrior" |
| solomon | solomon.zip | "Solomon's Key (Japan)" |
| sonson | sonson.zip | "Son Son" |
| spacduel | spacduel.zip | "Space Duel" |
| spaceatt | spaceatt.zip | "Space Attack II" |
| spacefb | spacefb.zip | "Space Firebird" |
| spacefev | spacefev.zip | "Space Fever" |
| spaceod | spaceod.zip | "Space Odyssey" |
| spaceph | spaceph.zip | "Space Phantoms" |
| spaceplt | spaceplt.zip | "Space Pilot" |
| spaceskr | spaceskr.zip | "Space Seeker" |
| spacetrk | spacetrk.zip | "Space Trek (Upright)" |
| spacezap | spacezap.zip | "Space Zap" |
| spacfura | spacfura.zip | "Space Fury (revision A)" |
| spacfury | spacfury.zip | "Space Fury (revision C)" |
| spcenctr | spcenctr.zip | "Space Encounters" |
| speakres | speakres.zip | "Speak & Rescue" |
| spectar | spectar.zip | "Spectar (revision 3)" |
| spectar1 | spectar1.zip | "Spectar (revision 1?)" |
| speedbal | speedbal.zip | "Speed Ball" |
| spiders | spiders.zip | "Spiders" |
| spiero | spiero.zip | "Super Pierrot (Japan)" |
| splat | splat.zip | "Splat" |
| sprint1 | sprint1.zip | "Sprint 1" |
| sprint2 | sprint2.zip | "Sprint 2" |
| sptrekct | sptrekct.zip | "Space Trek (Cocktail)" |
| sqbert | sqbert.zip | "FHMC Q*Bert" |
| sqixbl | sqixbl.zip | "Super Qix (bootleg)" |
| srumbler | srumbler.zip | "Speed Rumbler (set 1)" |
| srumblr2 | srumblr2.zip | "Speed Rumbler (set 2)" |
| sspaceat | sspaceat.zip | "Sega Space Attack" |
| ssprint | ssprint.zip | "Super Sprint" |
| stactics | stactics.zip | "Space Tactics" |
| starforc | starforc.zip | "Star Force" |
| stargate | stargate.zip | "Stargate" |
| starjack | starjack.zip | "Star Jacker (Sega)" |
| starjacs | starjacs.zip | "Star Jacker (Stern)" |

| ***Identificador*** | ***Fichero ZIP*** | ***Nombre del juego*** |
|---|---|---|
| startrek | startrek.zip | "Star Trek" |
| starwars | starwars.zip | "Star Wars" |
| stinger | stinger.zip | "Stinger" |
| stratgyb | stratgyb.zip | "Strong X" |
| stratgyx | stratgyx.zip | "Strategy X" |
| stratvox | stratvox.zip | "Stratovox" |
| subs | subs.zip | "Subs" |
| sucasino | sucasino.zip | "Super Casino" |
| superbon | superbon.zip | "Super Bond" |
| superg | superg.zip | "Super Galaxians" |
| superpac | superpac.zip | "Super Pac-Man (Midway)" |
| superpcn | superpcn.zip | "Super Pac-Man (Namco)" |
| superqix | superqix.zip | "Super Qix" |
| suprmatk | suprmatk.zip | "Super Missile Attack" |
| suprmous | suprmous.zip | "Super Mouse" |
| swat | swat.zip | "SWAT" |
| swimmer | swimmer.zip | "Swimmer (set 1)" |
| swimmera | swimmera.zip | "Swimmer (set 2)" |
| sxevious | sxevious.zip | "Super Xevious" |
| szaxxon | szaxxon.zip | "Super Zaxxon" |
| tacscan | tacscan.zip | "Tac/Scan" |
| tagteam | tagteam.zip | "Tag Team Wrestling" |
| tankbatt | tankbatt.zip | "Tank Battalion" |
| targ | targ.zip | "Targ" |
| tazmania | tazmania.zip | "Tazz-Mania" |
| teddybb | teddybb.zip | "TeddyBoy Blues" |
| tehkanwc | tehkanwc.zip | "Tehkan World Cup" |
| terracra | terracra.zip | "Terra Cresta (YM2203)" |
| terracre | terracre.zip | "Terra Cresta (YM3526)" |
| theend | theend.zip | "The End" |
| theglob | theglob.zip | "The Glob (Pac Man hardware)" |
| thepit | thepit.zip | "The Pit" |
| tigerh | tigerh.zip | "Tiger Heli (set 1)" |
| tigerhb1 | tigerhb1.zip | "Tiger Heli (bootleg 1)" |
| tigerhb2 | tigerhb2.zip | "Tiger Heli (bootleg 2)" |
| timeplt | timeplt.zip | "Time Pilot" |
| timepltc | timepltc.zip | "Time Pilot (Centuri)" |
| timetunl | timetunl.zip | "Time Tunnel" |
| tinstar | tinstar.zip | "The Tin Star" |
| tiptop | tiptop.zip | "Tip Top" |
| tnzs | tnzs.zip | "The New Zealand Story" |
| tnzs2 | tnzs2.zip | "The New Zealand Story 2" |
| todruaga | todruaga.zip | "Tower of Druaga" |
| tokio | tokio.zip | "Tokio / Scramble Formation" |
| tokiob | tokiob.zip | "Tokio / Scramble Formation (bootleg)" |
| tokisens | tokisens.zip | "Toki no Senshi" |
| tomahaw5 | tomahaw5.zip | "Tomahawk 777 (Revision 5)" |
| tomahawk | tomahawk.zip | "Tomahawk 777 (Revision 1)" |
| topgunbl | topgunbl.zip | "Top Gunner (bootleg)" |
| topgunr | topgunr.zip | "Top Gunner" |

| *Identificador* | *Fichero ZIP* | *Nombre del juego* |
|---|---|---|
| tornbase | tornbase.zip | "Tornado Baseball" |
| tp84 | tp84.zip | "Time Pilot 84 (set 1)" |
| tp84a | tp84a.zip | "Time Pilot 84 (set 2)" |
| trackflc | trackflc.zip | "Track & Field (Centuri)" |
| trackfld | trackfld.zip | "Track & Field" |
| tranqgun | tranqgun.zip | "Tranquilizer Gun" |
| travrusa | travrusa.zip | "Traverse USA" |
| triplep | triplep.zip | "Triple Punch" |
| trojan | trojan.zip | "Trojan (US)" |
| trojanj | trojanj.zip | "Trojan (Japan)" |
| turpin | turpin.zip | "Turpin" |
| turtles | turtles.zip | "Turtles" |
| tutankhm | tutankhm.zip | "Tutankham (Konami)" |
| tutankst | tutankst.zip | "Tutankham (Stern)" |
| twinbee | twinbee.zip | "TwinBee" |
| twincobr | twincobr.zip | "Twin Cobra (Taito)" |
| twincobu | twincobu.zip | "Twin Cobra (Romstar)" |
| uniwars | uniwars.zip | "Uniwars" |
| upndown | upndown.zip | "Up'n Down" |
| vangrdce | vangrdce.zip | "Vanguard (Centuri)" |
| vanguard | vanguard.zip | "Vanguard (SNK)" |
| vastar | vastar.zip | "Vastar (set 1)" |
| vastar2 | vastar2.zip | "Vastar (set 2)" |
| venture | venture.zip | "Venture (set 1)" |
| venture2 | venture2.zip | "Venture (set 2)" |
| venture4 | venture4.zip | "Venture (version 4)" |
| venus | venus.zip | "Venus" |
| vigilant | vigilant.zip | "Vigilante (US)" |
| vigilntj | vigilntj.zip | "Vigilante (Japan)" |
| vsyard | vsyard.zip | "10 Yard Fight (Vs. version)" |
| vulgus | vulgus.zip | "Vulgus (set 1)" |
| vulgus2 | vulgus2.zip | "Vulgus (set 2)" |
| warlord | warlord.zip | "Warlords" |
| warofbug | warofbug.zip | "War of the Bugs" |
| warpwar2 | warpwar2.zip | "Warp Warp (set 2)" |
| warpwarp | warpwarp.zip | "Warp Warp (set 1)" |
| waterski | waterski.zip | "Water Ski" |
| wbdeluxe | wbdeluxe.zip | "Wonder Boy Deluxe" |
| wbml | wbml.zip | "Wonder Boy in Monster Land" |
| wboy | wboy.zip | "Wonder Boy (set 1)" |
| wboy2 | wboy2.zip | "Wonder Boy (set 2)" |
| wboy4 | wboy4.zip | "Wonder Boy (set 4)" |
| wboy4u | wboy4u.zip | "Wonder Boy (set 4 not encrypted)" |
| wboyu | wboyu.zip | "Wonder Boy (not encrypted)" |
| wexpresb | wexpresb.zip | "Western Express (bootleg)" |
| wexpress | wexpress.zip | "Western Express" |
| wiz | wiz.zip | "Wiz" |
| wow | wow.zip | "Wizard of Wor" |
| xevios | xevios.zip | "Xevios" |
| xevious | xevious.zip | "Xevious (Namco)" |

| *Identificador* | *Fichero ZIP* | *Nombre del juego* |
|---|---|---|
| xeviousa | xeviousa.zip | "Xevious (Atari)" |
| xsleena | xsleena.zip | "Xain'd Sleena" |
| xybots | xybots.zip | "Xybots" |
| yankeedo | yankeedo.zip | "Yankee DO! (Two Bit Score)" |
| yard | yard.zip | "10 Yard Fight" |
| yiear | yiear.zip | "Yie Ar Kung Fu" |
| zarzon | zarzon.zip | "Zarzon" |
| zaxxon | zaxxon.zip | "Zaxxon" |
| zektor | zektor.zip | "Zektor" |
| zigzag | zigzag.zip | "Zig Zag (set 1)" |
| zigzag2 | zigzag2.zip | "Zig Zag (set 2)" |
| zoar | zoar.zip | "Zoar" |
| zookeep | zookeep.zip | "Zoo Keeper (set 1)" |
| zookeepa | zookeepa.zip | "Zoo Keeper (set 2)" |
| zzyzzyxx | zzyzzyxx.zip | "Zzyzzyxx" |

## 6.6. NOMBRES DE LAS ROMS

Los nombres de las carpetas o de los ficheros ZIP pueden comprobarse en el fichero "`gamelist.txt`". Los romsets deben de ser los del MAME 0.34 versión final (diciembre 1998).

Para convertir los romsets desde otras versiones distintas del MAME, utiliza el fichero "`clrmame.dat`" junto con el gestor de ROMS ClrMAME Pro, que se puede descargar desde la web: http://www.clrmame.com/

## 6.7. CREDITOS ORIGINALES

- MAME 0.34 versión original para DOS por Nicola Salmoria y el MAME Team (http://www.mame.net).
- Emulador de Z80: Z80Em Portable Zilog Z80 Emulator Copyright (C) Marcel de Kogel 1996,1997.
- Emulador de M6502 por Juergen Buchmueller.
- Emulador I86 David Hedley, modificado por Fabrice Frances.
- Emulador M6809 por John Butler, basado en L.C. Benschop's 6809 Simulator V09.
- M6808 basado en L.C. Benschop's 6809 Simulator V09.
- 80x86 asm M6808 emulator Copyright 1998, Neil Bradley
- M68000 emulator tomado de System 16 Arcade Emulator por Thierry Lescot.
- Emulador de 8039 por Mirko Buffoni, basado en 8048 emulator por Dan Boris.
- Emulador de T-11 Copyright (C) Aaron Giles 1998
- Emulador TMS34010 por Alex Pasadyn y Zsolt Vasvari.
- Emulador TMS9900 por Andy Jones, basado en el código original de Ton Brouwer.
- Emulador TMS5220 por Frank Palazzolo.
- AY-3-8910 basado en el trabajo de la siguiente gente: Ville Hallik, Michael Cuddy, Tatsuyuki Satoh, Fabrice Frances, Nicola Salmoria.
- Emulación de YM-2203 y YM-2151 por Tatsuyuki Satoh.
- YM-2203 basada en OPL por Ishmair
- Emulador POKEY por Ron Fries y la colaboración de Eric Smith, Hedley Rainnie y Sean Trowbridge.
- Información sobre el hardware de sonido de la NES por Jeremy Chadwick y Hedley Rainne.
- Emulación de YM3812 y YM3526 por Carl-Henrik Skårstedt.
- Emulación de YM2610 por Hiromitsu Shioya.

## 6.8. CREDITOS DEL PORT A GP32

Port a GP32 por Franxis (fjmar@hotmail.com) realizado a partir del código fuente del MAME 0.34 versión final (con fecha diciembre de 1998).

## 6.9. DESARROLLO

Desarrollado con:

- **DevKitArm** http://www.devkit.tk/
- Official **GPSDK** by GamePark.
- **GpSoundBuf** http://chn.roarvgm.com/
- **b2fxec** http://www.deadcoderssociety.tk
- **GP32 Converter** http://www.ifrance.com/edorul/gp32/
- **Geepee32** GP32 Emulator http://users.skynet.be/firefly/gp32/
- **MakeSMC** http://users.skynet.be/firefly/gp32/

## 6.10. FALLOS CONOCIDOS

- Sonido no perfecto o incompleto en algunos juegos.
- Lentitud en los juegos más modernos.
- Consumo de memoria si se cargan varios juegos consecutivos, sobre todo si hay algún error durante la carga o la emulación de los juegos. En ese, caso... resetear la consola por favor☺.

## 6.11. A MEJORAR

- Mejorar el sonido.
- Mejorar la velocidad: Esto se puede conseguir optimizando más el código fuente. Además se podría cambiar el core Z80 por el *DrZ80* de Reesy que está realizado en ensamblador ARM. Igualmente se podría cambiar el core 68000 por el *Zyclone* de Dave, igualmente en ensamblador. Sin embargo para estos dos últimos puntos será imprescindible contar con la colaboración de Reesy y Dave.
- Actualizar los ROMSets a los actuales del MAME, ó actualizar a otra versión del MAME más reciente.
- Incluir soporte para más juegos.

## 6.12. AGRADECIMIENTOS


- **Unai**: Gracias Unai, por todas las horas ayudándome de madrugada con el MAME, optimizaciones, etc...
- **Talfi**: Gran amigo que me cede espacio en su web para mi port del MAME Para GP32.
- **Chicho**: Amigo del alma, tenía fé ciega en que el MAME iba a funcionar en la GP32... xD
- **Rlyeh**: El amo y señor de la emulación en GP32... Es el señor de las 'pistas'... xD Pero algunas pistas han sido imprescindibles, así que gracias Rlyeh.
- **Antiriad**: Gracias, Antiriad, por el genial artwork ;-).
- **Baktery**: Consejos con el sonido.
- **Groepaz**: Más consejos con el sonido.
- **Chui**: De gran ayuda durante mi anterior proyecto MultiPac y muy exceptico con este nuevo ¿Con que un infierno eh? xD
- **D_Skywalk**: Gracias a él el Multipac funcionó y gracias a él aprendí grandes cosas de la GP32.
- **Locke**: Buenos consejos, beta-testing...
- **Ron**: Esta como una cabra, pero un gran tipo ;-). El organizador de las MadriDC.
- **Ilarri**: Ánimo ;-).
- **Fox68k**: Más buenos consejos.
- **Anarchy**: Incentivó hacer el MAME por su 'excepticismo' cuando le conté la posibilidad de hacer el port durante la pasada MadriDC 2004.
- La gente del canal **#gp32dev** del IRC-Hispano, beta-testing: Xenon, Mortimor, Nestruo, Dj_Syto, K-Teto, Enkonsierto, etc...
- **J.L.Martín Sánchez**: El tutor del proyecto en la universidad ;-).


## 6.13. WEBS INTERESANTES SOBRE EL MAME

- http://www.mame.net/
- http://www.mameworld.net/
- http://www.marcianitos.org/

## 6.14. WEBS INTERESANTES EN GENERAL

- http://www.talfi.net
- http://www.gp32spain.com
- http://www.gp32x.com

## 6.15. HISTORIAL DE VERSIONES

Versión 1.1 (Abril de 2005):

- Soporte para los juegos de la empresa Williams en un FXE aparte (mwilliam.fxe). 19 nuevos romsets: blaster, bubbles, bubblesr, colony7, colony7a, defcmnd, defence, defender, joust, joustr, joustwr, lottofun, robotron, robotryo, sinista1, sinista2, sinistar, splat, stargate.
- Correcciones de pantalla (Slap Fight, Tiger Heli, Twin Cobra...)
- Optimizaciones de la librería de sonido.
- Incrementado el volumen del sonido.
- Corregido el panning del volumen.
- Optimizaciones menores.
- START+SELECT durante el juego = Reset del juego (por ejemplo es necesario para arrancar los juegos de Williams).
- L+R en el menú de selección = Resetear GP32.
- Disparo con START cuando la pantalla esta rotada.
- Emulados controles analógicos (por ejemplo para Return of the Jedi)
- Segundo joystick emulado.
- Emulador YM3812 siempre activo con sonido activado. Eliminada opción "Sound ON+YM3812".
- Bit de intensidad de la paleta fijado a 0.

Versión 1.0 (Marzo de 2005):

- Primera versión.

# 7. SOPORTE INFORMÁTICO

Se adjunta un CD con el contenido de la memoria y el programa (fuentes y ejecutables). También una copia de todo el software auxiliar y la documentación usada durante el desarrollo.

Dentro del CD se encuentra la siguiente estructura de directorios:

- **\Compilador** : Dentro de este directorio se encuentra el entorno de desarrollo DevKitArm que incluye GCC 3.4.3, así como el componente MSYS. También otras versiones antiguas del entorno de desarrollo.
- **\Documento_TFC** : Aquí esta disponible el documento original en Word de este trabajo fin de carrera.
- **\Documentos** : Varios documentos utilizados durante el proyecto, como la documentación del GPSDK 2.1.0, documentos técnicos sobre los procesadores ARM, etc.
- **\Ejecutables** : El ejecutable del programa del proyecto para GP32, además de versiones antiguas y las versiones originales para DOS y Windows.
- **\Fuentes** : El código fuente del programa del proyecto, así como versiones antiguas y el código fuente original para DOS.
- **\GPSDK** : Las librerías del GPSDK 2.1.0, necesarias para compilar el código fuente del proyecto.
- **\Utilidades** : Diversas utilidades, como el B2FXEC, Geepee32, GP32Converter y MakeSMC128. Por último se incluye el descompresor de archivos WinRAR.

# 8. BIBLIOGRAFÍA

Toda la bibliografía del proyecto se ha conseguido de Internet, y se detalla a continuación:


- **GP32 GPSDK API Referente Ver2.1.0**: Por A-Chi Cheng y Ji-Hyeon Lim de la empresa GamePark Inc. R&D Center. Fecha: 10 de noviembre de 2001. Fuente: http://www.gp32.com/
- **GCC 3.4.3 Online Manual**: Copyright 1988-2004 Free Software Foundation, Inc. Esta disponible en la siguiente dirección web: http://gcc.gnu.org/onlinedocs/gcc-3.4.3/gcc/
- **ARM920T Technical Reference Manual revisión 1**. Ref: DDI0151C, Fecha: 20 de abril de 2001. Fuente: http://www.arm.com/pdfs/DDI0151C_920T_TRM.pdf
- **MAME F.A.Q v4.0**: Original en inglés por Shane R. Monroe y traducido al español por Gabriel Gazzán. Fecha: 30 de mayo de 2000. Página web oficial: http://www.mame.net/mamefaqspanish.html


Además han sido de gran utilidad las siguientes páginas web, para la obtención de diversa información interesante:

- Página oficial coreana de la consola GP32: http://www.gp32.com/
- Página europea de la consola GP32: http://www.gp32eu.com
- Página oficial del emulador Geepee32 para Windows: http://users.skynet.be/firefly/gp32/
- Página Web del compilador GNU GCC: http://gcc.gnu.org/
- Web de desarrollo GNU GCC para ARM: http://www.devkit.tk/ y http://sourceforge.net/projects/devkitpro/
- Web oficial de la gama de procesadores ARM: http://www.arm.com/
- Web en inglés con gran cantidad de información sobre la consola GP32: http://www.gp32x.com/
- La mayor Web en español sobre la GP32: http://www.gp32spain.com/
- Web oficial del emulador de recreativas MAME: http://www.mame.net/