Revista dedicada a la retro informática en general
Esta publicación es gratuita y de libre difusión
Año 4 número 11

# RetroWiki Magazine


## SamaruX

un shell tipo Unix para CP/M

## Inves Spectrum +

## A fondo


## Daffy Duck in Hollywood

## Master System

## 10 Modificaciones y mejoras

para el ZX Spectrum


## Entrevista: David Provencio

## como hacer mi propia recreativa

# Retro Wiki magazine you need


No matter the language, if you want to publish your article, send your article.
We look forward to hearing your input into this hobby.




INFO@RETROWIKI.ES

# El Staff de Retrowiki Magazine

*El Staff de RetroWiki Magazine lo forman los siguientes usuarios:*

## Dirección

ron
jojo073

## Edición

jojo073

## Colaboradores

Willco2009
Mcleod_ideafix
Chernandezba
Fermars
FloppySoftware
Rod

## Web

ron

## Portada


*Portada dedicada al Inves spectrum, una pieza que tratamos de encajar en el universo Sinclair.*

# editorial

***Hola queridos lectores... Comenzamos esta revista con este texto y espero que puedas leer con la misma emoción que aquel niño recibió su primer microordenador o consola de juegos. Lleno de felicidad e incertidumbre por no saber muy bien que le espera, pero que seguro que sera algo que le entusiasmara...***
***Con ese espíritu hemos trabajo duro para traerte este número especial sobre el inves spectrum +, un ordenador polémico del que no se tiene mucha información sobre sus orígenes y objetivo en el mercado, salio cuando Amstrad ya sacaba sus modelos de spectrum e Investronica perdía la licencia para fabricar maquinas Sinclair. Ahora podrás saber casi todo sobre su diseño y características.***

***Pero eso no es todo, tenemos un puñado de buenos artículos que harán las delicias de los amantes de la Retro informática...***

***Ahora volvemos a la actualidad, la Retro informática va camino de pasar de un hobby de unos cuantos a ser un negocio en toda regla, Se anuncian nuevas publicaciones, reediciones de viejos juegos, la prensa se interesa más por estos temas, incluso una cadena de venta de videojuegos anuncia que reparara y venderá consolas clásicas. Esto sin duda nos llevara a un nuevo nivel, por un lado entraran en escena los negociantes, que trataran de imponer su sistema para que todos tengan que pasar por caja y por otro los especuladores que acapararon material y ahora se disponen a ponerlo en circulación a precios astronómicos, así que iros olvidando de pillar un juego de megadrive en un mercadillo y mucho menos en una web de subastas, eso se ha terminado. Por otro lado tendremos mucha mas oferta para nuestras viejas y queridas maquinas, que aunque a precio desorbitado contaran con nuevos juegos... ¿Que podemos hacer al respecto, si no estamos de acuerdo? Pues solo nos queda una cosa, lo que seguimos haciendo, apoyar a la gente que crea proyectos culturares sin animo de lucro. Seguir informando sobre la informática a un buen nivel y esperar que pase la moda...***

***Otro tema es la cantidad de desinformadores que pueblan este pequeño universo, muchos de ellos concentrados en youtube y autodenominados con vocablos anglosajones para darse un aire mas tecnológico, algunos resultan realmente tristes y la verdad es que no seria algo para darle importancia si no fuera por que ellos mismos beben de sus mismas mentiras y de tanto contarlas suenan para muchos a verdades. Es descorazonador escuchar algunos de estos programas donde se impone un espectáculo fruto de la sobra de dinero y tiempo libre... En el caso que os encontréis con algunos de estos personajes nunca tratéis de rectificarlos o apuntarles en el dato correcto, por que seréis atacados sin miramientos... Mejor no moverse por esos espacios... La mayoría pierde el interés pasados unos meses y desaparecen...***

***Para finalizar nuestro mayor recuerdo y homenaje a José María Ponce que falleció recientemente. Muchos lo recordareis de las ilustraciones de microhobby y micromania. Su obra queda para recordarnos el gran artista que sera para siempre...***

# Sumario

RetroWiki Magazine

# ¿Sabías que... ?

## El tesoro perdido de Cuauhtémoc Próximamente para amstrad cpc



Hasta ahora no se sabe mucho del desarrollo de este juego, solo que se esta haciendo en España y que se sera un plataformas con guiños a otros juegos clásicos, pero con personalidad propia se enseñado un

beta con movimiento, según un miembro del equipo, puede que el personaje dispare. En otro orden hay especulaciones de edición fisica pero, segun declara este miembro del equipo, es algo que no se plantean ahora mismo, solo piensan en terminar el juego, sacarlo en edición física estaría bien pero ahora mismo se conforman con la edición virtual.

## Pentacorn Quest para zx spectrum


Creado con La herramienta la churrera Jarlaxe, Nightwolf y McKlain, nos traen este plataformas con toques de puzzle. tiene una gran cantidad de pantallas en

las que tenemos para usar objetos, activar mecanismos y recoger bellotas.

## Castlevania Spectral Interlude para zx Spectrum

Después De una larga espera Sánchez el desarrollador de esta versión del clástico castlevania para spectrum a publicado su trabajo, Este juego es para spectrum 128k y viene en varios idiomas, en español se ha

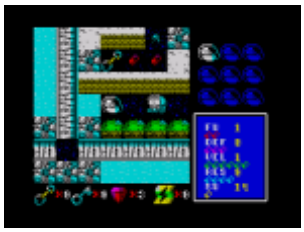
publicado una versión posterior que por lo visto lleva alguna mejoras respecto a la versión publicada originalente.

Añadida versión en español.

La puerta de Vadoma está abierta desde el comienzo.
Medusa ("el jefe verde") ahora es más difícil.
Error corregido: el protagonista era arrojado hacia la derecha luego de colisionar con alguno de los lados.
Error corregido: algunas veces el protagonista quedaba atascado en los muros.
Se corrigieron errores en el texto.

## The Tales Of Grupp para zx spectrum

Código, gráficos y sonido: Alxinho



Grupp es un apuesto y mujeriego caballero de la corte, pero el rey, harto de sus líos de faldas con sus nueve hijas, decide expulsarlo de su ejército y encierra por un largo tiempo a sus nueve hijas bajo llave. Grupp decide secuestrarlas a todas para su venganza aunque no va a ser tan fácil, el rey ha desplegado todo su ejército contra tí siendo consciente de tu intención.

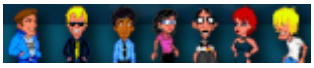
Así se nos presenta este juego para spectrum, con buenos gráficos y un completo set de juego.

## Ya se puede reservar en la web el teclado con forma de zx spectrum


Según su web: http://sinclairzxspectrum.elite-systems.co.uk
Ya se puede hacer una reserva de este teclado para dispositivos android, por un precio nada barato de 100 libras.
La verdad es que es una buena iniciativa, debido a la popularidad del spectrum, pero lo que nos preguntamos es: Si un teclado de características similares suelen vales entre 12 y 30€ ¿ por que este mas de 120€? No se, no se, pagar un pastizal para volver a quejarse de lo que se quejaba todo el mundo con el gomas, el teclado...

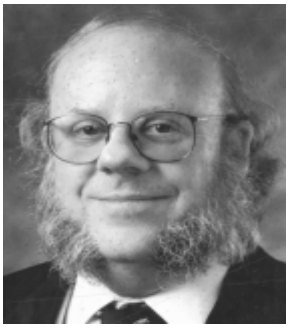
## Night of the Meteor


El grupo Edison Interactive, esta trabajando en una especie de secuela o rehecho de la aventura gráfica de Lucas, Maniac mansion. Muchos recordareis esta aventura con la que Lucas se sabio al tren de las aventuras gráficas que ya empezaban a tener éxito en los



ordenadores de 16 bits.

## MUERE STEVE BRISTOW PADRE DEL MÍTICO VIDEOJUEGO BREAKOUT



# ¿Sabías que... ?

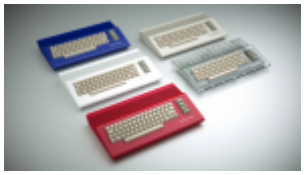
Principal creador de Breakout, Aquí mas conocido

como Arkanoid, uno de los videojuegos más editados de la historia, en multitud de variantes y plataformas. También estuvo en el equipo de trabajó de Tank, otro clásico de Atari 2600
La carrera de Bristow en el medio del entretenimiento electrónico inició cuando todavía era estudiante de la Universidad de California, él y su esposa diseñaron el modo para dos jugadores de Computer Space,

una de las primeras maquinas recreativas de la historia.

## Encuentran los moldes del commoodre 64c

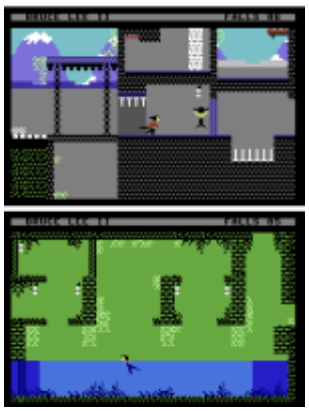
Según se cuenta en la web, Dallas Moore fue a una subasta de artículos de una empresa de inyección y entre todos los moldes para los tupperware encontró el del commodore 64c.



Ahora esta recaudando dinero en una campaña para hacer una tirada de carcasas en varios colores.

## Sacan una versión de Bruce lee II para commoodre 64


Bruce Lee II es una secuela creada por Bruno R Marcos para pc, que presenta dos modos graficos, CPC y C64. Ahora usandola se ha programado la versión de C64. Según hemos podido leer saldrá en versión disco y cartucho.

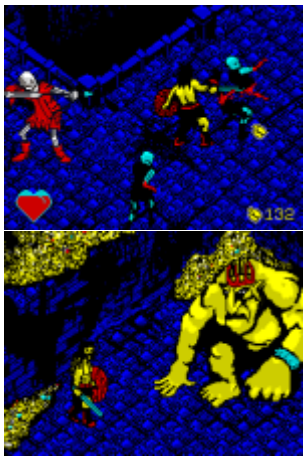


Un poco del argumento: El malvado Tao-Bao ha raptado la hermana de Bruce Lee y la ha llevado a lo mas profundo de su palacio. Allí, un grupo de luchadores reclutados para defender el palacio espera la llegada del Dragón para destruirle...

## Campaña de financiación para el Genesia Legado

En 1993 Microids y Software Toolworks lanzaron Genesia un juego de estrategia por turnos para Amiga. El juego a parte de la gestión de colonizar el nuevo mundo trataba también de la búsqueda de las piedras genesia. Ahora se esta llevando a una campaña para sacar un nueva versión, con gráficos mejorados, se espera que salga en formato digital para: Windows, Mac OS X y Linux) a través de Steam

## SINKH'S LAIR juego on line



Se supone que tiene graficos parecidos a los que produce un spectrum, aunque en cuanto empiezas a jugar se ve que solo se han basado en la paleta de spectrum, ya que no presenta el color class. De todas maneras hay que darle una oportunidad a este juego en el que avanzamos por una mazmorra matando esqueletos, zombies y fantasmas.

## Tower57 un juego inspirado en los clásicos de amiga

Aun no hay fecha de lanzamiento para este juego de acción inspirado en Alien Breed y Chaos Engine de amiga.
Pixel art 16 bits
entornos interactivos, destructibles, modo cooperativo en línea, multiplataforma Mac / Windows / Linux.

# Empezar desde cero o volver a empezar

***Desde que comenzó a hablarse y por ende a popularizarse todo lo referido a ordenadores antiguos, hemos venido siendo testigos de los cambios que se han producido en los conceptos que definen el gusto, afición o interés en los microordenadores de los años 80.***

Hemos sido espectadores de lo que ha venido aconteciendo hasta la fecha y lo que nunca hemos hecho es convertirnos en actores o en protagonistas y eso ahora es lo que toca. Digo lo de toca porque en realidad somos los usuarios los que hacemos que se hable y se mueva todo en torno a este fenómeno.
Para ponernos en situación hay tres prototipos de usuario ( aunque no nos gusten las etiquetas ) retro, vintage, viejuno, antiguo o de otra época y son:
A) El usuario que tuvo un micro en los 80

B) Aquel que no tuvo un micro en los 80 pero que recuerda la época y las vivencias.

C) El nuevo usuario, ni tuvo ni conoció, simplemente ha descubierto que esto existe.

Y dentro de este resumen hay quienes usaban los micros para aprender informática y programación o exclusivamente para jugar. La industria informática ya existía, basada en grandes ordenadores que requerían de mucho personal y espacio, de una gran organización de personas como analistas orgánicos y funcionales, programadores, operadores, administradores, etc... Que se fueron ampliando con el paso de los años conforme a la evolución y crecimiento de la misma industria.
Algunos de aquellos usuarios hoy en día son profesionales en alguno de los ámbitos de la informática y otros tomaron caminos que nada tienen que ver…

**El usuario de toda la vida…**
Pueden darse dos posibilidades, la primera es que aparte de haber usado micros en los 80, estos fueran conservados y que aún permanezcan en manos de su propietario original. La segunda es la de aquel usuario que bien por edad, estudios, madurez, mudanza u otras aficiones los dejase de lado y los perdiera, que es uno de los casos más comunes. Bien porque se lo regaló a


un hermano, pariente o vecino o bien porque lo vendió. En cualquier caso quien tuvo un micro pudo vivir en primera persona toda la época y los recuerdos permanecen en el canto de la memoria. Fieles testigos de aquel boom que tanto nos encandiló, con toda aquella parafernalia, juegos, libros y manuales, revistas y fancines, periféricos y ampliaciones, cartuchos, cassettes y discos, hacían las delicias de los usuarios y permitían que aquellas largas y oscuras tardes de invierno de los 80 se llevasen de otra manera.

**Aquel que no tuvo un micro…**
Pues es un tema que ha de ser comentado. Hubo quienes lo deseaban con todas sus ganas y ni aun así lo pudieron tener. Comprar un micro era una aventura cara y no todas las familias se lo podían permitir. Recuerdo aquellos compañeros que no

les compraban el Spectrum ni por la fiesta de Reyes Magos, que se lamentaban e intentaban ahorrar todo lo posible. Algunos lo consiguieron, en primer lugar debido a que los equipos de Sinclair eran " los más asequibles " y los que más diversión y entretenimiento ofrecían por menos dinero.
Gracias a los compañeros de clase, vecinos, parientes y en general el medio en el que cada uno de desenvolvía la gente se buscaba la vida y aunque no se tuviera un micro, seguro que el vecino o un compañero de instituto hacía que la magia se produjera ya que tenía uno. Ir a casa de uno y de otro era la tónica natural. Recuerdo haber conocido gente que tenía un montón de cintas con juegos pero que no tenían ordenador y las cambiaban y todo eso como si fueran propietarios de uno y hasta en algunos casos lo terminaron por conseguir. De aquella se valoraba mucho lo de la hucha y cuando había que romper el cerdito era por una causa muy deseada.

**Rara Avis… ¿nuevo usuario?**

Puede sonar raro, pero esto sucede. ¿Qué pasa si un chaval de los que ahora tienen entre 13 y 25 años descubre que estos cacharros existieron y que son la base de la informática doméstica actual y de la industria del ocio videojueguil?
Por pasar pueden suceder muchas cosas, pero la mejor y la más sorprendente es sin duda cuando ese chaval se engancha y decide que quiere tener un micro. Sin saber a qué se enfrenta, que retos le esperan y sobre todo con muchas preguntas e inquietudes. ¿Valdrá la pena la aventura que va a comenzar?, nosotros creemos sin duda que SI.
No solo porque todavía no se ha preservado ni encontrado todo, sino que se sigue día a día redescubriendo material, información, prototipos, proyectos y aparte que se hacen nuevos desarrollos, tanto de software como de hardware. Que nadie nace sabiendo y que solo mediante la interacción con otros usuarios este fenómeno se hace patente.

**¿Y qué ha pasado desde entonces …?**

La evolución de los mercados, las decisiones de los fabricantes, la negra mano de la prensa y la popularización de los micros hizo que los precios bajaran mucho en un corto periodo de tiempo.
Un ZX Spectrum de 48K costaba la friolera de 52.000 pts, unos 300€ al cambio. Y en España pagábamos un peaje muy caro ya que al principio todos eran importados y carecían de distribución oficial, con lo que si había problemas uno debía someterse a los servicios técnicos alternativos, que los había.
Pero es que en el Reino Unido la versión de 48K costaba 175 libras y la de 16K, 125. Es decir que al cambio por unas 26-29.000 pesetas podías tener tu ZX Spectrum. Hablamos de unos 180 € que se asemeja más al precio original fijado por Sinclair para el Reino Unido. Advirtiendo que un salario medio bueno era de 75-80.000 pesetas ya me diréis los esfuerzos que había que hacer para comprar uno. Pues a plazos, con un crédito o con las extintas letras de cambio.
Y eso que estamos hablando del ZX Spectrum. En el Reino unido salieron competidores a precios similares como el caso del Oric 1, costaba lo mismo o menos que un Sinclair, pero es que por 50 libras más uno tenía la opción de adquirir un Dragon o un Camputers y esas 50 libras eran los costes de un teclado profesional, una carcasa grande y un mejor diseño, aunque en términos de electrónica todo era parecido.

Después de saber que hay usuarios, nuevos y viejos, ¿qué tal si tratamos el asunto de Sobre la popularización de la retroinformática?
En RetroWiki WEB hemos hablado y discrepado sobre si la difusión que está teniendo todo lo relacionado con los 'sistemas retro' beneficiaba o perjudicaba al 'movimiento retroinformático'.
No tenemos todavía muy asimilado el concepto de 'movimiento retroinformático', pero creemos que eso es exactamente lo que nosotros hacemos.
Hace 15 años éramos más bien unos pocos los que mostrábamos interés por estos 'viejos' ordenadores, y a mucha gente ajena esto se le hacía raro. Hoy en cambio son muchos los que piensan que 'esto de lo retro mola' y como todo lo que mola se ha puesto de moda.

Lo que está claro que es que la retroinformática se ha convertido en algo más popular, y cuando esto sucede aparecen diferentes formas de vivirla o disfrutarla:
*- Los cacharreros, divulgadores, programadores y jugones de toda la vida que siempre han estado ahí*
*- Los que solo sacan su viejo cacharro para echarse unos vicios o rememorar viejos tiempos.*
*- Los que utilizan viejas glorias y se inspiran para hacer música o video grafismo.*
*- Los fetichistas de turno que se limitan a coleccionar viejos sistemas, o los especuladores que acaparan para tiempos futuros y hacer negocios sustanciales.*

Desde nuestro punto de vista esta popularización que ha tenido lugar los últimos años es buena. Es cierto que tiene daños colaterales y efectos negativos que a algunos puede llegar a resultar incómodo, pero al igual que sucede en muchos otros ámbitos, en el momento que algo tiene éxito aparece la especulación y la retroinformática no es una excepción.
Hay sistemas que hace unos años se podían adquirir a precios razonables y hoy están a precios prohibitivos. Se han convertido en auténticos artículos de lujo, pero ¿de quién es la culpa?, ¿de quienes los promocionamos y divulgamos?, o por el contrario, ¿es la propia corriente que todo esto genera?.

No obstante las cosas no son tan negativas, por otro lado esta popularización ha hecho aumentar la masa crítica que facilita que:

- Haya gente de siempre y nueva interesada en desarrollar juegos y software 'homebrew',
- Se diseñen nuevos periféricos o implementaciones de sistemas retro.
- Se celebren reuniones, quedadas, eventos, ferias y fiestas en casi cada ciudad

Pese a todo esto nos falta casi lo más importante, que esos nuevos usuarios se incorporen a los sitios en donde seguimos cultivando la afición. Esto es aún una asignatura pendiente.
Para mantener este movimiento es necesaria una masa crítica, y dentro de esta habrá diferentes grados de implicación o formas de vivirla. Falta ver que si la gente que se ha incorporado al movimiento retro los últimos tiempos son solamente gente de nuestra generación o si también hay nuevas generaciones interesadas en seguir trasteando y conociendo los "viejos" sistemas.

Lo que nos gusta menos de todo esto es ver a gente encumbrarse como gurús de la retroinformática, como los que cagan mármol, que tienen miles de seguidores en youtube y les da igual rellenar su falta de conocimientos o de realidad inventando lo que sea, todo para seguir en el candelero...
Hay mucha desinformación fruto de esta gente, aunque pasada la moda desaparecerán. Para muchos solo es solo un hobby, una manera de ocupar un tiempo de ocio, para despejar la mente en este terrible tiempo de incertidumbres que nos ha tocado vivir. Lo lamentable es que para algunos sea algo más... una guerra contra todos, por comprar y vender y conseguir, consumir...

***Ron***

# El ZX Spectrum Vega


**Muchos de los aficionados a la retro informática han oído hablar del proyecto zx spectrum Vega, en el que una empresa va a comercializar un spectrum consolizado un spectrum para jugar directamente, sin complicaciones, ponerlo y acceder a 1000 juegos... Si aun no te suena o si quieres saber más este articulo te interesa.**

**Lo primero es agradecer esta oportunidad para los lectores de RW Magazine, poder hacer una serie de preguntas directamente a sus creadores, es todo un lujo.**

**La comunidad de aficionados al spectrum esta muy pendiente al proyecto zx spectrum Vega y nos vienen a la mente muchas preguntas, el objetivo de este articulo es buscar un acercamiento a este proyecto, para ello los lectores nos han enviado sus preguntas y nosotros se las hemos pasado a los desarrolladores del proyecto para ser contestadas en la medida de lo posible, comenzamos...**

**Quisiéramos conocer un poco a la persona que dirige el proyecto y que nos hable sobre su desarrollo:**
**¿Quien es David Levy en el mondo de la retroinformatica?**
Estos últimos 38 años he estado metido en el desarrollo de productos electrónicos de consumo, por ejemplo computadoras de ajedrez, rompecabezas y otros productos. La primera vez que trabajé en este campo en 1977, fui consultor de Texas Instruments y diseñé el programa de ajedrez para el ordenador de 8 bits TI 99/4. Después de ese proyecto en el que también dirigí el desarrollo de, bridge, gin rummy y muchos programas entre ellos, juegos de estrategia para varios equipos domésticos de 8 bits. Estuve muy presente en empresas que desarrollaron el ordenador de 8 bits a principios y a mediados de 1980.

**¿Cual es su implicación en este proyecto?**
Yo soy uno de los fundadores y el presidente de Retro Computers Ltd, la empresa que ha desarrollado y esta detrás de la fabricación de la Vega. Mis socios en la empresa son el propio Sir Clive Sinclair, Paul Andrews, que es el Director General, y Chris Smith, quien es nuestro genio tecnólogo y que ha sido responsable del diseño actual y el desarrollo del hardware y firmware Vega.


**¿Cual es el objetivo del proyecto?**
Pues revivir el ordenador Sinclair Spectrum, un ordenador brillante de la década de 1980, mostrar al mundo que sigue siendo en gran medida un producto válido y que sus juegos son todavía muy jugables.

**Ahora pasamos a las preguntas sobre la maquina propiamente dicha:**

**¿Que tipo de conectores tendrá la maquina, usb, entrad y salida de audio, SD, conector josytick, teclado, salida de vídeo compuesto, HDMI?**
La primera versión del Vega vendrá con salida de vídeo compuesto y audio. Hemos pensado la manera de añadir otras conexiones más adelante, pero no en la versión inicial de la Vega.


**¿Tendrá acceso al basic, para poder programar en la maquina?**
Ahora mismo no. El propósito de la Vega en principio es el de enchufar fácilmente y tener acceso rápido a 1.000 juegos, que los usuarios sean capaces de usar directamente el Vega en sus televisores. Las futuras versiones podrían permitir el acceso al lenguaje BASIC, todo depende de la demanda.

**¿Siempre tiene el mismo basic y rom, o podemos acceder a las roms de distintos spectrum?**
El Vega tiene como misión cargar los juegos para el 48K, 128K y así que tendrá lo necesario para ejecutar los juegos.

**¿Se podrá jugar a juegos conversacionales?**
Supongo que habla de las aventuras de texto, aquí se les llama así. Bien, pues estamos trabajando en un teclado virtual, esto quiere decir que podremos tener un teclado en pantalla, al estilo de los teléfonos y tabletas actuales, donde podremos elegir la tecla a pulsar, así que si, jugar a este tipo de juegos debería ser posible en el Vega.

**¿Al cargar un juego como seleccionamos las opciones, si por ejemplo la selección es con los números o con teclas?**
Todos los juegos serán de fácil acceso a través de menús personalizados para poder jugar en el Vega

**Sabemos que el equipo vendrá con juegos precargados , ¿habrá posibilidad de ampliar este catálogo de alguna forma?**
Sí. Los usuarios podrán añadir otros juegos que puedan descargar o adquirir legalmente en formato digital, y acceder a ellos a través de la ranura para tarjetas SD que tendrá el Vega.

¿Se incluirá algunos de los grandes juegos desarrollados en los últimos años?
Sí. Además de los juegos originales ZX Spectrum de los años 80 también hemos sido capaces de incluir juegos de programadores actuales, como Jonathan Cauldwell por ejemplo.

**Si el producto tiene una buena aceptación , ¿podríamos ver en un futuro una versión con teclado completo?**
Esta no es una pregunta que podamos responder por el momento, pero es siempre una posibilidad. Desde luego, no pretendemos ser una compañía de un solo producto.

**¿Es actualizable el firmware? Por ejemplo, si se descubre un error ¿saldrán actualizaciones que lo corrijan?**
Sí. Esto será posible mediante la tarjeta SD.

**¿Porqué se decidió utilizar emulación en lugar de hacer un clon?**
Queríamos que el ZX Vega pudiera ser tan ampliable como sea posible, desde el punto de vista del hardware actual.

**¿Las funcionalidades se limitarán exclusivamente a las de un zxspectrum 48Kb o se añadirán algunas funcionalidades "extra" tipo las que llevan los emuladores, tal y como son modo "turbo", grabación de snapshoots, etc.?**
Ahora mismo nuestro objetivo es que el Vega pueda ejecutar la mayoría de los juegos que un emulador de Spectrum puede ejecutar.

**¿Habrá algún modo de añadirle hardware compatible al Vega?**
Sí. Existe la posibilidad de añadir otro hardware para el ZX Vega. Hemos proporcionado salidas en el PCB para este propósito.

**¿Cuales serán los canales de venta del producto?**

Para los dos primeros lotes de producción (1,000pcs y 3.000 piezas respectivamente) hemos reservado 2.300 Vegas para las personas que apoyaron nuestra campaña Indie y otras 1.700 personas podrán comprarlos en nuestro sitio web, Esto para ahora mismo. Después de eso vamos a seguir vendiendo en nuestro sitio web. Estamos hablando sobre la distribución con varias empresas que se nos han acercado, esperamos llegar a un acuerdo lo más pronto posible. Tenemos muestras para enviar a los posibles distribuidores. Estamos muy entusiasmados con el potencial de mercado en España, porque como ustedes saben, España fue el segundo país después del Reino Unido en popularidad del spectrum durante la década de 1980, y había muchas casas de software en España de gran calidad, que producían grandes juegos en esa época. Podemos ver el interés por la informática retro en España. El zx spectrum aún tiene una buena base de fans en su país.

**¿Pensaron en emular algo similar al Loki retrocompatible con spectrum o un simil al Sam Coupé?**
Es algo que podríamos considerar para futuras actualizaciones.

**Se ha publicitado que entre los 1000 juegos que dispondrá el ZX Vega, habrá algunos escritos exclusivamente para ella. ¿Tiene ZX Vega características no presentes en los Spectrums originales, que permitan escribir nuevos programas para su ejecución exclusivamente en ZX Vega?**
No hay nuevas características en esta primera versión de la Vega, pero vamos a tener algunos juegos nuevos para el Spectrum Vega, y si un juego se puede ejecutar en un emulador, entonces debería funcionar en la Vega. Habiendo dicho eso, el sistema de actualizaciones mediante la tarjeta SD significa que podemos mejorar las capacidades del Vega en el futuro, y los usuarios podrán disfrutar de las nuevas características que se le sumen al firmware.

**Si es el caso, ¿se espera que podamos ver en un futuro cercano nuevos desarrollos escritos exclusivamente para ZX Vega?**
Estamos considerando muchas posibles opciones futuras para el ZX Vega

**¿Publicarán ustedes el API para usar esas nuevas características, así como las normas para que un juego pueda ser marcado como "ZX Vega ready"?**
Una vez más, esto es algo que podríamos mirar para el futuro.

**En el sitio web del ZX Vega, en la sección de noticias se puede leer lo siguiente (énfasis mío):**
**"We are especially grateful to those of you who have made suggestions as to additional features you would like to see in the Vega, and we have taken all those suggestions seriously, we have discussed them, and we have decided to implement two of them in the first release of the Vega".**
**¿Cuáles son esas dos características adicionales?**
**http://www.zxvega.co.uk/news**
Hemos añadido teclas extra al Vega, hemos añadido una conexión de expansión al Vega con vistas a la creación de futuras expansiones.

**No se si se ha preguntado, pero a mí me gustaría saber qué tienen pensado a medio-largo plazo. Porque si se vende bien, ¿tienen en mente algún otro proyecto futuro?**

Sí. No tenemos la intención de ser una empresa de un solo producto, por lo que podemos esperar que habrá productos futuros, pero quién sabe qué producto sera el siguiente!.
Somos muy conscientes de lo que fue y es el Spectrum en España, por lo que una posibilidad será una edición especial del Vega que contenga muchos juegos desarrollados por empresas de software españolas y programadores españoles. Ya que estamos, nos gustaría ponernos en contactado con los desarrolladores de software español y programadores individuales que sean los dueños de los derechos de algunos juegos de Spectrum, para que podamos hablar y tal vez usar sus juegos en el Vega.

# SamaruX

## un shell tipo Unix para CP/M

***Una cosa que siempre se le ha achacado a CP/M, es que el CCP (Console Command Processor) es más bien espartano y limitado.***
***Si bien es cierto que su utilidad es limitada, no lo es menos también, que ofrecer ciertas funcionalidades con tan poco código, es más bien difícil.***

El CCP se diseñó para ser útil, pero pequeño, ya que, por lo general, los sistemas en los que se ejecuta CP/M, no suelen disponer de unidades de disco de gran capacidad y los programas -incluyendo el propio CP/M - han de ejecutarse en 64 KB de memoria RAM, como mucho (hay sistemas con bastante menos memoria).
Además, hay que tener en cuenta el hecho, de que tras cada arranque en caliente (Warm Boot), se ha de recargar el CCP desde disco, por lo que un CCP de cierto tamaño, requeriría bastante tiempo de carga.
Por todo lo cual, considero que hace un trabajo más que digno, aun cuando se echa en falta de vez en cuando, determinadas funcionalidades (redirección, job control, etc.).


**Sustitutos**
Así pues, al CCP, le han ido saliendo muchos sustitutos (replacements), de diferentes pelajes y con más o menos fortuna.

Quizás el más conocido de todos sea el ZCPR (Z-System) en sus diferentes versiones, aunque también tenemos a C/NIX (The Software Toolworks), ConIX (Computer Helper Industries), Miscroshell (New Generation Systems) y otros proyectos más personales, como Little Shell (Steve Blasingame) o ZCCP (Simeon Cran), entre otros.
Casi todos ellos, han tenido como modelos a los diferentes shell de Unix.

Cabe citar, también, la inmensa cantidad de utilidades tipo Unix que han ido apareciendo (cat, cp, ls, etc.), aunque se sale del objetivo de este modesto artículo.

**Reinventando la rueda**
SamaruX es un proyecto totalmente personal (¿cuáles no lo son?), ya que desde hace bastante tiempo, me rondaba la idea de escribir un shell tipo Unix, para CP/M.
Lo cierto es que, tras esta idea, hay otra mayor: escribir un pseudo-sistema operativo tipo Unix, para CP/M. Tiempo, al tiempo. Ya hay ideas madurando, a partir de SamaruX.
¿Y qué mejor lenguaje de programacion para escribir un shell tipo Unix, que el lenguaje C?
Ya está MESCC llamando a la puerta, otra vez.
MESCC (Mike's Enhanced Small C Compiler) es mi versión de Small C para CP/M y Z80.
Aun cuando hay compiladores mucho mejores (incluso cross-compilers), así todo queda en casa.
Además, hay funcionalidades de SamaruX, que requieren de ciertos ejercicios gimnásticos y de la estrecha colaboración del compilador, programando nuevas bibliotecas de funciones y modificando otras, para añadir y mejorar funcionalidades, no disponibles de otro modo.
En cualquier caso, SamaruX, no pretende reemplazar al CCP, sino complementarlo.
Por cierto, que la idea general viene de BusyBox (www.busybox.net), un shell que implementa internamente un buen conjunto de comandos y

utilidades Unix, en un sólo programa ejecutable, ahorrando de esta manera espacio de disco y memoria, a costa de perder más o menos funcionalidades.

**Funcionalidades**
Ciertas funcionalidades Unix, no son fáciles de implementar en un compilador "normal" para CP/M, si no vienen ya de serie.
Las funcionalidades más importantes a implementar son:
· Redirección y piping en la línea de comando.
· Posibilidad de incluir varios comandos, en una misma línea.
· Variables de entorno.
· Un conjunto de comandos internos tipo Unix (cat, ls, cp, rm, more, etc.).
· Manual de ayuda en línea (man).
· Ejecución de comandos en batch, con parámetros opcionales y control de flujo.
· Inclusión del área de usuario de CP/M en los nombres de fichero.
· Integración en CP/M.

En los apartados siguientes, voy a tratar de explicar cómo se han implementado algunas de estas funcionalidades.

**Redirección y piping**
Ésta es una de esas funcionalidades, que requieren la colaboración estrecha del compilador o, más concretamente, de su biblioteca de funciones.
Precisa de la disponibilidad de los streams stdin, stdout y stderr, así como de la posibilidad de redireccionarlos, lo cual no siempre es cierto.
En MESCC, ha habido que implementarlo, aunque el proceso ha sido bastante indoloro.
Una vez conseguido esto, hubo que escribir el código que gestiona la redirección y el piping en la línea de comandos, lo cual no ha sido fácil, pero el resultado ha sido bueno.
CP/M es un sistema operativo mono-tarea, por lo que el piping se ha implementado mediante ficheros temporales.
Veámoslo con un comando de ejemplo:

```
cat informe.txt | more
```

En primer lugar, la salida de cat al stream stdout, se redirige a un fichero temporal en disco.
En segundo lugar, la entrada a more desde el stream stdin, se toma desde el fichero temporal.
Una vez ha finalizado el comando more, el fichero temporal es eliminado.
Veamos otro ejemplo más complicado:

```
cat informe.txt | grep 'gastos' | more
```

Aquí, el fichero temporal de salida de cat, es la entrada de grep, el cual, a su vez, tiene como salida un fichero temporal, que es la entrada de more.
¿Y qué tiene esto de complicado?
Varias cosas, entre ellas, ¿qué nombre le damos

```
[FloppySoftware at M00:] $ man cat
NAME
        cat - Concatenate files

SYNOPSIS
        cat [-kn] [file ...]

DESCRIPTION
        The command cat writes its input to the standard output.

        If no arguments are given, it reads from the standard
        input.

        The - file means the standard input.

OPTIONS
        -k  Skip empty lines
        -n  Number lines

EXAMPLES
        cat
        cat -kn phrases.txt
        cat -n facts.doc
        cat header.txt - footer.txt > myletter.txt

[FloppySoftware at M00:] $
```

al fichero temporal, si en grep lo necesitamos tanto para la entrada, como para la salida?
Si le damos el mismo nombre, tendremos un conflicto, ya que al crear el de salida, eliminaremos el de entrada, por lo que la solución más lógica, es dar un nombre al fichero de entrada y otro diferente, al de salida.
¿Y qué ocurre, cuando un fichero de salida de un comando, es el fichero de entrada del siguiente?
Que colisionan los nombres, así que hay que proceder a un renombrado previo, antes de ejecutar el siguiente comando.
Supongo que se podría haber implementado mediante un sistema de nombrado automático (por ejemplo PIPE0001.TMP), pero así es cómo ha quedado la cosa.
Y si algo funciona, mejor no tocarlo.

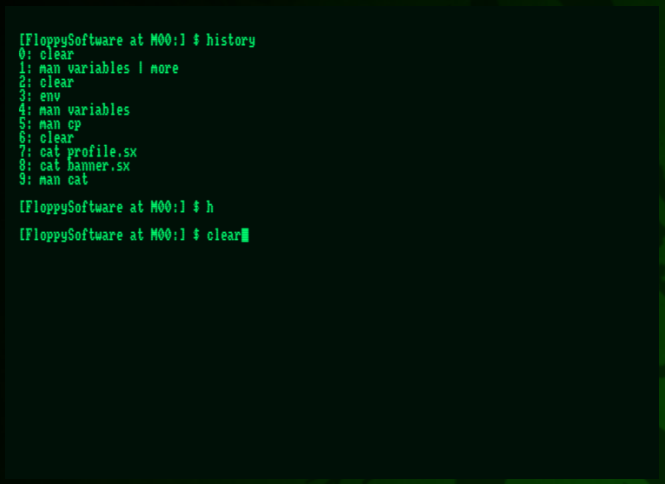
```
if(stdin != NULL)
{
        fclose(stdin);

        if(pipe_in)
                remove(sv_pipe_in);
}

if(stdout != NULL)
{
        fclose(stdout);

        if(pipe_out)
        {
                remove(sv_pipe_in);
                rename(sv_pipe_out, sv_pipe_in);
        }
}
```

**Variables de entorno**
Aun cuando tienen sus detractores, las variables de entorno son muy útiles a la hora de configurar y personalizar el sistema.




CP/M no dispone de variables de entorno, aunque la versión 3 (CP/M Plus) implementa el SCB (System Control Block), del que hacen uso las distintas utilidades del sistema operativo (SETDEF, el CCP, etc.).
Para implementar esta funcionalidad, precisamos de 4 elementos, principalmente:
· Buffer en memoria, para alojar las variables de entorno.
· Un comando que nos permita consultar / crear / modificar / eliminar variables de entorno (env).
· Mecanismos que permitan el acceso a dichas variables, al resto de comandos del sistema (funciones del sistema).
· Utilización de las variables de entorno, en la línea de comandos.
El esquema empleado para el buffer de memoria, se ha desarrollado de tal forma, que puede ser utilizado por otros elementos del sistema (alias, entre ellos).
Se trata de un par de arrays de punteros a strings, uno apuntando a los nombres de las variables de entorno, y otro apuntando a sus valores.
Para su implementación se ha utilizado el binomio malloc() y free(), de tal forma que la gestión de la memoria es dinámica, utilizando únicamente la necesaria en cada momento, y dejando el resto disponible para SamaruX y las utilidades del sistema.

```
/* Environment names */
if((sv_env_name = KeyAlloc(SX_MAX_ENV)) == NULL)
        return ErrorMem();

/* Environment values */
if((sv_env_value = KeyAlloc(SX_MAX_ENV)) == NULL)
        return ErrorMem();
```

Si queremos utilizar una variable de entorno en la línea de comandos, debemos preceder su nombre con $.
Por ejemplo, con el siguiente comando creamos una variable de entorno llamada AGENDA:

```
env AGENDA miagenda.txt
```

Y con éste, lo utilizamos en la línea de comandos:

```
cat $AGENDA
```

Pues internamente, se conviert en:

```
cat miagenda.txt
```

Hay ciertas variables de entorno utilizadas opcionalmente por el sistema y sus comandos, siguiendo el esquema Unix. Algunas de ellas, son:
· $HOME - Directorio por defecto.
· $TERM - Nombre de la terminal.
· $LINES - Nº de líneas de la terminal.
· $COLUMNS - Nº de columnas de la terminal.
· $USER - Nombre de usuario.
· $PROMPT - Cadena configurable del prompt.
· $TMPDIR - Directorio para ficheros temporales.
· $MANPATH - Path para el fichero de ayuda.

Como se ha comentado, su uso es opcional y en el caso de no existir, se toman los valores por defecto en tiempo de compilación:

```
ManFn()
{
        char *fn, *pfn;

        fn = SX_MANFILE;

        /* If MANPATH exists and seems to be a valid path, use it */

        if((pfn = AddPathToFn(EnvGet("MANPATH"), fn)) != NULL)
                return pfn;

        /* Else, use the man file name without path (current directory) */

        return fn;
}
```

También existe una serie de pseudo-variables de entorno.

Por ejemplo, $? nos da el código de salida del último comando ejecutado.

**Manual de ayuda en línea**
Algo muy importante en todo programa que se precie de ser útil, es la documentación.
Ésta ha de ser lo más completa posible y ser accedida con rapidez.
Además, en un entorno susceptible de cambios, la documentación ha de ser fácilmente modificable.
En SamaruX, toda la documentación está en un fichero denominado "man.sx", que puede ser consultado con el comando man.
Además, se ha desarrollado una serie de scripts sencillos, para exportar dicha documentación, a un formato fácilmente editable e imprimible con otros programas.
El contenido del fichero "man.sx", es simple texto plano, por lo que puede ser editado con cualquier programa de edición.
Tan sólo hay que observar las siguentes reglas, para que el comando man pueda manejarlo convenientemente:
· Una línea de texto precedida del carácter '#', es un comentario.
· Una línea de texto precedida del carácter '@', marca el inicio de un topic.
· Una línea de texto precedida del carácter '$', es un título de sección.
Los títulos de sección predefinidos, son:
· # $NM es NAME
· # $SY es SYNOPSIS
· # $DE es DESCRIPTION
· # $OP es OPTIONS
· # $EX es EXAMPLES
· # $NT es NOTES
Veamos un ejemplo:

```
@cat
#
$NM
cat - Concatenate files
$SY
cat [-kn] [file ...]
$DE
The command cat writes its input to the standard output.

If no arguments are given, it reads from the standard
input.

The - file means the standard input.
$OP
-k  Skip empty lines
-n  Number lines
$EX
cat
cat -kn phrases.txt
cat -n facts.doc
cat header.txt - footer.txt > myletter.txt
```

**Ejecución en batch**

```
[FloppySoftware at M00:] $ man variables | more
NAME
        Environment variables

SYNOPSIS
        Environment (& pseudo-environment) variables

DESCRIPTION
        Environment variables can be used in the command line as
        arguments and they will be substituted with their values.

        Pseudo-environment variables are not stored as environment
        variables, but they can be used in command lines as if
        they were.

        Some SamaruX environment variables include:

          $HOME    - Default directory.

          $TERM    - Terminal name.

          $LINES   - Terminal rows.

          $COLUMNS - Terminal columns.

          $USER    - User name.

          $PROMPT  - Prompt string, with some special sequences:

                     %$ : $ symbol.
                     %U : User name (reads $USER).
-- More (q:quit) --
```

El comando batch, permite la ejecución de líneas de comandos, leídas desde un fichero de texto.
Mediante el uso de los comandos if y goto, junto con la definición de etiquetas, podemos controlar el flujo de la ejecución.
La pseudo-variable $# contendrá el nº de argumentos pasados al comando batch, mientras que los propios argumentos (hasta un máximo de 10)estarán en $0, $1, etc., hasta $9.
Por ejemplo, el comando:
batch imprime doc1.txt carta.doc
ejecutará los comandos incluidos en el fichero "imprime.sx", inicializando previamente las pseudo-variables de esta forma:
$# = 2
$0 = doc1.txt
$1 = carta.doc

Podemos incluir comentarios, iniciando las líneas con el carácter '#'.
Las etiquetas se definen dentro de los comentarios, separándolas del carácter '#' con un espacio, y finalizándolas con el carácter ':'.
Veamos un ejemplo:

```
# -------------------------- #
# tty.sx - Print the TTY type #
# -------------------------- #
#
if TERM not in_env goto Unknown
#
echo TTY type is $TERM
goto Quit
# Unknown:
echo TTY type is unknown
# Quit:
```

**Alias**
Ésta no es una característica fundamental, pero se ha implementado por tener cierto atractivo.
El uso del comando alias es muy sencillo.
Por ejemplo, para definir el alias "dir" como "ls -l":

```
alias dir ls -l
```

Puede servirnos también, para poder ejecutar un fichero batch, como una orden normal de SamaruX:

```
alias backup batch backup.sx
```

De esta forma, al introducir "backup" en el inductor del shell, realmente estaremos ejecutando "batch backup".

**Histórico de comandos**

Esta funcionalidad, nos evitará tener que repetir los mismos comandos, una y otra vez.
Mediante el comando history, gestionamos el histórico de líneas de comandos que introducimos en el shell (hasta un máximo de 10).

```
i = atoi(argv[1]);

if(i < 0 || i  >= SX_MAX_HIST)
        return Error("Bad history number");

if(sv_hist[i] != NULL)
{
        strcpy(sv_ed_buf, sv_hist[i]);
        sv_flg_buf = 1;
}
```

Este simple alias, nos permitirá volver a editar el último comando introducido:

```
alias h history 0
```

**Integración en CP/M**

Como ya se ha apuntado, SamaruX no pretende reemplazar al CCP, sino complementarlo.
En cualquier caso, y por comodidad, se permite la ejecución de comandos externos a SamaruX, y el retorno a éste, de forma automática:

```
cpm basic monopoly
```

La ejecución de dichos comandos, se realiza mediante el uso de ficheros "$$$.SUB" gestionados por el propio CCP, por compatibilidad con las distintas versiones de CP/M, aunque ocasiona ciertos problemas.
CP/M Plus permite un método más elegante, mediante la utilización de la función del BDOS 47 / CHAIN TO PROGRAM, por lo que en futuras versiones de SamaruX, es posible que pueda utilizarse este mecanismo de forma opcional.
También podemos ejecutar un único comando de SamaruX, desde el propio CCP:

```
sx ls -l
```

SamaruX hace uso del SCB de CP/M Plus, para integrarse en lo posible:

```
/* HELPER for TermRows() and TermCols()
   ------------------------------------
*/
TermVal(s, scbf)
char *s; int scbf;
{
        char *v, b[2];
        int i;

        if((v = EnvGet(s)) != NULL)
        {
                i = atoi(v);

                if(i > 0 && i < 256)
                        return i;
        }

        if(sv_cpmver == 0x31)
        {
                b[0] = scbf; b[1] = 0;

                return bdos_a(0x31, b) + 1;
        }

        return 0;
}
```


**Directorios**

En CP/M no hay directorios (mal vamos).
Sin embargo, hay algo llamado áreas de usuario, que se le parece (algo, es algo).
Ahora bien, las áreas de usuario, nunca han sido muy utilizadas por el propio CP/M. Más bien, son toleradas.
En CP/M 2, el uso de las áreas de usuario, plantea bastantes problemas, mientras que en CP/M 3, aun cuando su uso es algo mayor, igualmente deja bastante que desear.
Éste es un tema, que nunca ha sido explotado convenientemente por CP/M.
De hecho, sólo podemos trabajar sobre ficheros que estén en el área de usuario actual (generalmente, la 0).
Tampoco podemos hacer referencia, de forma estándar, al área de usuario de un fichero, ni a nivel de comando en el CCP, ni a nivel de funciones del BDOS.
Esto, lógicamente, plantea muchos problemas, a

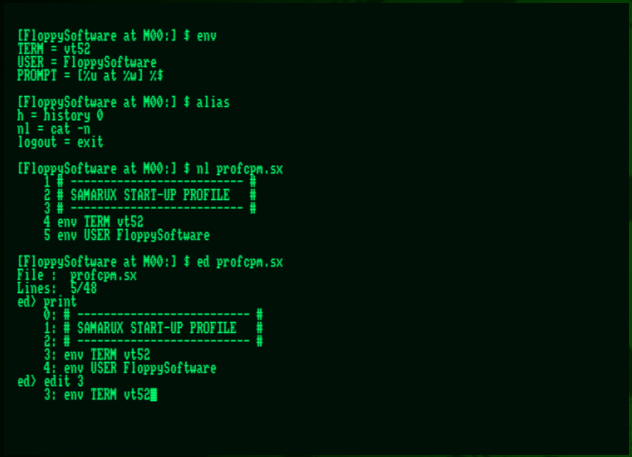
la hora de escribir un shell, que se precie mínimamente de serlo.
De todas formas, como menos es nada, el plantemiento que se ha hecho en SamaruX, ha sido el de hacer equivalentes las áreas de usuario a los directorios (nada novedoso, por otra parte).
Por ejemplo, para referenciar al área de usuario 5 de la unidad de disco B:, lo haríamos así:

```
B5:
```

E incluyéndolo en un nombre de archivo:

```
B5:NOTAS.TXT
```

Además de estos inconvenientes, tenemos el hecho de que, por lo general, las bibliotecas de funciones de los compiladores de C para CP/M, no permiten el uso comentado.
Aqui es cuando cobra todavía más fuerza, el uso de MESCC, ya que podemos modificar sus




bibliotecas estándar, a placer.
Queda abierta la posibilidad de dar nombre a dichos directorios, en una futura versión de SamaruX (similar al método utilizado por ZCPR, pero más elaborado, lo que plantea algún que otro problema).
Por lo tanto, se plantean varios retos:
· Equivalencia del binomio unidad de disco / área de usuario al directorio.
· Referencia al área de usuario en los nombres de fichero.
· Utilización del área de usuario en las operaciones con ficheros (FCB, lectura, escritura, creación, renombrado, etc.).
· Modificación de las bibliotecas del compilador (fopen, fread, fgetc, fputc, remove, etc.).
Los dos últimos puntos, pasan por extender el FCB (File Control Block) de CP/M, para que haga referencia también al nº de usuario, e implementar una función en C, que permita realizar operaciones sobre ficheros, en números de usuario distintos al actual.
Esto último, implica utilizar dicho FCB extendido (FCX), en base a este código:

```
/*          unsigned char bdos_fcx_a(unsigned int bc,
unsigned int de)

            Calls BDOS with FCX.
            Returns A.
*/
bdos_fcx_a(fun, fcx)
int fun; BYTE *fcx;
{
            int val; int old_user;

            if(*fcx)
            {
                        if((old_user = bdos_a(BF_USER,
0xFFFF)) != *fcx - 1)
                        {
                                    bdos_a(BF_USER, *fcx -
1);         /* Set user to FCX user */
                                    val = bdos_a(fun, fcx + 1);  /*
Call BDOS function */
                                    bdos_a(BF_USER,
old_user);          /* Set old user */

                                    return val;
                        }
            }

            return bdos_a(fun, fcx + 1);
}
```

**Comandos del shell**
Actualmente (versión 1.03), SamaruX contiene 31 comandos internos:
· alias ascii batch builtin cat cd chmod clear
· cp cpm df dump echo ed env exit goto grep
· history if ls man mem more mv pwd read
· rm sort ver wc

Una vez implementado el grueso de SamaruX, escribir los comandos internos es, por lo general, una tarea relativamente sencilla.
Esto es así, porque SamaruX pone a disposición de los comandos, una serie de funciones de apoyo generales (manejo de errores, arrays, lectura de ficheros, acceso a variables de entorno, etc.).
Veamos por ejemplo, la implementación del comando clear, que borra la pantalla:

```
ClearMain(argc, argv)
int argc, argv[]; /* char *argv[] */
{
            char *term;
            int k;

            /* Check arguments */

            if(argc != 1)
                        return ErrorPars();

            /* Clear */

            if((term = EnvGet("TERM")) != NULL)
            {
```

```
                if(!strcmp(term, "vt52"))
                {

putch(27);putch('H');putch(27);putch('E');
                }
                else if(!strcmp(term, "vt100") ||
!strcmp(term, "ansi"))
                {
                        putch(27);putstr("[2J");
                        putch(27);putstr("[1;1H");
                }
                else if(!strcmp(term, "ascii"))
                {
                        k = TermRows();

                        while(k--)
                                putchar('\n');
                }
                else
                        return Error("Unknown terminal
type");

                return 0;
        }

        /* Failure */

        return Error("Can't find $TERM");
}
```

**El futuro de SamaruX**

La verdad, es que programar SamaruX ha sido (y es) una gozada.

Una vez construido el "kernel", el resto ha sido bastante más fácil (y ameno) de lo que pensaba.

Evidentemente, tiene carencias, unas más fáciles de resolver que otras.

Actualmente, veo cuatro caminos a seguir:

· Adición de más comandos.

· Acceso opcional a los directorios mediante un nombre.

· Ejecución de comandos SamaruX externos.

· Mayor integración en CP/M.

En cuando a comandos internos, hay que llegar a un compromiso, entre memoria disponible y funcionalidad. Simplemente, no se puede añadir todos los comandos que uno quiera. La TPA de CP/M tiene un límite.

Ahí es donde cobra fuerza, la posibilidad de ejecutar comandos externos de SamaruX.

En este sentido, las ideas van madurando, y creo que es factible su utilización, a partir de:

· Ejecutables SamaruX como ficheros PRL (relocalizables) de CP/M.

· Asignación de memoria dinámica para su ejecución.

· Acceso a las funciones de SamaruX y de la biblioteca de funciones de MESCC, al estilo overlay.

En cuanto a la posiblidad de referenciar los directorios mediante un nombre, el camino a seguir se presenta similar al enfoque de ZCPR, pero algo más elaborado (seguramente, mediante la utilización de los comandos mount y umount).

SamaruX ha sido publicado bajo licencia GPL, por lo que su código fuente está disponible para todo aquél que lo quiera.

**Resumiendo**

He intentado ofreceros una visión global de SamaruX, resaltando los puntos más importantes de su implementación, ilustrando el proceso con algo de código.

Evidentemente, no se ha podido profundizar, pues el tema tiene chicha en cantidad.

En cualquier caso, espero que os haya resultado ameno.

Desde aquí, os animo a entrar en el mundo de la programación para CP/M.

Aunque a primera vista pudiera parecer algo con pocos incentivos, os aseguro (yo, que lo cato día sí y día también), que no os sentiréis defraudados.

SamaruX, como todo lo que programo para CP/M, está dedicado a la memoria de Gary Kildall, creador (entre otras cosas) de dicho sistema operativo.


**FloppySoftware**
**floppysoftware@gmail.com**
**www.floppysoftware.vacau.com**
**cpm-connections.blogspot.com**

# IT CAME FROM THE DESERT AHORA EN ESPAÑOL

***Para los que no conozcan este titulo, les presentamos este juego que salio para commodore amiga en 1989 de la mano de cinemaware y que fue portado a varios sistemas.***
***Tenemos con nosotros a Diego de Amigamers, que se nos hablará sobre su ultimo trabajo, y no es un vídeo, en este caso es una tarea que ha consistido en traducir al castellano este fabuloso juego. Si, como lo estáis leyendo, Diego ha trabajado en una traducción para que todos podamos disfrutar de este juego. Ahora vamos ha hacerle unas preguntas para saber más sobre este tema...***

**Hola diego, bienvenido a Retrowiki magazine. Antes de hablar de tu trabajo con este juego, háblanos un poco de el para los que no lo conozcan, ¿cual es su genero? ¿sus creadores? ¿su argumento inicial?**

Bueno, It Came from The Desert es una aventura al estílo cásico de Cinemaware. Esta empresa acuñó el término, película interactiva, y echando la vista atrás, hay que admitir que lo hicieron muy bien, teniendo en cuenta las limitaciones del hardware de su época. Estamos hablando de finales de los 80. El desarrollo de este juego me recuerda mucho a las novelas de "liege tu propia aventura" que leía de adolescente, solo que este juego intercala algunas fases arcade para darle más chispa. En cuanto al argumento, está inspirado en una película de 1954 llamada Them!. Este título nos traslada a los años 50, a un pueblo en mitad del desierto de Nevada, llamado Lizard Breath. La reciente caída de un meteorito ha desencadenado una serie de sucesos.
Este juego ofrece, no solo hormigas mutantes, sino triángulos amorosos, timadores, especuladores de terrenos y mucho más. Para mi gusto, un gran juego.


**¿Tuviste este juego en tus tiempos de amiga? ¿Que te pareció?**
Si, aunque debo admitir que mi copia era pirata. En aquella época le compraba a un grupo de Oviedo llamados TOAD (The Oviesoft Amiga Division). Me impactó mucho el apartado gráfico. Cinemaware siempre mimaba sus productos en este punto. Sin embargo, mi nivel de inglés no era suficiente para hacerme con el control del juego, y me dedicaba a vagabundear y explorar sin saber muy bien lo que hacía. En aquella época nunca me lo acabé.

**¿Como se te ocurrió traducir este juego?**

Este juego me volvió a la memoria con motivo del vídeo que hice sobre la historia de Cinemaware. Lo jugué un poco para sacar algunos gameplays y ya me fuí picando hasta que lo acabé. La verdad es

que me pareció un juego con una atmósfera increíble y me dio algo de pena no haberlo podido disfrutar en su día por culpa de las limitaciones del idioma. Hace un mes aproximadamente me plantee traducirlo y me puse a ello echándole algunos ratos libres.

**¿Como fue el trabajo? ¿que herramientas se usan para traducir un juego de amiga?**

La mayoría del trabajo lo realicé con una herramienta de Amiga llamada File Master. Es un editor Hexadecimal, el cual me permitía acceder a las cadenas de texto del juego para sustituirlas por su traducción. Esto plantea varios problemas el primero de ellos, es que la longitud de una frase no puede superar ni en un solo carácter a la del original. Esto no debería ser un problema, pero si lo es. Por ejemplo, la opción "Sleep" (cinco caracteres) se debería traducir como "dormir", pero eso es imposible por la longitud de la palabra, así que tuve que utilizar "soñar".

Otro punto que me frenó mucho el trabajo fueron algunos personajes, como por ejemplo Geez, que es un "redneck" que se come las palabras. Imagina que eres un Americano que estudia Castellano y luego te ponen un texto escrito tal como pronuncia un andaluz "cerrao". Pues te vuelves loco, claro. También había uno con acento frances y un tartamudo.

**¿Que fue lo mas difícil?**

Lo más difícil fue el tema de las tildes, la Ñ y los

símbolos de puntuación en castellano, como la interrogación o el signo de admiración abierto. En este menester fue vital la ayuda de Fernando Esteban García, el cual modificó la fuente de letra utilizada por el propio juego y empleando caracteres que no se utilizan, como la arroba o los corchetes, redibujó todos los caracteres que necesitaba. De esta forma, el juego dispone de tildes, y todo lo que nuestro idioma necesita. Sin embargo, no se podían escribir directamente así que cuando llegaba a una palabra con (por ejemplo) una "i" minúscula debía escribir "3E" en su lugar. Una vez en el juego, si aparece la vocal con tilde.

**¿Cuanto tiempo has tardado?, mas o menos**

Pues me ha llevado unas tres semanas, claro que tampoco le he dedicado muchas horas. Lo iba haciendo a ratos, cuando el resto de obligaciones me lo permitían. El juego está totalmente traducido y lo he testeado bastante, aunque puede que haya alguna errata por ahí suelta.
Lo único que me falta por traducir es la voz en off de la secuencia de introducción, pero para la versión definitiva intentaré que esté también en Español. Creo que es el elemento que le falta para tener una traducción 100%. Por el momento, se puede ver un longplay del juego traducido, aún en fase de testeo, (por lo que habrá algunos fallos) en mi canal de Youtube.

**¿Sabe algo de esto la empresa Cinemaware? ¿Que les pareció?**

Me puse en contacto con ellos por Twitter y les pareció una idea estupenda. Hay que señalar que con una gente muy accesible. Ya se pusieron en contacto conmigo cuando hice los dos vídeos de Cinemaware y me pidieron que los tradujera al

inglés, pero la verdad es que no estoy muy por la labor... Si acaso subtítulos, pero nada más. De todas formas, antes de colgar la versión definitiva en la red, les pediré permiso expreso, para evitar sorpresas.

**¿Vas a publicar el juego traducido? ¿Donde se puede conseguir?**

Publicarlo no. Este trabajo no tiene ánimo de lucro. Una vez me autorice Cinemaware, lo colgaré en internet para que todo aquel que lo desee pueda disfrutarlo, ya sea en emulador o en un Amiga real. Lo que si he pensado (y lo tenía claro desde el principio) es en crear una edición

física del juego. Ya sabes, caja de cartón, con sus textos también traducidos, sus diskets, manual en castellano, hoja de referencia, mapa y alguna que otra sorpresilla. Esta edición física será en un número muy limitado y será para regalar o sortear, pero no pienso ganar nada con ella.

**¿Hay proyecto de traducir alguno más? si es así háblanos de ellos...**


Pues ya metido en el tema me gustaría hincarle el diente a Ant-heads, la secuela de It Came. También hay otros muchos que me gustaría ver traducidos, como por ejemplo, Infetation de Psygnosis, pero por el momento mi objetivo es acabar con este, luego ya se verá.

**Cambiando de tema un poco ¿Que nos deparan tus documentales sobre amiga? desvelanos algo sobre programas futuros**

Bueno, recientemente el disco duro donde almacenaba todo el material de Amigamers, ha pasado a mejor vida. Esto me va a frenar en la producción de vídeos y me ha echado a perder dos que ya estaban casi finalizados. Sin embargo no me rindo y he comenzado a



editar de nuevo. Próximamente tendremos un vídeo especial sobre los mejores juegos de Amiga y también se está cociendo otro sobre mi juego favorito de Microprose...

**Si quieres añadir algo mas...**

Pues solo agradeceros el trabajo tan bueno que hacéis con vuestra revista y agradeceros también el que contarais conmigo nuevamente para formar parte de ella.

**Bueno pues esto es todo, gracias por atendernos y como siempre, es un placer tenerte en RetroWiki Magazine**

# El Inves Spectrum+


a fondo

*El Inves Spectrum+, o como se le suele llamar, simplemente, "el Inves", ha sido durante mucho tiempo, y casi desde su lanzamiento al mercado español a finales de 1986, el patito feo de los usuarios del ZX Spectrum. Aquellos niños a los que sus padres les compraban este ordenador se sentían en ocasiones frustrados porque los juegos que probaban en su flamante Inves no funcionaban igual que en casa de sus amigos, o incluso no funcionaban en absoluto. Daba igual que "su" Spectrum incorporara un conector de joystick, y no necesitaran comprar aparte el consabido interface Kempston o Sinclair. Daba igual que la máquina fuese un poco más amigable para los recién llegados a esto de la microinformática, al presentar todos los mensajes de error en perfecto español. Daba igual que tuviera no uno, sino dos lugares reservados para letra más patria del idioma español (uno para la Ñ mayúscula, y otro para la ñ minúscula), y que te permitiera imprimir en pantalla frases como "FELIZ AÑO NUEVO" en el televisor durante una celebración de Año Nuevo con la familia, sin que la falta de Ñ hiciera sacar los colores a ese tío tuyo enganchado muy a su pesar a la crema Hemoal. Daba igual que Microhobby, la revista de referencia en el panorama del ZX Spectrum en España, se deshiciera en elogios sobre ella.*

Todo eso daba igual. El Inves estaba gafado. Historias (reales o no tanto) sobre sus incompatibilidades, fallos o incluso supuestas formas de averiarlo permanentemente con solo teclear unos comandos, planearon sobre él casi desde el mismo día en que se puso a la venta. Los que pudieron, lo vendieron para comprarse un flamante Spectrum 128K y seguir así "con lo último" de la máquina de Sinclair. Para los que no lo hicieron, quizás fue su primer y último microordenador, antes de dar el paso a PC's, Amigas o Ataris, o pasarse "al otro bando" de Amstrad, Commodore o MSX.

Sea como fuere, la historia comercial del Inves Spectrum+ fue muy corta, truncada por la popularización de los modelos de 128K, claramente superiores a lo que ya había y que de hecho estaban ahí antes que el Inves, y por sus propias fallas, sobre las que los responsables no pudieron, o no quisieron pronunciarse. El último estertor de la que fuera distribuidora en exclusiva de Sinclair Research en forma de micro de 8 bits llegó al mercado tarde y con más fallos que soluciones.

Con esta carta de presentación, a pesar de lo que pudiera parecer, no pretendemos echar más leña al fuego. Todo lo contrario: si algo le faltó al Inves Spectrum+ fue tiempo para que los programadores y diseñadores hardware se sentaran frente a él y pensaran: "a ver, qué pasa aquí y qué podemos hacer".

Ha pasado tiempo, mucho tiempo: casi 28 años después de su lanzamiento. Aquí los presentes autores nos hemos sentado (casi literalmente) delante de la máquina, la hemos examinado por dentro y por fuera, y por fin sabemos qué pasa con ella: por qué funciona como funciona, qué fallos reales tiene, qué características, nunca publicadas por la prensa de la época, tiene, y cómo podemos aprovecharla a tope. El resultado es un trabajo inédito, en el que se muestra al público como unos ingenieros sin nombre, allá por 1986, diseñaron una máquina con innovaciones que de haber funcionado como debieran, habrían hecho sonrojar a Amstrad cuando diseñó los modelos +2 y +3. Innovaciones algunas que no volverían a verse hasta que en la ex-URSS diseñaron el Pentagon.

***Señoras y señores: el Inves Spectrum+, como nadie nunca lo mostró antes.***

(Los autores desean expresar su agradecimiento al staff de la revista RetroWiki Magazine, por ofrecerse a publicar este artículo, y por ser de las pocas revistas del panorama retro capaz de alojar un artículo alejado de los temas habituales sobre juegos nuevos/juegos viejos/consolas nuevas/consolas viejas)

## Historia

Probablemente, la primera noticia que tuviera el público en general de la existencia del Inves sea una columna en la página 4 de la sección Micropanorama, en la revista Microhobby, en su número 101 (publicado a principios de Noviembre de 1986). En aquella reseña, lo único que trascendió fue el rediseño del sistema operativo, con todos los mensajes en español: no se hizo mención al port de joystick. Por supuesto, se aseguraba y se juraba que la máquina "debía" ser compatible con todos los títulos de software del momento. También se decía que la máquina sería fabricada integramente en España, cosa que no sé si llegó a ser cierta, o se hizo lo más normal, que es hacer outsourcing a alguna empresa oriental como ya se había hecho con el Spectrum 48K en alguna de sus issues (fabricada por Samsung), o más tarde con los +2A y +3 (fabricados en Taiwan).


Habría que esperar alrededor de un mes para que en el número 106, y de nuevo en la sección de Micropanorama (página 7), se diera, en una escueta nota de prensa, información sobre su disponibilidad en grandes almacenes. Entonces fue cuando nos enteramos de que la nueva máquina además incorporaba port de joystick y un rediseño completo de la placa base.

Quien suscribe no estaba en aquella época informado de los cotilleos internos que no aparecían en las revistas de Informática, así que no puedo contestar a por qué surge el Inves, cuando desde hacía algún tiempo ya existía el ZX Spectrum 128K, muy superior en todos los sentidos al Inves, y con menos incompatibilidades, y para colmo desarrollado conjuntamente con, presuntamente, los mismos ingenieros que después diseñaron el Inves Spectrum+. Años más tarde se han escuchado cosas como que "Investrónica tenía el "derecho moral" de sacar un nuevo modelo de Spectrum", etc. Parece ser que lo más probable que pasara es que por aquella época ya se había consumado, o estaba a punto de hacerse, la compra de Sinclair Research por parte de Amstrad, lo cual suponía dejar a Investrónica fuera del negocio de la distribución de los nuevos modelos en España, ya que la distribuidora en exclusiva para España de Amstrad era Indescomp. Investrónica no tenía más remedio que sacarse un ordenador "patrio" de la manga para sobrevivir. Claro que todo esto no son más que especulaciones, y no es mi campo de experiencia, así que prosigamos con los detalles más…. técnicos.

## Características publicadas

En el número 108 de Microhobby, en su página 24 y siguientes, Primitivo de Francisco analizó a fondo las características del recién salido microordenador. Entre ellas se pueden destacar las siguientes (los comentarios no son de Primitivo, sino míos):

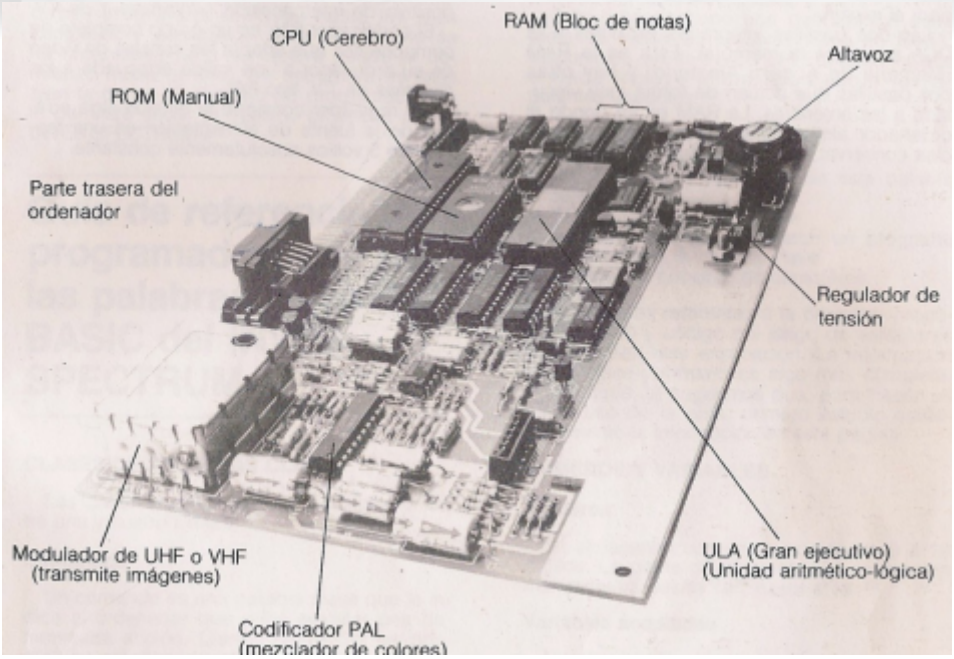
**Su precio:** 19.900 pesetas; lo curioso es que el propio Primitivo lo destaca frente a las 52.000 pesetas que costó originalmente el Spectrum 48K. Sin embargo, en el mismo número de la revista, en la publicidad de la empresa Delta, se oferta el Spectrum 128K por 26.500 pesetas, y el Spectrum Plus de 64K (¿?) por 22.900 pesetas. El Inves es equivalente a un Spectrum Plus más un interface de joystick. Un rápido cálculo mirando los precios en otra página del mismo número, nos da un precio de 1.300 pesetas para una interface de joystick. Así, sería más adecuado comparar esas 19.900 pesetas del Inves con las 22.900 + 1.300 = 24.200 pesetas del Spectrum Plus + interface joystick.

**Teclado españolizado**, con teclas Ñ mayúscula y minúscula, cedilla, U con diéresis, acento grave y signos de apertura de interrogación y admiración.

**Entrada canon DB-9** para conexión de joystick.

**Procesador Zilog genuino Z80-A**. Lo de "genuino" es un adjetivo que Primitivo usó años antes de que Intel lo incrustrara a fuego en sus procesadores para poder leerlo con la instrucción CPUID y así

diferenciarlos de su más directo competidor AMD. Se supone que al ser de Zilog debería ser "mejor". Sin embargo, las malas lenguas (en WOS) dicen, por irónico que parezca, que la de Zilog no es la mejor implementación del Z80, sino la de NEC. Como dato, el test que prueba el comportamiento de los flags de forma exhaustiva, incluyendo los misteriosos bits 3 y 5, comparan sus resultados no con un Zilog, sino con un NEC, también "genuino".




**Sistema operativo** en EPROM de 16KB.

**48KB de RAM** en solamente dos chips. El autor del artículo indica que el refresco de estas memorias se realiza "con sólo 4 circuitos integrados en lugar del chip de 40 pines que se montó en los últimos modelos de Spectrum" exclusivamente con este cometido. La verdad es que sólo hay 1 circuito integrado responsable de su refresco.

**Reloj maestro** de 17,7345 MHz, frente a los 14 MHz del reloj maestro del Spectrum 48K. La misma frecuencia maestra que el Spectrum 128K, de quien cogió algunas ideas.

**Codificador de color** MC1377: más calidad que el Lm1889.

**Cambios en los slots de expansión:** las señales Y,U y V desaparecen, así como las tensiones de 12V y -5V.

**El circuito convertidor de tensión** es más sofisticado, y desaparece el transformador de ferrita, sustituyéndose por dos bobinas con aspecto de resistencia de medio watio y un chip con comparadores analógicos. Sin embargo, hubo al menos una revisión anterior de la placa base en la que seguía estando el transformador de ferrita y no estaba el chip de comparadores. Esa misma revisión anterior usa un tornillo con tuerca para unir el regulador 7805 al disipador en lugar del remache que se usó en la versión más popular del equipo.
La publicidad de la época, por supuesto, destacó hasta la saciedad la compatibilidad de todo el software existente con la máquina, con vocación de continuidad, para que los usuarios puedan seguir aprovechando todo el software ya existente.

## Incompatibilidades con software de la época

Durante la primera mitad del año 1987, depués de la campaña de Navidad en la que se ha presentado el Inves, comienzan las consultas sobre incompatibilidades. Microhobby admite que no sabe cómo resolverlas.

### COMPATIBLE "A MEDIAS"

Tengo, desde Navidad, un Inves Spectrum de 48 K, y me gustaría poder jugar algún día con él. El caso es que, según dicen, resulta «totalmente compatible» con todo el software y el hardware del antiguo Spectrum de 48 K; pero, por lo que yo he podido comprobar, no es cierto. Hay programas como el «Comando», «Mugsy», «Top Gun», etc., que o bien no corren, o se bloquean en algún punto.

El ordenador ya lo he cambiado una vez donde lo compré, y sin ningún resultado, y conozco a varios usuarios que se encuentran con el mismo problema. ¿Qué podemos hacer? Los juegos no resultan lo suficientemente baratoss, para que los compremos por el simple placer de leer las instrucciones.

**Sebastiá NOGUERO**-Lérida

■ *El Inves Spectrum es un compatible «a medias». Hay interfaces con los que no funciona (por ejemplo, con el Disciple) y hay programas con los que se «cuelga» o funciona defectuosamente. Hay, incluso, una posición de memoria a la que, si se hace un RANDOMIZE USR..., el ordenador ¡se avería! (es el primer caso que conocemos de un ordenador que se pueda averiar por software). Una de las posibles causas de incompatibilidad podría ser que el programa en cuestión chequeara la ROM (hay algunos que lo hacen como protección); aunque sospechamos que hay otras razones que justifican estos fallos de compatibilidad.*

*Actualmente, no tenemos respuesta para estos problemas; aunque estamos investigando en ello y esperamos poder publicar pronto algunas soluciones para los sufridos usuarios de los Inves Spectrum.*

(Sección "Consultorio" del número 136)

Quizás precisamente como respuesta colectiva a los usuarios afectados, Oscar García Reyes escribe un artículo en el número 139 de Microhobby (páginas 24 y siguientes) resultado de haber analizado el equipo, y concluyendo que la incompatibilidad de algunos juegos se debe a alguna de estas tres causas:

•Juegos que presentan sprites parpadeantes, o movimiento no fluido, o incluso sprites que no llegan a verse del todo. Como acertadamente se dice, esto es causado por la (gran) diferencia de tiempo que hay entre el momento en que ocurre el pulso de interrupción del retrazo vertical y el momento en que se comienza a pintar el "paper" en la pantalla. En el Inves ese tiempo es mucho más pequeño, pero el software no lo sabe, y algunos programas realizan el redibujado de los sprites cuando el haz de rayos catódicos ya ha pasado por esa parte de la pantalla, haciendo que en algunos casos, el sprite se vea con un ostensible parpadeo, o directamente no se vea. La otra razón que se arguye, que la CPU trabaja a 3.54MHz en lugar de a 3.5MHz en la práctica no es problema, siendo la diferencia de velocidad practicamente despreciable para la inmensa mayoría del software.

•Juegos que usan la ROM de alguna forma: aquí se incluyen los juegos que usan rutinas de protección con encriptación de datos usando a la ROM como clave criptográfica, o juegos que sencillamente usan alguna rutina de la ROM (o parte de ella). La ROM del Inves cambia en bastantes cosas respecto de la de Sinclair, más que nada para poder dar cabida a un teclado con caracteres en español e impresión en español. Las rutinas que más cambian son las relativas al teclado, pero hay algún que otro cambio más. El artículo documenta dichos cambios.

•Juegos que se comportan de forma extraña con el sonido u otros periféricos. Hablaremos un poco sobre ello:
Oscar comienza detallando los problemas que hay con el sonido, pero en la explicación hay algunas inconsistencias. Realmente esto es lo que pasa: en el ZX Spectrum se usa el puerto $FE en escritura para generar el sonido, y en concreto, dos bits dentro de éste: el bit 3 que se emplea para activar MIC, y el bit 4 que activa MIC y el altavoz. Si se usa solamente el bit 3, la señal de audio producida por la ULA (la original de Ferranti) no tiene suficiente voltaje para polarizar la base del transistor TR7 que se usa como amplificador de audio para el altavoz y en consecuencia, el sonido no se oye, pero sí por MIC, que no tiene esa limitación. Si se usa el bit 4, sólo o junto con el bit 3, el voltaje de la señal producida excede el mínimo necesario para polarizar a TR7 y en consecuencia, el sonido se oye tanto por MIC como por el altavoz. De hecho, el mayor nivel de sonido se obtiene activando o desactivando a la vez, ambos bits. En cuanto a EAR, está conectada al mismo pin de sonido que alimenta a MIC, pero con una impedancia más baja, lo que resulta en un nivel de sonido más alto. Oscar argumenta que la creciente costumbre de los usuarios de prescindir del altavoz y usar la señal de MIC o EAR para escuchar el sonido amplificado obligó a los desarrolladores de software a usar los dos bits del puerto $FE en lugar de uno solo. Lo cierto es que para escuchar por EAR o MIC la señal de audio, basta con usar uno cualquiera de los bits, pero si se quiere escuchar un sonido alto por el altavoz interno, es mejor alzar o bajar ambos bits a la vez. Así que el razonamiento del autor iba bien encaminado, ¡pero al revés!
Decíamos que en la ULA de Ferranti, usar ambos bits da una señal de audio más alta que usar cualquiera de los dos por separado. Esto funciona en esa ULA porque la misma puede generar y procesar señales analógicas, y en este caso, lo que hace simplemente es sumar la información de ambos bits en una suerte de conversor DAC de 2 bits no lineal. El equivalente a la ULA de Ferranti en el Inves, el TAHC10 de Texas Instruments, no es un chip de señal mixta (que puede usar señales analógicas y digitales) sino 100% digital, así que no se pueden mezclar los datos de dos bits y esperar una señal más alta. En lugar de ello, lo que sale por el pin de sonido del TAHC es el resultado de la operación XOR de los valores de los bits 3 y 4. Lo que significa esto es que si ambos se ponen a 1 ó a 0 a la vez, el resultado es 0, y eso es lo que pasa con el Inves en los juegos que alzan y bajan a la vez ambos bits: que en lugar del efecto esperado de un sonido más alto, lo que se obtiene es un silencio absoluto. En la tabla adjunta se muestran los niveles de tensión en el pin 28 de la ULA en un Spectrum de Sinclair, issue 2 e issue 3, según los valores que se escriban en los bits 3 y 4, y para comparación, los valores de voltaje en un Inves (pin 14 del TAHC10)

| Bit 4 (SPK) | Bit 3 (MIC) | Voltaje issue 2 | Voltaje issue 3 | Voltaje Inves |
|---|---|---|---|---|
| 0 | 0 | 0,39 V | 0,34 V | 0 V |
| 0 | 1 | 0,73 V | 0,66 V | 4,80 V |
| 1 | 0 | 3,63 V | 3,56 V | 4,80 V |
| 1 | 1 | 3,79 V | 3,70 V | 0 V |

http://www.worldofspectrum.org/faq/reference/48kreference.htm#PortFE)

INFORME

INVES SPECTRUM +
¿TOTALMENTE COMPATIBLE?

Otra fuente de incompatibilidad relacionada con los periféricos estriba en que la implementación que el Inves hace del puerto de joystick Kempston hace que éste se active con tan sólo que la dirección del puerto al que se acceda tenga su bit 5 a 0. Esto supone una relajación de las condiciones respecto a cómo se debería haber implementado dicho puerto: decodificando al menos A5, A6 y A7 de forma que si estos tres bits valen 0, y A0 vale 1, se habilita el puerto Kempston, dando como dirección de puerto la $1F. Decodificando únicamente A5 también se habilita el puerto para la dirección $1F por supuesto, pero también lo habilita para muchas otras direcciones de puerto que pueden entrar en conflicto con la interface Kempston integrada. En concreto, podría dar problemas con la interfaz de joystick que incorpora la interface Mikro-Plus del juego Shadow of the Unicorn.

La otra fuente de incompatibilidad es más conocida (al menos ahora) y tiene que ver con el comportamiento del bus flotante, habitualmente implementado en el puerto $xxFF (donde xx es cualquier valor aunque algunos programadores como Joffa Smith suelen usar aquí el valor hexadecimal 40, para que la dirección resultante, $40FF sea un puerto con contienda). Los juegos que no funcionan en el Inves por esta razón tampoco lo harán en el +2A y +3, los cuales ya estaban presentes cuando el Inves estaba aún vendiéndose.

Lo que el artículo no cuenta es que hay al menos otra causa más de incompatibilidad con juegos, y ésta sí que obedece a un fallo de diseño del Inves Spectrum. Lo documentaremos en la sección de Interrupciones ya que es a éstas a las que afecta.

## Leyendas urbanas

Microhobby a veces sirvió también como fuente de “desinformación”. La siguiente respuesta a una consulta en el número 156 de la revista, en su página 32 dedicada a las “Microconsultas”, decía textualmente lo siguiente (captura de pantalla tomada del proyecto microhobby.org)

*Aprovechamos para avisar a los usuarios del Inves, que nos ha llegado el rumor de que haciendo:*
*BORDER 5*
*RANDOMIZE USR 4665*
*se avería el ordenador. No hemos tenido aún ocasión de comprobarlo (de funcionar, supondría cargarse un ordenador) pero esperamos poderlo verificar en un futuro. En una ocasión dijimos que era imposible averiar un ordenador desde el teclado; bien, ésta sería la excepción que confirma la regla.*

En un video que publiqué en la web de ZX Projects demostré que no había nada de malo en ese RANDOMIZE USR. De hecho no tenía sentido que una llamada a la ROM (aunque fuera en medio de una instrucción) pudiera tener ese efecto tan devastador. Por si acaso, descargué de la web de Philip Kendall el binario de la ROM del Inves, y lo probé con el emulador Spectaculator, haciendo la traza de la ejecución en ROM de la rutina llamada. El valor 4665 cae en medio de una instrucción multibyte, lo que significa que las dos o tres primeras instrucciones son decodificadas incorrectamente, pero tras ello, el resto es ejecución de código estándar de la ROM, que seguramente se ejecutará en muchas más ocasiones durante el funcionamiento de la máquina, estando en BASIC.

Ni que decir tiene que en la máquina real pasó exactamente lo mismo que en el emulador. No hubo daños a la máquina, ni temporales ni permanentes. Al hacer el RANDOMIZE USR 4665, la máquina vuelve inmediatamente al BASIC con el mensaje de inicio “<Sistema preparado>” con el borde celeste. Al pulsar ENTER da el mensaje “C NO EXISTE EN BASIC, 0:1”. Al pulsar ENTER de nuevo, la zona de

edición (las líneas inferiores de pantalla donde se escriben los comandos BASIC) se llenan de basura, con sentencias, números, y demás caracteres ininteligibles. En este punto el intérprete de BASIC se comporta de forma inestable, y la única forma de salir es con un reset.
Si se realiza esta sencilla prueba con la ROM original de Sinclair, el resultado es idéntico, sólo que cambia el texto de los mensajes que aparecen.
En la misma respuesta a la consulta, de hecho, en los párrafos precedentes, Microhobby nos obsequiaba con la siguiente información:

van algunos. Sin embargo, la causa más grave de incompatibilidad del Inves radica en el manejo de puertos. De momento, hemos detectado que las salidas de buses al exterior se realizan a través de unos «buffers» que protejen el micro de sobrecargas; sin embargo, es necesario utilizar ciertos puertos para abrir esos buffers y ésto crea ciertos problemas de incompatibilidad. Si de-

Aunque aún no hemos llegado a la parte en la que describimos el hardware del Inves, ya adelanto que no hay tales búferes. Matizo: en el Inves hay dos chips que hacen de bufferes tipo 74LS244 (tres si contamos como buffer a un buffer inversor tipo 74LS240 que es lo que implementa la interface Kempston). Y aunque es cierto que dos de estos tres bufferes tienen que ver con la entrada/salida, es falso que alguno de ellos (o algún otro chip) proteja al Z80 de las salidas del exterior del bus de expansión. Es más: en la placa del Inves, el Z80 está prácticamente pegado al bus de expansión, y todas las señales del Z80 presentes en el bus trasero, vienen directamente del procesador. No existen por supuesto tampoco esos “ciertos puertos” para abrir esos bufferes. Si existieron en algún prototipo al que Microhobby tuviera acceso lo desconozco. En la máquina que fue dada a conocer a los usuarios, objeto de la consulta, no había, ni hay, nada de esto. En la sección de descripción técnica de la máquina podremos ver qué tiene y qué no tiene.

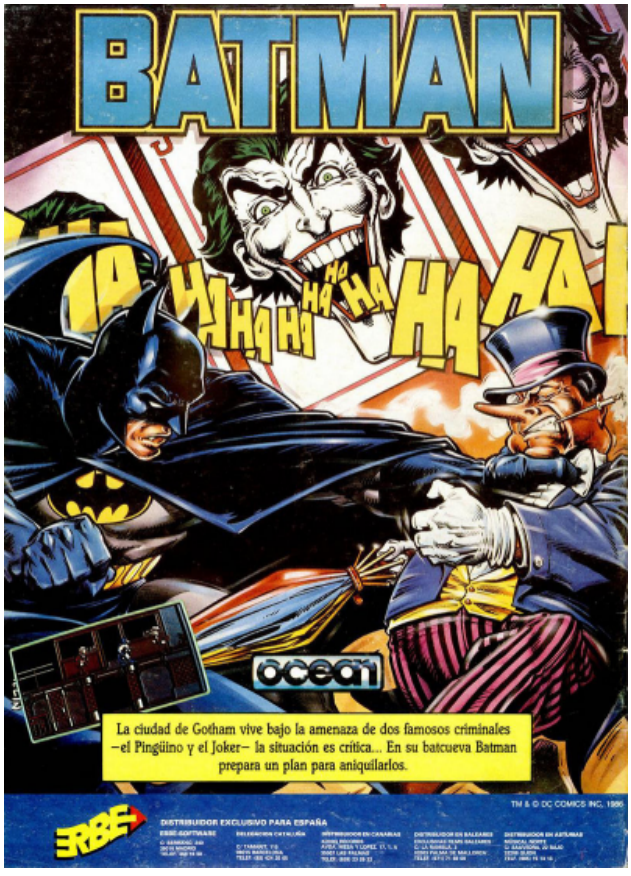
## ZXSpectr y los primeros pinitos en la investigación sobre las interioridades del Inves

(*por César Hernández Bañó*).

Antes de conseguir el Inves, en mi casa hubieron unos cuantos ordenadores más. Primero fue el ZX-81, luego vino un Sinclair Spectrum 48k, y posteriormente un Sinclair QL. A mediados de 1988, concretamente a final de curso, mi padre me dijo a mi y uno de mis hermanos, que por las buenas notas obtenidas, nos compraría un ordenador a cada uno. Yo tenía entonces 10 años, mi hermano 12, y mi otro hermano 16 (y él usaba el QL). Ya había planeado comprar un PC, que vendría unos meses mas tarde.
Así que todo ilusionados, fuimos a unos "grandes almacenes" muy famosos, a comprar dos ordenadores.
Al llegar allí sólo habían dos tipos de Spectrum, un flamante +3 con su unidad de disco, y luego al lado un Inves Spectrum +. Si no recuerdo mal, el +3 valía casi el doble que un Inves, y por tanto, cuando a mi padre le dijimos que cada uno queríamos un +3 nos dijo: ".... si queréis un +3, os compraré sólo 1 y deberéis compartirlo...". Mmmm pues no, ya que nos habíamos hecho a la idea tener un ordenador cada uno, pues nos tuvimos que conformar con el Inves.
Yo creo que en aquel momento lo vimos como un Spectrum + tal cual, no nos llamó mucho la atención el logo de Inves que había en la carcasa. Felizmente nos fuimos a casa cada uno con su ordenador, y lo bueno estaba por empezar...
Yo con aquél ordenador aprendí a programar Basic y código máquina. Antes con el Spectrum que había en casa sólo lo había usado para jugar y no había experimentado ningún interés por como funcionaba por dentro.

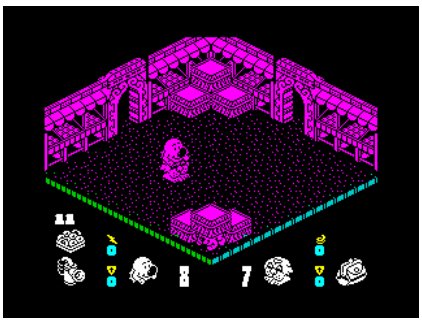

El primer descubrimiento extraño con el Inves fue con el pack de juegos "El Lingote”
(http://www.worldofspectrum.org/infoseekid.cgi?id=

0014484 )
Mi primo nos lo regaló en las navidades de 1988. Incluía 10 juegos, entre ellos, dos que jamás pude jugar en el Inves: Arkanoid y Short Circuit, pese a que cargaban perfectamente. La causa era debida a la inexistencia del "bus idle port" del Inves, de manera similar a como sucedía en los Spectrum +2A y +3. La lectura de puertos no asignados en Spectrum debería retornar el byte que leía la ULA... pero en Inves no retornaba eso, simplemente retornaba el valor 255. En esos dos juegos y en muchos otros era fatal... pues usaban la lectura de ese puerto para sincronizar el juego. En Arkanoid se podía ver la intro del juego, y cuando aparecía la pantalla de juego... la bola no se movía, ni la nave, ni nada... se quedaba colgado del todo. En Short Circuit el efecto era similar, aparecía el menú, seleccionabas la opción de iniciar el juego, y se quedaba ahí en el inicio del juego bloqueado, sin mas...
El siguiente descubrimiento extraño con el Inves afectaba al sonido.... había determinados juegos, sobre todo aquellos que hacían sonido a "dos canales" mediante el Speaker, en que sólo se oía un sólo canal, que correspondía al "ritmo" de la canción, mientras que el otro canal, el de la melodía, no se escuchaba. Esto sucedía por ejemplo en los juegos de Code Masters, como el ATV o el Dizzy, o también en el Batman (the Caped Crusader). Al principio yo pensaba que en esos juegos se escuchaba así la música, pues los había probado inicialmente con el Inves. Pero tenía dos amigos con Spectrum +2, y cuando cargabas aquellos juegos en modo 48K, se escuchaban completamente diferente....




Por aquél entonces mi hermano y yo ya teníamos una frase para cuando veíamos este tipo de "efectos": "otro defecto del Inves..." :P
El siguiente descubrimiento y el primer "susto" fue al toparnos, sin saberlo, con la ram oculta del Inves... Habían algunos juegos, creo que los que cargaban con protección Alkatraz 2 (esa carga turbo donde el tono guía se oye entrecortado) en que cuando dejabas de usarlos, al hacer reset (con el botón de reset del Inves), volvía correctamente al Basic, pero con el borde de color negro y sin sonido. El primero de ellos fue el Enduro Racer, que por cierto, en el propio juego no se oía ningún sonido.
Por mucho que le dieras al botón de reset el borde seguía ahí, negro, y el sonido, igual, mudo...


La solución era desconectar de la corriente y volver a conectar, y todo volvía a la normalidad. Aunque siempre te quedaba la duda de: "y si un día se queda así para siempre?"



Otro de los descubrimientos sobre el Inves, que no se apreciaba a simple vista, se producía con algunas pantallas de carga. Recuerdo la del Head Over Heels... en esa, cuando habían determinados colores uno al lado del otro, se producía un efecto de "raya vertical" de 8 píxeles de alto (y no hablo del atribute clash). Aparecían esas líneas molestas en algunas pantallas... Recientemente descubrí que ese efecto se denomina algo así como ULA Color Delay, y se produce no solo en el Inves sino también en muchos clones rusos. E incluso también en el Spectrum original, pero soló con el Flash, según información de Chris Smith.
Luego también había otro efecto en el Inves que se apreciaba en unos pocos juegos. Eran aquellos que hacían efectos en el borde; me viene a la cabeza ahora estos tres: Paperboy, Aqua Plane y Super

Wonder Boy. En un Spectrum normal se empieza a dibujar el borde superior cuando se lanza una interrupción. Pero en el Inves, cuando se lanza la

interrupción, se empieza a dibujar la pantalla, a la altura de donde empieza el borde izquierdo y derecho. Por tanto, los efectos en esos juegos, se ven desplazados hacia abajo considerablemente: el manillar del Paperboy del borde superior, aparece como una franja de colores sin sentido en el principio de uno de los borde laterales; el horizonte del Aqua Plane no continua en el borde, sino que provoca una especie de "cascada"; las letras del Wonder Boy al hacer la pausa, se ven comprimidas a los bordes laterales, haciendo mas difícil su lectura.

El siguiente descubrimiento sobre el Inves afectaba a la gestión de interrupciones. Recuerdo que tenía información de como activar las interrupciones IM2, y se comentaba que la CPU leía el vector de interrupciones desde la dirección I * 256 + valor bus de datos... Aunque ese valor del bus es siempre 255, en dicha información se decía que no se podía estar seguro, y lo mejor era llenar 257 direcciones de memoria, con valor FF, de tal manera que el vector de interrupciones seria FFFFH, independientemente del valor del bus. Luego el truco estaba en poner en la dirección FFFFH una instrucción "JR" (byte 24), y el desplazamiento del JR vendría en la dirección 0000h, un 243 en este caso, que genera al final un JR -13, que acababa saltando a la dirección 65524.
Luego la rutina de interrupciones se debía ubicar en la 65524. En el Inves, si se creaba esa tabla de 257 direcciones de memoria en RAM, la rutina de interrupciones funcionaba. Pero, en dicha información que tenía, o en otra que leí de otra revista, se decía que se podía aprovechar una tabla ya existente en la ROM, llena precisamente con valores FF. En la ROM original del Spectrum esa tabla empieza en la dirección 14446, y ocupa 1170 bytes, más que suficiente para usarlo como vector de interrupciones (poniendo el registro I con valor 57 por ejemplo). Pero... en el Inves... lo que pasaba en el Inves es que si hacías eso se reseteaba sin mas...
Mirando esas supuestas direcciones que debían estar a FF no era así, había código... Es más, desensamblando esas direcciones se puede ver que hay rutinas de paginación de 128k! Cual fue mi sorpresa al ver eso.... Pensé durante un tiempo que quizá el Inves tenía mas de 48k (aunque sí que tenía mas de 48, pero no 128 ;) ), e hice muchas pruebas a enviar sentencias OUT a los puertos de paginación a ver que pasaba. Pero no fue así. Lo que deduje es que esa ROM del Inves era casi la misma (excepto por los mensajes de error traducidos diferente y poco mas)


que la ROM 1 del Spectrum 128k (versión Española). Esas direcciones que están a FF en un Spectrum original, no lo están ni en los modelos 128k ni en el Inves.... Creo que hay algunos juegos que no funcionan en Inves (de la misma manera que no funcionan en modelos 128k) debido a esto.
Todos esos efectos o "particularidades" pasaron a formar parte de nuestras vidas, o mejor dicho de las del Inves. Esto duró aproximadamente 4 años, aproximadamente desde 1988 hasta 1992.
Por allá en 1992, en que yo ya sabía más de código máquina, me dio la vena de desproteger algunos juegos. Para el que no lo recuerde, desproteger consistía básicamente en pasar los juegos de carga turbo a carga normal. Mi hermano ya lo había conseguido con algún juego en turbo, y yo me atreví con algún Speedlock. Concretamente con el Enduro Racer...
Entre mi hermano y yo estuvimos montón de tiempo intentando desproteger el Speedlock, cosa que no conseguimos... hay montones de bucles de desencriptado y otros creados para evitar cualquier "intruso" que estuviese desensamblando con el MONS, como nosotros.
Pero viendo el código, hubo algo que nos llamó la atención... Uno de aquellos bucles consistía en modificar toda las direcciones de memoria, tanto RAM como ROM.... En cuanto pasamos aquel bucle, el borde de repente se quedó negro... y el Inves enmudeció completamente... de la misma manera que enmudecimos nosotros dos al ver aquello.
Dios! Habíamos visto el porqué el Enduro Racer nos dejaba siempre el Inves lelo...

A partir de aquél momento, vinieron un montón de pruebas, primero para reproducir el borde negro... Tarea fácil, pokear toda la ROM a 0 y listos.... Durante unos días no se nos ocurrió como revertir el efecto. Sabíamos como estropearlo y poco más.... Recuerdo cuando le comentamos el descubrimiento a mi padre... nos dijo: "veis esto de aquí? (refiriéndose a la ROM, mostrándonos una foto del interior del Inves) Pues es una EPROM, no una ROM, o sea que vigilad con escribir en ella....". No le hicimos mucho caso, y a mi se me ocurrió variar ese valor del poke a ROM a ver que sucedía.
Con una rutina sencilla en assembler que pokeaba en la ROM al instante, con un valor concreto, fui haciendo pruebas a cambiar el valor. Primero con 1, luego con 2... ahí fui viendo las distintas combinaciones, tanto las que afectaban al borde como las que afectaban al sonido. Lo fácil fue ver que con valor 255, se quedaba el Inves como al encenderlo. Pero lo mejor fue descubrir combinaciones de ese valor para que se escuchase el sonido de manera correcta en aquellos juegos con música a dos canales. Con el valor 23 se conseguía “silenciar” el bit del MIC del puerto FEH y permitía que se escuchase la música de manera perfecta!


Grabé esa pequeña rutina de poke a ROM, en una cinta vieja, que etiqueté como "Beep Inves". Y lo grabé no solo una vez, sino repetidas veces en toda la cinta entera (cinta de 60) por las dos caras, para que así, cada vez que quisiese jugar a alguno de esos juegos, metía la cinta del Beep Inves, sin rebobinar ni nada, y en un momento tenía resuelto el tema del sonido.
Excepto, claro está, en todos aquellos con carga Speed lock 2, dado que la propia rutina de carga se encargaba de pokear en la ROM con valor 0, fastidiándome cualquier poke que hubiera hecho antes.
A partir de aquel momento hice un “remember” de todos aquellos juegos que se escuchase mal la música y los fui probando uno por uno.
En el limbo quedaron algunas otras pruebas sobre esos pokes, como por qué habían combinaciones de pokes que dejaban el borde en un color concreto y que al hacer un comando "BEEP" del basic alteraban el borde con franjas de colores, de manera similar a las franjas de carga o grabación...
Recordamos que estábamos en el año 1992 aproximadamente. Un año mas tarde, un amigo mio, decidió cambiarme su Spectrum +2A por mi Inves, cosa que le agradecí mucho. Creo que estaba descontento con su Spectrum porque, por una parte, el sonido por la TV no se le oía bien (y mi Inves como se oía por el Speaker, no dependía de la TV). Y creo que la imagen tampoco se le veía muy bien.... sea como fuere, me lo cambió. Y yo conseguí un +2A y me olvidé de los problemas del Inves para siempre.... Aún así, mi hermano seguía conservando su Inves.
Un par de años mas tarde, hacia 1995, nos fuimos a vivir al extranjero. Como inicialmente nos fuimos "con lo puesto", los Spectrum se quedaron en casa. Pero teníamos un PC.... Yo tenía "mono" de mi Spectrum, y ya había probado algún emulador, como el Spectrum de Pedro Gimeno.
Me atraía la idea de crear mi propio emulador, así que, con las microfichas de Microhobby y un listado de instrucciones de Z80 (que sí que me había llevado con la mudanza) me puse a escribir mi propio emulador de Spectrum, en assembler, el ZXSpectr.
Por supuesto, cuando el emulador funcionaba bien como un Spectrum 48k, decidí agregarle emulación del Inves, con las particularidades del poke a rom.
Desde aquellos años hasta hace unos meses, poco contacto he tenido con el Inves y todo lo relacionado con él. Hasta recientemente, al conocer a Miguel Ángel y ver las pruebas que ha estado haciendo él.... Con todas sus pruebas se ha "cerrado el círculo", creo, a todas las interioridades del Inves. Todo lo que pude experimentar tiene sentido con la información que se trata en este artículo. Todas estas particularidades del Inves, tanto las que yo conocía, como las nuevas (y la manera correcta de como funcionan), las he implementado en mi otro (nuevo) emulador de Spectrum, ZEsarUX.

Tras las últimas actualizaciones del emulador, corregí un fallo en la instrucción OUT (n),A, en que el Z80 mete como byte alto de la dirección del puerto el

mismo valor de A, y como byte bajo el valor de n. Esto lo probé por curiosidad en el juego Hysteria, y me di cuenta de un comportamiento curioso de dicho juego:
El puerto al que envía el sonido el juego no es el FEh, sino que es XXFEh, donde ese XX es bastante aleatorio, dado que usa dicha instrucción OUT (n),A. Pero los valores de A van más allá de activar los bits de ear y mic o color del border, y se usan incluso los bits 5, 6 y 7 de puerto FEh. De esta manera, el puerto final, cubre todo el rango de 16 bits (0000H-FFFFH), cosa que en Inves tiene el efecto fatal de que el valor final enviado al puerto depende de toda la RAM del Inves. Por tanto, la música del juego, se oye ligeramente diferente que un Spectrum normal. A continuación listo unos cuantos valores para el puerto de sonido que genera el juego en cuanto empieza la música (justo después del "fade in" sonoro del principio):
puerto: 254 (FEH) valor 0 (0H)
puerto: 4350 (10FEH) valor 16 (10H)
puerto: 4350 (10FEH) valor 16 (10H)
puerto: 8446 (20FEH) valor 32 (20H)
puerto: 12542 (30FEH) valor 48 (30H)
puerto: 16638 (40FEH) valor 64 (40H)
puerto: 20734 (50FEH) valor 80 (50H)
puerto: 24830 (60FEH) valor 96 (60H)
puerto: 28926 (70FEH) valor 112 (70H)
puerto: 33022 (80FEH) valor 128 (80H)
puerto: 37118 (90FEH) valor 144 (90H)
puerto: 41214 (A0FEH) valor 160 (A0H)
puerto: 45310 (B0FEH) valor 176 (B0H)
puerto: 49406 (C0FEH) valor 192 (C0H)
puerto: 53502 (D0FEH) valor 208 (D0H)
puerto: 57598 (E0FEH) valor 224 (E0H)
puerto: 61694 (F0FEH) valor 240 (F0H)
Se puede ver, por ejemplo, que el último valor enviado, el del puerto 61694, dependerá de la misma dirección de memoria 61694. Es posible que el método de generar la música sea el mismo en otros juegos, y por tanto, la música también se escuchará diferente en Inves


# Descripción técnica:

## Características

En esencia, la lista de características compilada por Primitivo de Francisco en su análisis del número 108 de Microhobby es correcta, si bien con algún que otro matiz que ahora detallaremos. Dado que toda la información de esta sección es conocida por César (parte porque la descubrió él mismo, y parte porque ha sido informado a medida que se han ido descubriendo cosas durante la escritura de este trabajo), su emulador ZesarUX implementa, en modo Inves, todos los detalles, fallos y peculiaridades que se van a exponer.

•El refresco de la RAM dinámica del Inves no corre a cargo de los 4 circuitos integrados que señala en la placa. El refresco se genera desde el propio TAHC10. De hecho, en los últimos modelos de Spectrum no es cierto que el “chip específico de 40 patas” sea el responsable del refresco: en el Spectrum 16K/48K/Plus, los responsables del refresco de la memoria son: la ULA, para la memoria baja, y el Z80 para la memoria alta. El chip de 40 patas al que se refiere Primitivo es una PLD que sustituye a 6 chips en modelos anteriores, y cuyo cometido principal es la multiplexión de las señales del bus de direcciones del Z80 para acceder a las filas y columnas de las memorias dinámicas. Aparte de eso, se encarga, con ayuda de una célula de retardo, de generar las señales RAS y CAS para el acceso a la memoria alta. Contribuye, eso sí, a que los ciclos de bus que el Z80 crea para el refresco lleguen a la memoria. Es lo que solemos llamar, un chip de “glue logic”.

•El reloj maestro es cierto que es de 17,7345 MHz cuando en el Spectrum original son 14 MHz. Lo que no quiere decir que en el caso del Inves, la frecuencia de la CPU se obtenga como en el Spectrum de Sinclair, dividiendo la frecuencia del reloj maestro entre 4. En el caso del Inves, hay que dividir entre 5, para obtener 3,5469 MHz frente a los 3,5 MHz de la máquina original.

Así, y tras las matizaciones, a las características publicadas hay que añadir las siguientes:

•Primer modelo de Spectrum 48K con 64K de memoria RAM real de 150ns. *Es posible escribir en cualquiera de las 65536 posiciones de memoria de esa RAM, y el dato se guardará*, aunque lamentablemente, cuando leamos alguna posición comprendida entre 0 y 16383, lo que obtendremos será lo que haya en ROM en esa posición.

•*No existe contención de memoria*. Es decir, el Inves Spectrum funciona a su máxima velocidad (los referidos 3,5469 MHz) independientemente de a qué zona de memoria se acceda, o de que se acceda a los puertos de teclado, altavoz, etc. Es posible por tanto alojar rutinas sensibles a la temporización en la zona denominada “memoria baja” (direcciones 16384 a 32767)

•*Temporización de pantalla idéntica a la del Spectrum 128K*. Es decir, 228 T-estados por línea de pantalla y 311 líneas.

•Única señal de reloj maestro para todo el circuito, lo que significa que el reloj de color (4,433625 MHz) estará en fase con el reloj de pixel (7,0938 MHz) dando como resultado una mejor imagen en video compuesto, al desaparecer el efecto de “hormiga” en las líneas verticales que se tracen en pantalla. En el Spectrum 48K original se usaban relojes diferentes, por lo que no estaban en fase y ocurría este fenómeno.

•Desde que se dispara la interrupción por retrazo vertical hasta que se lee el primer byte de memoria de pantalla pasan 212 T-estados, en contraposición a los 14336 T-estados en el Spectrum 48K original.

•Puerto de joystick con norma Kempston disponible en el puerto $DF (223). También es posible acceder al joystick por el puerto habitual, el $1F (puerto 31). Al contrario de lo que hizo Amstrad con el +2A/B/3, el puerto de joystick tiene el conexionado compatible con Atari, y además el pin “COMMON” del conector está conectado a masa, en lugar de a +5V como otras interfaces, o a una señal digital, lo que mejora la compatibilidad con ciertos joysticks que usan contactos electrónicos en lugar de mecánicos.

•El puerto $FF devuelve siempre el valor 255. De hecho, cualquier puerto no implementado devolverá 255. El teclado, altavoz, y EAR se acceden por el puerto $FE pero en realidad valdrá cualquier puerto cuyo bit A0 valga 0, como en el Spectrum original.

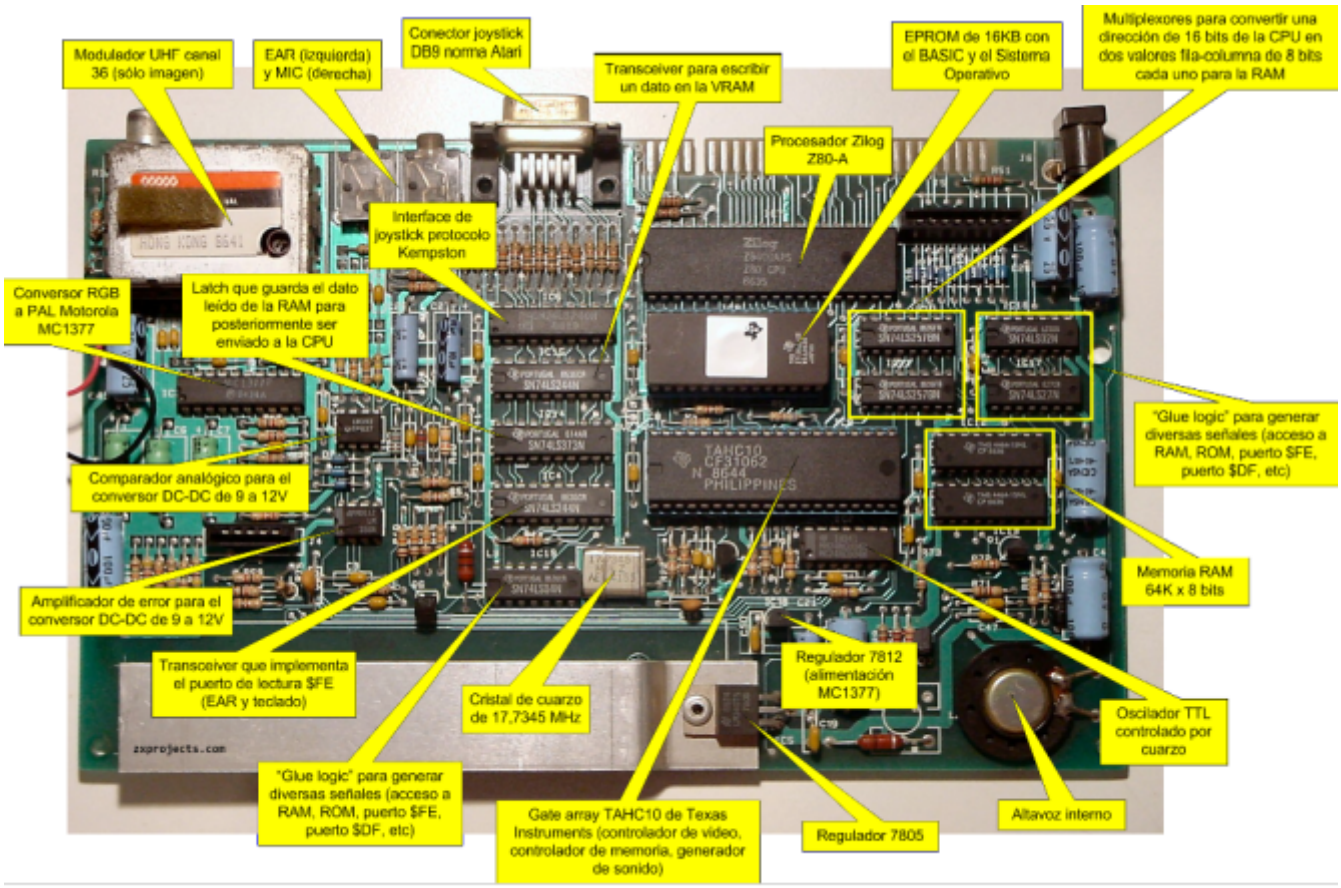
•En ausencia de periféricos externos que modifiquen este valor, una interrupción IM 2 siempre usará como vector el valor $FF. Sin embargo veremos más tarde que esto tiene “truco”.

•Auto-mute en el jack de MIC: si se enchufa un conector a la toma de MIC, el altavoz interno es silenciado.

## Esquema de bloques del ordenador

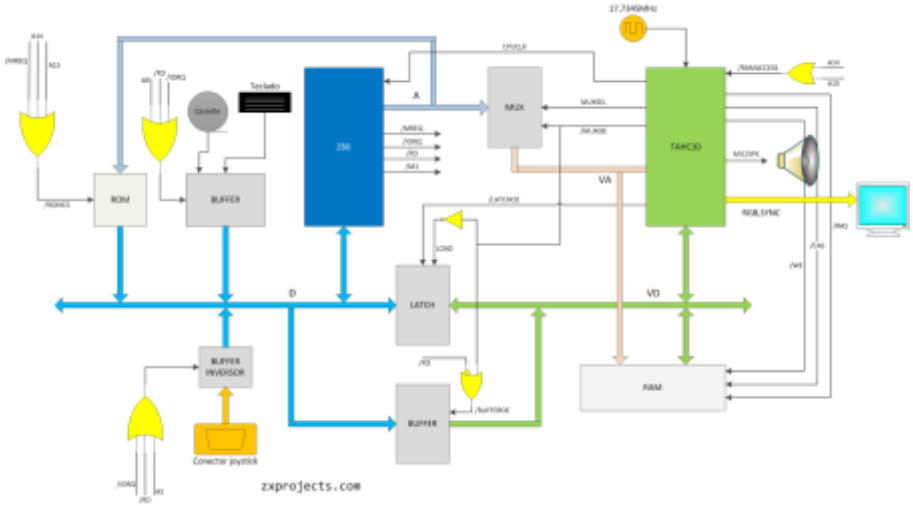
Emulando al gran Primitivo en su artículo del número 108 de Microhobby, nos atrevemos a publicar nuestro propio gráfico de la placa del Inves, detallando, ahora sí, la verdadera función de todos y cada uno de los chips que lo componen:

El Inves Spectrum+ es, como sabemos, un microordenador diseñado en torno al procesador Z80-A, que para esta máquina en particular suele ser la versión que fabrica directamente Zilog. La interface del procesador con la memoria ROM es directa, y sólo se necesita una puerta en IC17 y otra en IC2 para activarla cuando se quiere acceder a los primeros 16K del mapa de memoria. El acceso a la RAM es otro




Descripción de los elementos internos de la placa del Inves Spectrum+

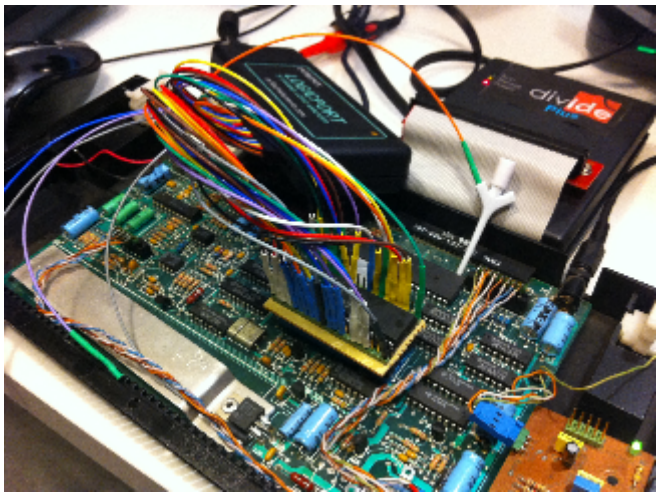
El primer análisis (aún en proceso) que se ha realizado sobre la placa del Inves ha resultado en el siguiente diagrama de bloques.




cantar: al ser memoria DRAM se necesita multiplexar los 16 bits del bus de direcciones en un valor de fila y otro de columna, ambos de 8 bits. De ello se encargan IC20 e IC21. Las señales de RAS, CAS y ecritura en DRAM son generadas directamente por el TAHC10.

Los dos puertos de lectura del ordenador, es decir el puerto $FE para leer teclado y señal EAR de cinta, y el puerto $DF para leer el joystick Kempston, están implementados con lógica discreta, o sea, con chips fuera del TAHC10. Aquí por tanto hay una diferencia sustancial respecto a las

máquinas de Sinclair, en las que la gestión de la señal EAR y del teclado se hace dentro de la ULA. En concreto, el buffer triestado IC4 se encarga de suministrar al Z80 la información del teclado y EAR, y el buffer triestado inversor IC6 hace lo propio con las señales del joystick. Este último invierte las señales que hay en su entrada para así poder usar un voltaje de 0V como señal de común en el conector de joystick: así, una dirección seleccionada o un disparo pulsado se convierten en una entrada 0 (cortocircuitada al común de señales), y que el buffer invierte y traduce como 1, que es lo que se espera en una interfaz Kempston.
El circuito integrado específico TAHC10 será objeto de una sección aparte, pero comentar que se encarga practicamente de lo mismo que la ULA en el



Spectrum: generación de la imagen y del sonido, temporización y acceso a la RAM, generación de la señal de interrupción y reloj para el Z80, y generación de la señal de reloj de color para el codificador PAL. Como ya se señaló, las tareas de "escuchar" la señal de EAR o leer del teclado no se realizan en ella.
Para obtener la información que se presenta en las siguientes secciones se ha analizado un Inves Spectrum+ usando un analizador lógico de 32 canales, y un DivIDE para lanzar programas rapidamente, así como para poder ejecutar código arbitrario tanto en la zona de ROM como en la de RAM. El objeto principal de estudio ha sido el TAHC10, y en él se han usado la mayoría de las sondas del analizador.

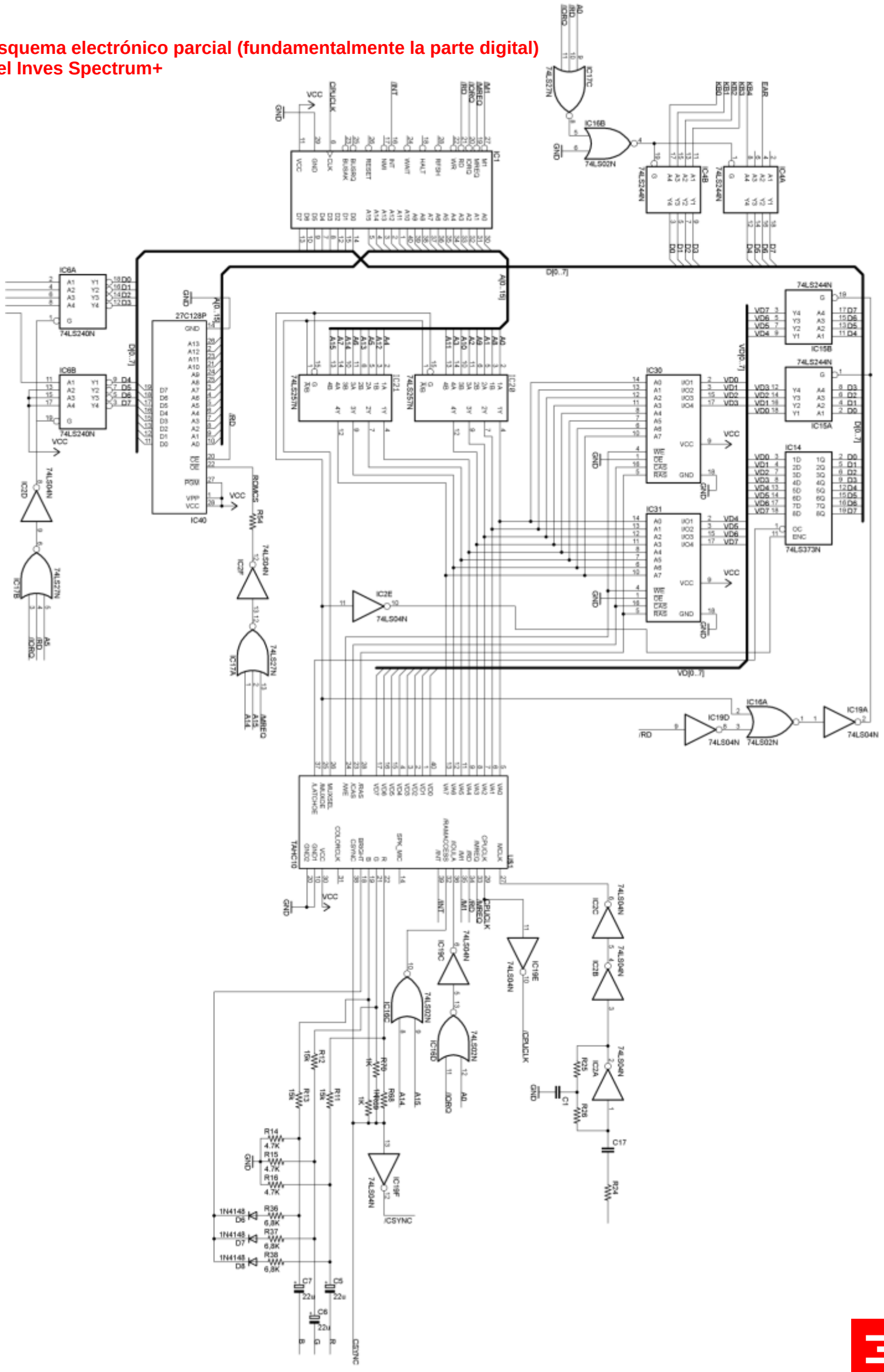
El TAHC10 del Inves Spectrum+ bajo el analizador lógico
Para poder operar con la máquina aun estando abierta, se ha usado un conversor de teclado PS/2 – Spectrum versión interna, una de las unidades de prototipo que fabriqué hace años. La salida de video es el típico mod de video compuesto con transistor, que funciona sin problemas en el Inves.
A continuación se presenta un esquema eléctrico parcial del Inves. Aún hay bastantes líneas que trazar en la placa, y gran parte de la etapa analógica del equipo están aún sin documentar. Sí que lo está la parte digital, y gracias a ello se ha podido obtener una descripción detallada de su funcionamiento, comenzando por el componente más misterioso: el circuito integrado TAHC10 de Texas Instruments.
La versión más actualizada de este esquemático podrá encontrarse en: http://www.zxprojects.com/inves/

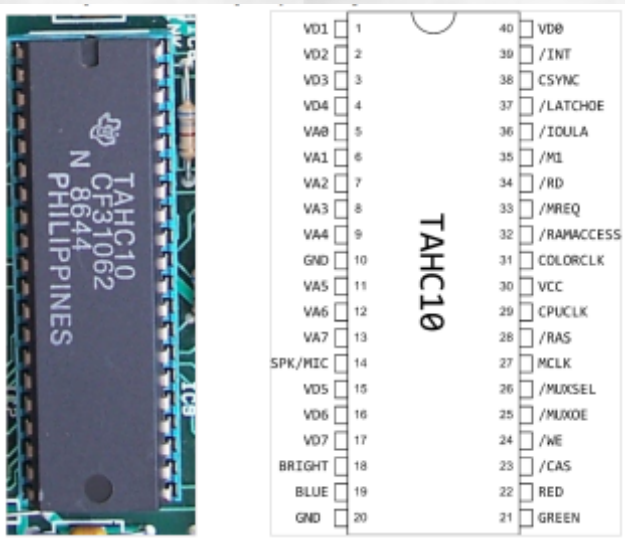
Esquema electrónico parcial (fundamentalmente la parte digital) del Inves Spectrum+ **(página siguiente)**

## El TAHC10 de Texas Instruments

El TAHC10 es un chip semicustom tipo gate array de Texas Instruments de la serie TAHC, fabricado en tecnología HCMOS de 3 micrómetros. El TAHC10 es el de mayor capacidad de la serie, con un equivalente a 1120 puertas NAND de dos entradas, y 40 buffers de entrada/salida. Cualquier puerta lógica, y por extensión, cualquier circuito digital (flip-flops, contadores, sumadores, registros de desplazamiento, etc) puede hacerse unicamente con puertas NAND de estas características. Debido a esta estructura, el TAHC10 no es válido para generar señales analógicas, por lo que no puede generar, por ejemplo, una señal YUV como hacía la ULA de Ferranti, o una señal de audio en varios niveles, tal y como se ha comentado anteriormente.
Estos circuitos se fabrican capa a capa, siendo la última una capa llamada de metalización, en la que usando una difusión de aluminio se crea un entramado de pistas que une a los transistores MOS que hay en las capas inferiores. Dichos transistores están siempre en el mismo sitio sea cual sea el TAHC10 que se use. Es la capa de metalización la que proporciona la "personalización" del chip conforme a los deseos del cliente.
De las 1120 puertas, se pueden acceder a un máximo de 896 si se usa el sistema de diseño asistido por ordenador que provee el fabricante, aunque pocos diseños pueden usar más allá del 50% de éstas, dado que hay que tener en cuenta ciertas restricciones de layout y rutado (como sólo hay una capa de metal, algunas puertas lógicas han de "sacrificarse" para poder soportar trazados más complejos)
Una vez "personalizado", el TAHC10 se encapsula en un formato DIP de 40 pines, idéntico al de la ULA, pero totalmente incompatible con ésta. Su patillaje es el siguiente:

**Esquema electrónico parcial (fundamentalmente la parte digital) del Inves Spectrum+**






Fotografía y pin-out del circuito integrado TAHC10 de Texas Instruments

A continuación se ofrece la descripción de sus pines. La dirección es relativa al TAHC10. Los niveles son todos CMOS (compatible TTL):

| Nombre | Dirección | Descripción |
|---|---|---|
| VA0-7 | OUT | Dirección de 8 bits de fila/columna generada por el TAHC10 cuando quiere leer la DRAM para formar la imagen de pantalla. En alta impedancia cuando es el procesador quien pone la dirección de fila y columna a través de los multiplexores |
| VD0-7 | IN | Bus de datos de 8 bits del TAHC10. Conectado directamente a la DRAM, e indirectamente al bus de datos del Z80 a través del buffer IC15 |
| /RAS | OUT | A la entrada de selección de fila de las DRAMs |
| /CAS | OUT | A la entrada de selección de columna de las DRAMs |
| /WE | OUT | A la entrada de habilitación de escritura de las DRAMs |
| SPK/MIC | OUT | Salida de MIC y altavoz. |
| BRIGHT | OUT | Señal de brillo de la imagen |
| RED | OUT | Señal de rojo de la imagen |
| GREEN | OUT | Señal de verde de la imagen |
| BLUE | OUT | Señal de azul de la imagen |
| CSYNC | OUT | Señal de sincronismo compuesto de la imagen. También, en forma invertida, sirve como entrada para el conversor estático de 12V |
| /MUXSEL | OUT | Habilita la entrada A o B de los dos multiplexores. La entrada A se habilita cuando esta señal vale 0, y está conectada a los bits A0-A7 del bus de direcciones del Z80. La entrada B se habilita cuando la señal vale 1, y está conectada a los bits A8-A15 del bus de direcciones del Z80 |
| /MUXOE | OUT | Los dos multiplexores permiten triestado, y esta señal controla esa posibilidad. Cuando vale 0, la salida de los multiplexores entra al bus de direcciones de las DRAMs permitiendo que el Z80 las direccione. Cuando vale 1, el bus de direcciones del Z80 está aislado del bus de direcciones de la memoria. Una versión invertida de esta señal controla la carga del latch IC14, que guarda el último dato accedido en la memoria RAM. Es decir, cuando vale 1, el dato que está presente en el bus de datos de la memoria RAM se carga en este latch. |
| MCLK | IN | Reloj maestro del TAHC10. 17.7345MHz, señal cuadrada, niveles TTL |
| CPUCLK | OUT | Reloj directo al Z80. Es el valor de MCLK dividido entre 5, es decir, 3.5469MHz, señal cuadrada, niveles TTL |
| COLORCLK | OUT | Reloj de la subportadora de color. Es el valor de MCLK dividido entre 4, es decir, 4.433625MHz, señal cuadrada, niveles TTL |
| /RAMACCESS | IN | Vale 0 cuando el Z80 accede a una dirección en el rango $4000-$FFFF |
| /LATCHOE | OUT | Control de habilitación triestado del latch IC14. Hace que la salida de este latch alcance el bus de datos de la CPU |
| /MREQ | IN | Señal de acceso a memoria desde el Z80 |
| /RD | IN | Señal de petición de lectura desde el Z80 |
| /M1 | IN | Señal de indicación de ciclo M1, desde el Z80 |
| /IOULA | IN | Vale 0 cuando el Z80 inicia un ciclo de bus de E/S a una dirección de puerto que tenga el bit A0 = 0 |
| /INT | OUT | Señal de interrupción enmascarable para el Z80 |
| VCC | Power | Entrada de alimentación de 5V |
| GND | Power | Común de señales y alimentación del chip (2 pines) |

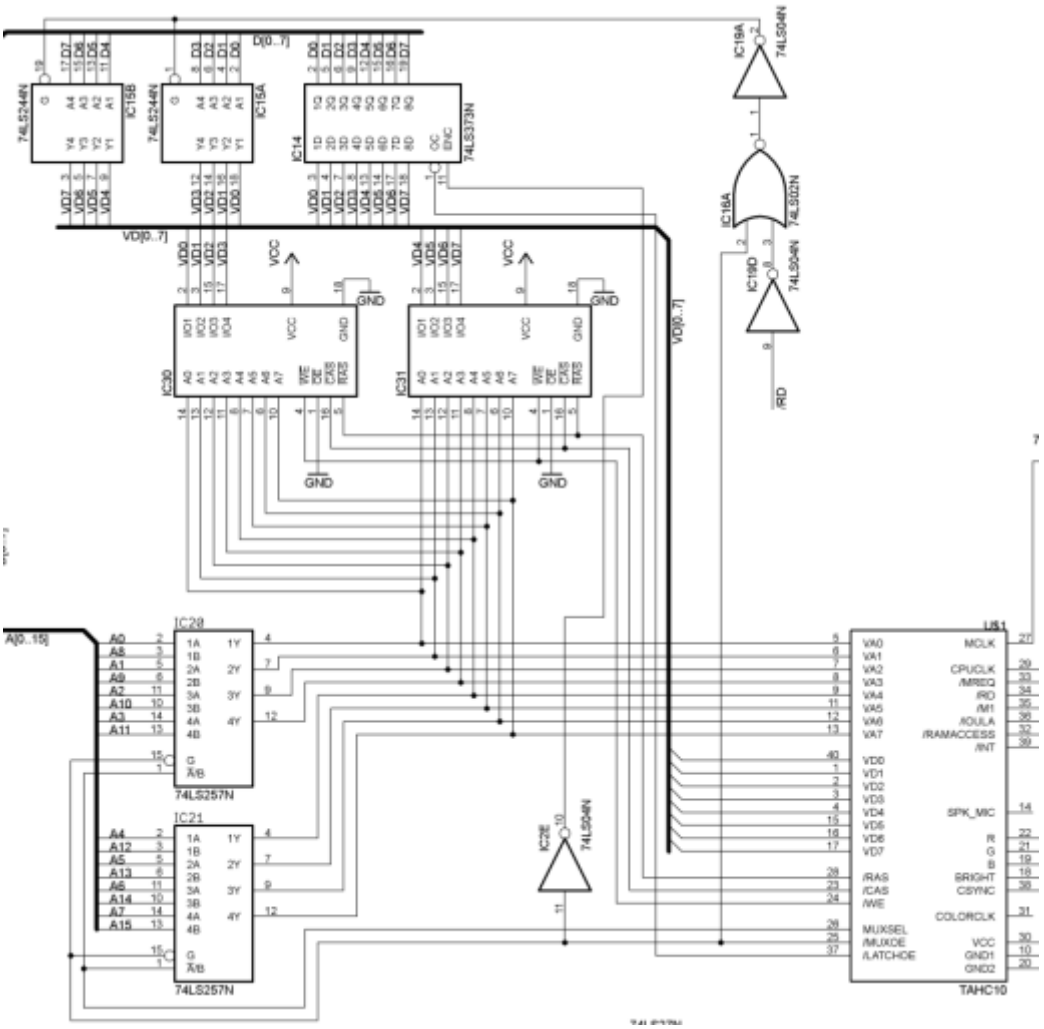
Al ser CMOS, el consumo de este gate array es bastante menor que el de la ULA de Ferranti, cuya tecnología es bipolar. De hecho en el Inves el chip que más corriente consume es quizás el propio Z80. Si el lector ha tenido tiempo para curiosear el esquema eléctrico que hemos obtenido, y la disposición de señales del TAHC10, se habrá dado cuenta de algo: no se usa la señal WR del procesador. Es más: está desconectada. El único sitio a donde va esa señal en la placa del Inves es al conector de expansión. Nada, en toda la placa del Inves, usa la señal WR. Y aun así, el procesador es capaz de leer y escribir en memoria. Lo cierto es que una de las misiones del TAHC10 es “adivinar” cuándo se va a producir un ciclo de bus de escritura en memoria antes de que el Z80 señalize de ello. Esto es necesario para resolver la contienda de memoria, que será descrita a continuación. Por decirlo de alguna manera, el TAHC10 es muy impaciente y quiere saber cuanto antes si el ciclo actual es de lectura o escritura. Lo malo es que a veces su “impaciencia” le juega malas pasadas.

## Implementación de la solución para la contienda de memoria

Una de las cosas que más choca en el Inves Spectrum+ es que toda la memoria RAM se implementa con sólo dos chips, cuando en los modelos de Sinclair necesitaron 16 chips. Por supuesto, 4 años de diferencia entre los dos modelos dan para muchas innovaciones, sobre todo en la integración de memorias, pero no es eso lo chocante. En realidad, lo que extraña, o debería haber extrañado a los chicos de Microhobby cuando hicieron su análisis del Inves, es que hay un único banco de memoria.

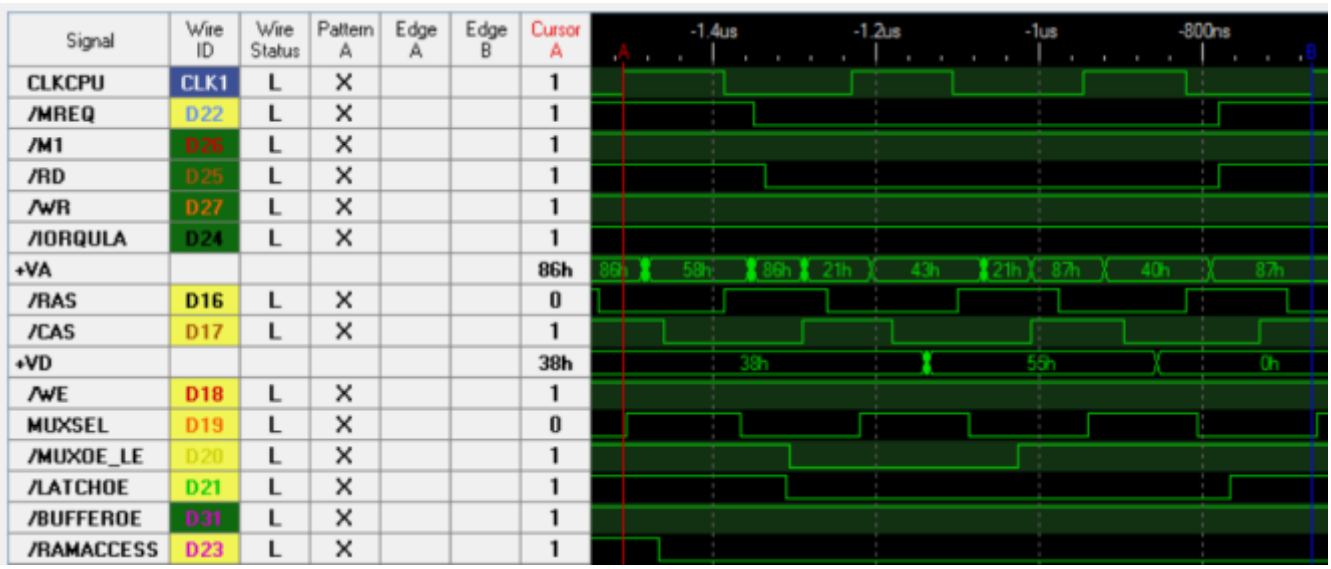
Un profano puede pensar que al haber dos chips, cada uno de ellos se ocupa de 32KB, y puede dar por sentado que uno de ellos se usará para el banco contenido (direcciones 16384 a 32767) y el otro para el banco no contenido (direcciones 32768 a 65535). De hecho el clon Harlequin usa esa

disposición con dos memorias 62256.
Pero no: las dos memorias no direccionan segmentos diferentes del mapa de memoria: ambas funcionan a la vez. Cada una suministra 4 bits de datos, así que las dos a la vez suministran 8 bits. A efectos prácticos se comportan como una única memoria dinámica de 64KB. Todo el mapa de memoria RAM: banco contenido y no contenido, está en el mismo chip, lo que significa que para que el Z80 acceda a una dirección de RAM, cualquiera que ésta sea, tendrá que acceder a la única RAM de que dispone. Esto se puede deducir mirando únicamente el tipo de memoria que se usa: TMS4464 (64K x 4 bits, las mismas que se usan en el +2A/3, en la gama CPC+ de Amstrad y en el C64-C, por poner algunos ejemplos). Así que la cuestión es: si la ULA (ehem… el TAHC10) y el Z80 acceden a la misma RAM, sólo pueden pasar dos cosas: o que todo el espacio de memoria RAM está en contienda, o que ninguno lo esté.
Un clon de Spectrum con toda la memoria en contienda se hubiera notado desde el minuto 0 en practicamente todos los juegos que se hubieran probado, y ni que decir tiene que cualquier juego que tuviera una rutina de carga no estándar en RAM no hubiera siquiera cargado. Pero entonces, ¿hay contienda aunque sea pequeña, o no la hay? La respuesta corta: No, no la hay. En los cronogramas siguientes, la señal de reloj de CPU aparece en la primera línea, y se puede ver que independientemente de la interacción que haya entre CPU y TAHC10, esta señal no cambia, ni se corta. El TAHC10 por otra parte no comanda ninguna de las otras señales que pudieran servir para parar al Z80, tal como /WAIT o /BUSRQ.
El mecanismo para la resolución de la contienda implica al propio TAHC10 y algunos de sus componentes. Al igual que el mecanismo de contienda desarrollado por Altwasser para la ULA del ZX Spectrum, el desarrollado para el TAHC10 denota un amplio conocimiento de la temporización de los buses en el Z80. La parte del circuito que nos interesa se ha destacado en la siguiente figura.




IC30 e IC31 son los dos chips de memoria RAM. Como se puede ver, todas sus señales, excepto las del bus de datos, están compartidas en ambos integrados, así que a todos los efectos, los consideraremos como un único módulo al que llamaremos sencillamente RAM.
El bus de direcciones de esta memoria puede tener un valor proveniente, bien del TAHC10, o del Z80 a través de dos multiplexores, IC20 e IC21. Estos multiplexores son muy parecidos a los que existen en el ZX Spectrum, pero con la salvedad de que los usados en el Inves tienen la capacidad de ponerse en modo de alta impedancia. Se activan cuando /MUXOE vale 0. MUXSEL indica qué entrada del multiplexor es la que se verá en la salida. La

dirección de fila aparece cuando MUXSEL vale 0, y la dirección de columna cuando MUXSEL vale 1.
El bus de datos de la RAM se denomina VD y está fisicamente separado del bus de datos del procesador (D en el esquemático). Al ser la RAM el único elemento que vuelca datos al bus, no necesita su



| Signal | Wire ID | Wire Status | Pattern A | Edge A | Edge B | Cursor A |
|---|---|---|---|---|---|---|
| CLKCPU | CLK1 | L | X | | | 1 |
| /MREQ | D22 | L | X | | | 1 |
| /M1 | D26 | L | X | | | 1 |
| /RD | D25 | L | X | | | 1 |
| /WR | D27 | L | X | | | 1 |
| /IORQULA | D24 | L | X | | | 1 |
| +VA | | | | | | 86h |
| /RAS | D16 | L | X | | | 0 |
| /CAS | D17 | L | X | | | 1 |
| +VD | | | | | | 38h |
| /WE | D18 | L | X | | | 1 |
| MUXSEL | D19 | L | X | | | 0 |
| /MUXOE_LE | D20 | L | X | | | 1 |
| /LATCHOE | D21 | L | X | | | 1 |
| /BUFFEROE | D31 | L | X | | | 1 |
| /RAMACCESS | D23 | L | X | | | 1 |

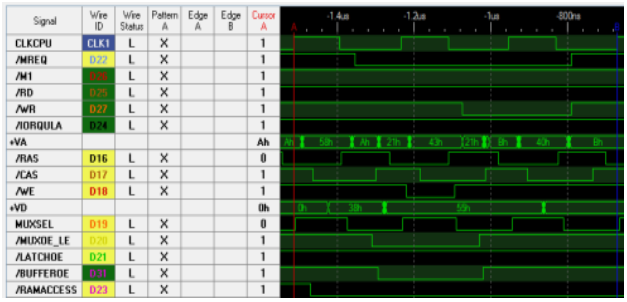


control de triestado, por lo que la señal OE de la misma está siempre activa, conectada a GND.
Aunque VD y D están separados, hay dos elementos que los conectan, controlados por el TAHC10. Uno es IC14 y el otro Ic15.
IC15 es un latch, un elemento de memoria, que es capaz de almacenar 1 byte de información. Su entrada es el bus de datos VD de la memoria, y su salida, el bus de datos del Z80, el bus D en el esquemático. Este latch es capaz de almacenar un byte proveniente de la memoria y guardarlo el tiempo que sea necesario para que la CPU lo recoja cuando quiera. Se carga con nuevos datos cada vez que su entrada ENC vale 1, y eso ocurre cuando /MUXOE vale 0.
IC14 es un buffer triestado. Cuando se activa, permite el paso desde el bus de datos de la CPU al bus de datos de la memoria. Es decir, el sentido es de D a VD. Esta activación se produce cuando /MUXOE vale 0 y /RD vale 1. En los cronogramas, a esta señal se la denomina /BUFFEROE.
Así que tenemos que cada vez que el Z80 quiere usar la RAM, /MUXOE vale 0, y esto además de abrir los multiplexores, hace que la conexión entre VD y D se establezca, bien en un sentido, o en otro.
Si la operación es una lectura, /RD vale 0 a la misma vez que el Z80 activa /MREQ para indicar un acceso a memoria. Un poco antes el bus de direcciones ya se ha estabilizado con la dirección de memoria a la que se quiere acceder. A partir de ahí, y dependiendo del ciclo en el que estemos, hay entre 1 y 2,5 ciclos de reloj para terminar el acceso de lectura. El primer caso es el de un acceso para leer una instrucción (ciclo M1). El segundo caso es un acceso a memoria para leer un dato.

*Cronograma de un ciclo de lectura de datos a memoria RAM*

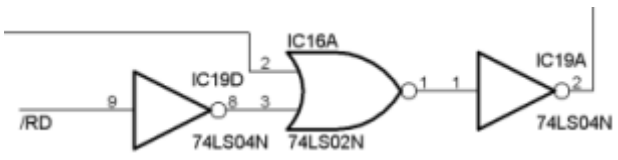
El cronograma muestra un ciclo de bus de lectura de datos. Este ciclo de bus dura 3 ciclos de reloj y se ha enmarcado entre los cursores A (rojo) y B (azul). Cuando /MREQ y /RD se activan, lo hace también /MUXOE, permitiendo que la dirección multiplexada proveniente del Z80 llegue a la RAM. MUXSEL envía primero la fila y luego la columna, asegurándose de que ambas están presentes en el bus de direcciones de la memoria cuando se activan /RAS y /CAS respectivamente. /LATCHOE también se activa, permitiendo a la CPU leer su contenido. La señal de carga de este latch también está controlada por /MUXOE de forma que cuando vale 0, el latch está en modo transparente, lo que significa que cualquier dato que tenga en su entrada se copia a su salida. Cuando /MUXOE vuelve a valer 1 el latch retiene el último dato que había en su entrada. Así, el tiempo que la RAM ha estado realmente ocupada con el procesador ha sido el tiempo que /MUXOE ha estado activo a 0, aproximadamente 1 ciclo de reloj de CPU. El Z80 no lee realmente el dato hasta el flanco negativo del ciclo 3 del ciclo de lectura, unos nanosegundos antes de que /MREQ y /RD vuelvan a desactivarse. Quien le proporciona el dato es el latch. Para entonces la RAM hace ya tiempo que ha vuelto a pasar a control del TAHC10 para seguir leyendo datos de la pantalla. En el cronograma se ve que

estaba leyendo un byte del área de atributos (valor $38 en VD) desde la posición $5886 (que se puede ver en el bus VA) cuando el Z80 requirió la lectura de RAM en la posición $4321. El valor leído, $55 ocupó el bus de datos de la RAM durante un ciclo de reloj de CPU, para luego continuar con la lectura del siguiente dato de pantalla, un byte en la posición $4087, área de bitmap con el valor $00.
En un acceso de escritura, la señal /WR del procesador ocurre demasiado tarde, y esto trastoca la estrategia diseñada en el TAHC10 de darle a la CPU acceso a la memoria lo más pronto posible, y quitárselo también lo más pronto posible. Este es el cronograma de una escritura a RAM:



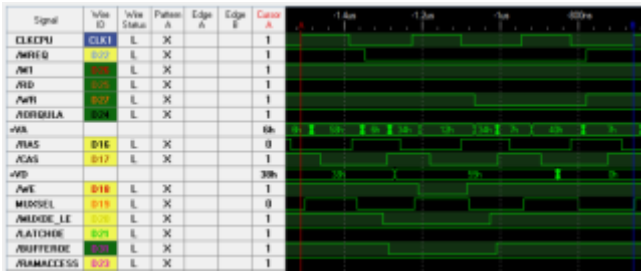

*Cronograma de un ciclo de escritura a memoria RAM*

Como en el caso anterior, en cuanto se activa /MREQ y se sabe que esto es un acceso a RAM (se activa /RAMACCESS), el TAHC10 activa /MUXOE y comienza la selección de fila y columna a la RAM. En este caso, y como no se ha observado que se haya activado /RD, el TAHC10 adivina que esto es un ciclo de escritura a RAM. El resultado es que se activa /BUFFEROE.  Esta señal se genera fuera del TAHC10, usando dos inversores y una puerta NOR. La entrada es la señal /RD del Z80 y la señal /MUXOE del TAHC10.



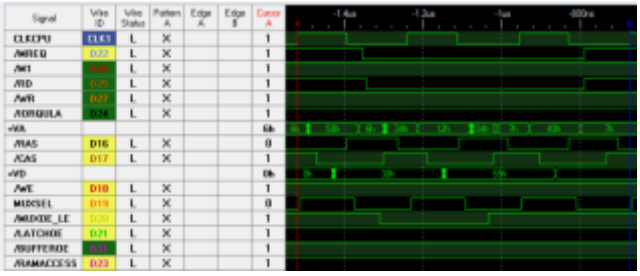

Circuito generador de la señal /BUFFEROE. La otra entrada de la NOR es /MUXOE.

/LATCHOE no se activa. El resultado neto es que se abre una vía de comunicación desde el bus de datos del Z80 al bus de datos de la memoria y el dato ($55) se escribe en la posición $4321 de la RAM. Todo esto nanosegundos antes de que el propio Z80 active la señal /WR. Para cuando esto pasa, la RAM ya ha vuelto a estar controlada por el TAHC10 que sigue su tarea de generación de la imagen.
¿Y si la dirección de memoria en la que se quiere escribir pertenece al espacio de ROM? Esto es lo que se refleja en el siguiente cronograma:



*Cronograma de un ciclo de escritura a una dirección de memoria en el espacio de ROM*

Como se puede observar, el comportamiento es idéntico al mostrado en la escritura a RAM. En aquella se escribe a la dirección $4321 el valor $55, y aquí se escribe a la posición $1234 también el valor $55. El ciclo de escritura RAS-WE-CAS se cumple, el dato de la CPU ($55) está visible en el bus de datos de la RAM y efectivamente, ésta guarda el dato. Entonces, ¿cómo es que al leer de posiciones de la ROM no leemos ese dato guardado? Obviamente debe haber un mecanismo para que esto no ocurra. De otra forma se habría detectado este comportamiento en las primeras pruebas con el ordenador. Esto es lo que ocurre:



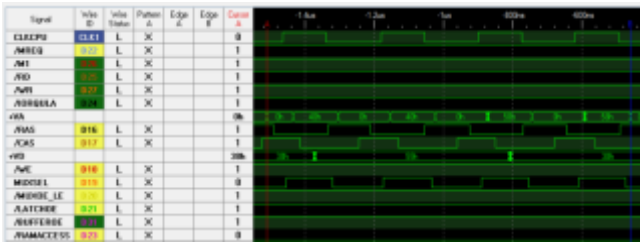
*Cronograma de un ciclo de lectura de memoria en una dirección del espacio de ROM*

Cuando el Z80 pide leer de memoria, sea RAM o ROM, el TAHC10 obediente realiza el acceso RAS-CAS necesario en la RAM, y se puede observar en el cronograma que en la posición de memoria $1234 sigue guardado el valor $55 que escribimos antes. El latch IC14 se pone en modo transparente y permite la entrada del dato en él, pero sin embargo, la señal /LATCHOE no se activa, por lo que el dato que entra en el latch, no sale de él, y no llega al bus de datos de la CPU. Ésta, lo que lee, es el dato que le suministra la EPROM que también ha sido activada en este ciclo

de bus.
De aquí podemos deducir que la acción de /LATCHOE debe estar de alguna forma controlada por la señal /RAMACCESS de forma que si la segunda no se activa, la primera no lo hace.
El siguiente cronograma muestra otro ciclo de lectura, esta vez un ciclo de búsqueda de instrucción (ciclo M1) enmarcado entre los cursores A y B. Este ciclo de bus dura 4 ciclos de reloj, pero solamente se lee la memoria en los dos primeros. Aunque el comportamiento de las señales es el mismo que en caso anterior, ilustra el mecanismo que implementa el TAHC10 para discernir entre un acceso de escritura a RAM y otro que no lo es, aunque en ambos casos se activen las mismas señales.

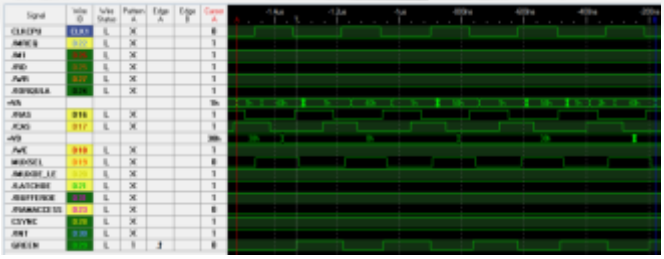
*Cronograma de un ciclo de bus M1*

Según la descripción del comportamiento que hemos hecho para el ciclo de escritura, en los ciclos 3 y 4 del ciclo de bus M1 se da el mismo comportamiento para la señal /MREQ y el contenido del bus de direcciones: en efecto “parece” que se quiere acceder a RAM y no es una lectura porque /RD no se ha activado. Nosotros sabemos que no es una escritura porque /WR tampoco se ha activado pero esto no lo sabe el TAHC10.
Lo que sí sabe el TAHC10 es que acaba de terminar un ciclo M1, y lo sabe porque puede leer la señal /MREQ, /RD y /M1. Cuando estas tres señales pasan de estar activas a estar inactivas, el ciclo M1 ha terminado, así que lo que ocurre es que durante los dos ciclos de reloj de CPU siguientes, ignora cualquier contenido de las señales /MREQ y /RAMACCESS, no dejandose engañar por algo que parece un ciclo de escritura a RAM.
Con estos cronogramas hemos visto que una operación de lectura o escritura a RAM sólo consume un ciclo de reloj de CPU, independientemente de la duración del ciclo vista desde el procesador, pero ¿cómo se comparte realmente la RAM entre el TAHC10 y el Z80? Para ello, vamos primero a ver cómo lee el TAHC10 la RAM cuando tiene que generar la pantalla, y suponiendo que la CPU está fuera de la ecuación. Para ello, dejamos pulsado el botón de RESET del Inves. Eso hace que todas las señales de la CPU pasen a estado inactivo y su bus de datos quede en alta impedancia. El cronograma siguiente muestra el momento en que se comienza a leer la pantalla:



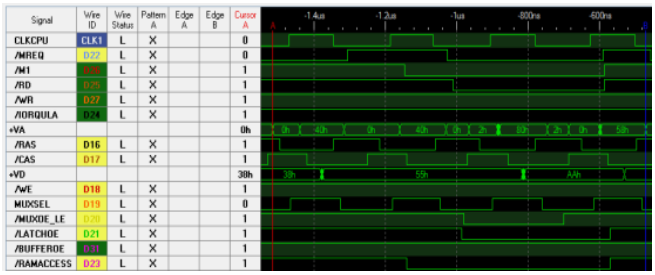
*Cronograma de lectura de la dirección $4000 y $5800 por parte del TAHC10 para generar la imagen. El Z80 está desactivado.*

La señal MUXSEL sigue generandose, aunque no gobierne en este caso los multiplexores que están desactivados. La dirección de fila y columna que vemos en VA, sincronizada con los flancos de /RAS y /CAS nos dice qué dirección de memoria está leyendo el TAHC10.
Para empezar, lo primero que se observa es que, a diferencia de la ULA de Ferranti, el TAHC10 usa como direcciones de memoria las mismas que ve la CPU. En la ULA de Ferranti la primera posición de pantalla es $0000 aunque para el Z80 sea $4000. En el TAHC10, la primera posición de pantalla es también $4000 y así se ve en el cronograma, en donde el primer ciclo de lectura muestra una dirección de fila $00 y una de columa $40. El valor leído de ahí es $55 que en binario es 01010101. Visualmente, una serie de pixeles apagados y encendidos unos junto a otros en la esquina superior izquierda de la pantalla.
La lectura de la posición $40000 se repite de hecho dos veces, para pasar a continuación a leer la posición $5800, que es la dirección de atributo que le pertenece a esta dirección de bitmap. De aquí se lee el valor $38, es decir, no flash, no brillo, paper blanco y tinta negra. La lectura de la dirección $5800 también se repite dos veces.
Tras estos 4 ciclos de lectura (2 repetidos para leer $4000 y otros 2 para leer $5800) vienen otros 4 ciclos para leer las posiciones $4001 y $5801. Para entonces, el TAHC10 ya empieza a mostrar los pixeles que ha leido en los ciclos anteriores.



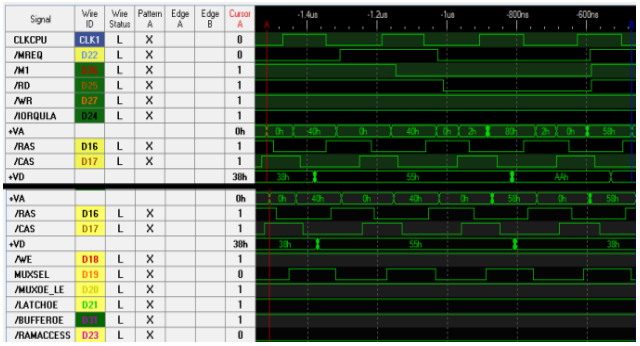
*Cronograma de la temporización de los píxeles en el TAHC10*

En el cronograma se ha añadido la información del color verde. Los píxeles pintados son en realidad blancos y negros, siendo el primero blanco (color del paper). El borde es negro en este cronograma.
Para leer la información de 1 byte de bitmap + 1 byte de atributos se han usado por tanto 4 ciclos de reloj de CPU, que son 8 ciclos de reloj de pixel. Estos mismos 8 ciclos de reloj es lo que tarda esa información en consumirse en el registro de desplazamiento interno que el TAHC10 posee para ir enviando píxeles al monitor, así que mientras pueda leer datos de la memoria a ese ritmo, la generación de pantalla no se verá afectada.
Pero, ¿por qué repite dos veces la lectura de cada dato si con leerlo una vez sería suficiente? Es ahora cuando entra la CPU en este juego:




*Cronograma de lectura de la dirección $4000 y $5800 por parte del TAHC10 para generar la imagen. El Z80 está activo.*

Este cronograma muestra de nuevo al TAHC10 leyendo la información de las direcciones $4000 y $5800 para formar los primeros 8 píxeles de la imagen. Sin embargo la CPU requiere leer una instrucción. El TAHC10 usa uno de los dos ciclos de acceso a RAM y se lo “presta” a la CPU para que haga su acceso. El otro ciclo lo sigue usando para sí misma. Esto ocurre en el cronograma con el primero de los dos accesos para leer la dirección de atributo $5800. El TAHC10 empieza a poner la dirección de fila ($00) de la posición de atributo pero la CPU requiere acceso a la RAM y el TAHC inmediatamente cambia el valor que había puesto por el valor $02, que es la parte menos significativa de la dirección donde la CPU quiere leer datos ($8002), a la vez que abre los multiplexores y el latch. Una vez que termina este primer acceso, realiza el segundo ahora sí con la dirección completa de atributo, $5800. Si superponemos ambos cronogramas veremos que los datos de bitmap y atributo están dentro del TAHC10 en el mismo tiempo independientemente de que la CPU entre en contienda por la RAM. Cuando esto ocurre, el TAHC simplemente presta uno de sus accesos para la CPU, a sabiendas de que el otro lo tiene para sí mismo. En la siguiente figura, el cronograma superior corresponde a un acceso a RAM compartido entre el TAHC y el Z80, el mismo que hemos puesto antes. El cronograma inferior corresponde a la misma situación, pero sin el Z80.



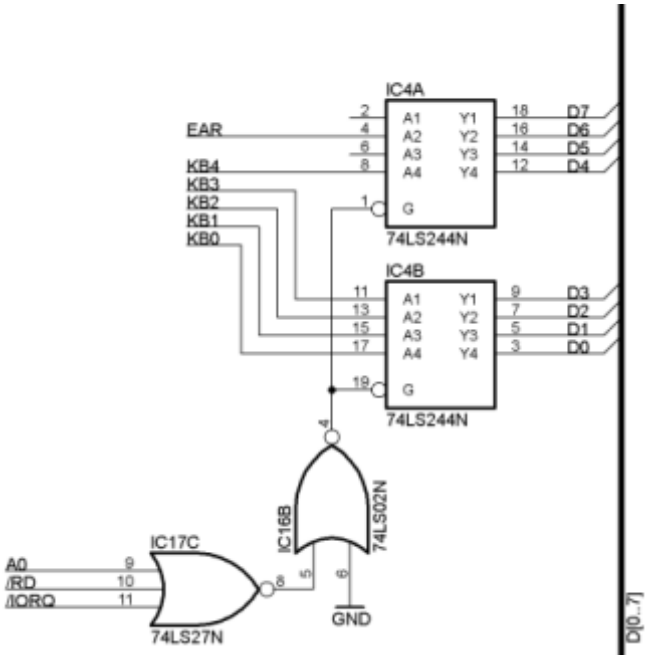

*Comparación entre los ciclos de lectura a RAM por parte del TAHC10 compartiendo slots de acceso con el Z80 (arriba) y sin compartirlos (abajo)*

Un clon de Spectrum sin contienda en memoria es algo que muchos de nosotros no veríamos hasta la llegada de los clones rusos. La implementación final de Amstrad con el +2A y el +3 mejoraron algo la contienda al eliminarla de la E/S, pero no de la memoria. Quizás se piense que no hay demasiada mejora de velocidad, pero miren si no esta comparativa con dos juegos: Fred y La Pulga, en donde se hacen muchos accesos a memoria a causa de que ambos usan scroll de pantalla casi completa y en varias direcciones. La comparativa es entre un ZX Spectrum 48K y un Inves Spectrum+ : https://www.youtube.com/watch?v=RG9ucaFKsdY

## Implementación de los puertos de E/S internos

El Inves Spectrum+ implementa dos puertos de E/S de lectura y uno de escritura. Los dos puertos de lectura curiosamente los implementa con electrónica discreta, mientras que el puerto de escritura está integrado en el TAHC10.
Puerto $FE en lectura. Está implementado usando

algunas puertas lógicas para decodificar la condición /IORQ=0 , /RD=0 , A0=0, más un buffer triestado para volcar la información del puerto al Z80. La parte del esquemático del Inves que se encarga de esta tarea se detalla en la siguiente figura:




Implementación del puerto de lectura $FE (cassette y teclado)

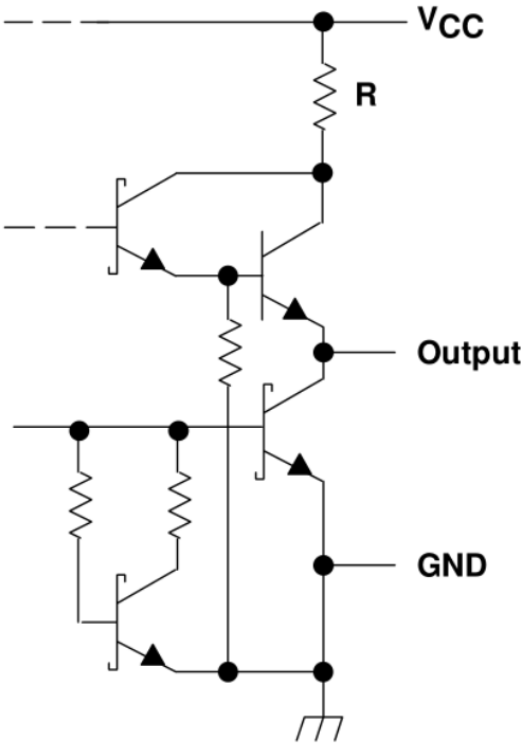
IC17 se emplea como decodificador para la condición anterior. IC16 se usa como inversor de forma que en el control de habilitación de triestado de IC4 tenemos la señal de habilitado solamente cuando se accede en lectura al puerto de E/S $FE. De hecho, cuando se acceda en lectura a cualquier puerto de E/S que tenga una dirección par. Este comportamiento es por supuesto idéntico al del ZX Spectrum original.

Las entradas al buffer son los 5 bits del valor de columna del teclado, provenientes del conector de 5 pines enchufado a la membrana. Ocupan las posiciones D0 a D4. D6 viene de la entrada EAR. En el esquemático, esa parte de la circuitería está aún sin documentar. Suponemos que hay un paso amplificador y escuadrador de señal. Los bits D7 y D5 también están sin documentar, aunque probablemente estén fijados a +5V para que se lea como un 1 lógico.

Al estar implementado este puerto fuera del TAHC10 no se emplea su bus de datos interno (recordemos que éste está conectado solamente a la RAM), así que un acceso al mismo no genera contienda de ningún tipo.

IC4 está conectado directamente al bus de datos del Z80, y su salida, cuando está habilitada, es totem-pole. Esto plantea un pequeño problema de incompatibilidad en periféricos externos que pretendan sustituir o emular al teclado: en el ZX Spectrum, la información de tecla pulsada la suministra la propia ULA mediante un acoplo débil del bus de datos mediante resistencias. Esto hace posible que un periférico externo que responda al mismo puerto de E/S tenga preferencia sobre la ULA. Así es como funcionan periféricos tales como el Interface 2, el interface de joystick programable por clavijas de COMCOM, cualquier otro interfaz de joystick programable, o cualquier periférico que pretenda sustituir al teclado.

En el Inves no hay acoplo débil sino fuerte: cualquier periférico que pretenda poner un dato en el bus como respuesta a la lectura del puerto $FE se encontrará de cara con que el LS244 también pone un dato. La etapa de salida del buffer tiene este aspecto:



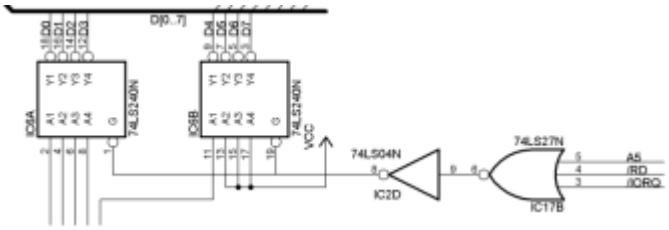

**'LS240. 'LS241, 'LS244: R = 50 Ω NOM**
**'S240, 'S241, S244: R = 25 Ω NOM**

Etapa de salida de un 74LS244

Lo que significa que cuando el LS244 está poniendo un 1 en el bus (tecla no pulsada), lo hace poniendo 5V a través de una resistencia de 50 ohmios. Si un periférico externo quiere poner un 0 (tecla pulsada), debe vencer ese pullup fuerte ofreciendo una resistencia más baja, y forzando al LS244 a que circulen hasta 100mA por uno de sus transistores de

salida. El datasheet especifica que en caso de cortocircuito de alguna de las salidas, debe respetarse que no se cortocircuite más de una salida a la vez, y que la situación de cortocircuito no exceda de 1 segundo. En la práctica, que un periférico sea capaz o no de “vencer” al LS244 dependerá de cómo tenga implementada su etapa de salida, y en particular, la impedancia que ofrezca al poner un 0 en el bus. Un ciclo de bus de E/S tiene una duración de 4 ciclos de reloj, de los cuales en 3 de ellos el buffer está habilitado. Esto supone una duración máxima para la condición de cortocircuito de unos 845,8 nanosegundos en cada ciclo de E/S.

**Puerto $DF en lectura.** Es el puerto de acceso a la interfaz de joystick Kempston. Está implementado con un buffer triestado inversor IC6 (74LS240), más unas puertas lógicas para decodificar la condición de lectura del puerto $DF. Esta es la parte de esquema donde se implementa:




Implementación del puerto de joystick Kempston

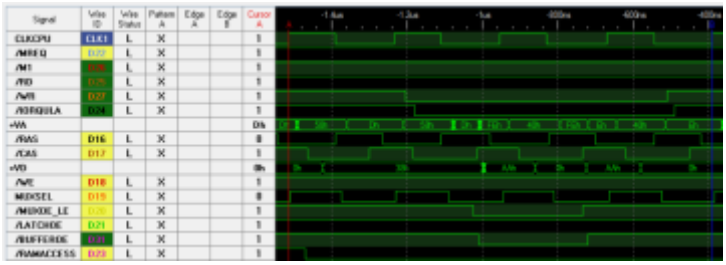
IC17B e IC2D forman entre las dos una puerta OR de 3 entradas, que decodifica la condición de acceso a la interfaz Kempston habilitando en su caso el triestado del buffer. Las entradas al buffer vienen de los 4 pines de direcciones del joystick, más el pin de disparo. Para activar uno de estos pines hay que cortocircuitarlo a GND para que en lectura se lea como un 1. Esto es lo que se hace en el interior del joystick, en donde los pulsadores que se accionan con la palanca o el botón de disparo cortocircuitan momentáneamente el pin correspondiente al pin común de señales del conector de joystick, que aquí está fijado a GND. El resto de pines no usados se fuerzan a +5V para que en lectura se lean como 0’s. Al estar este puerto implementado con lógica discreta y conectado directamente al bus de datos, no presenta problemas de contienda con el TAHC10. Asimismo, y como en el caso del puerto $FE en lectura, la decodificación incluye la señal de lectura /RD por lo que no vuelca datos durante un ciclo de aceptación de interrupción (no corrompe el bus de datos durante una interrupción).

**Puerto $FE, en escritura.** Este puerto está integrado dentro del TAHC10 por razones obvias: el color del borde y el altavoz se gestionan desde este chip. Dado que el TAHC sólo tiene un bus de datos y éste está aislado del bus de datos de la CPU, es lógico suponer que los bufferes y latches que actuan en los accesos a la RAM también sean necesarios aquí.

Como ya estábamos alertados por el estudio que había hecho César, el test que se creó para probar las escrituras a este puerto se diseñó de la siguiente manera:

Primero el procesador escribe ciertos valores en RAM. En concreto, en la dirección $40FE escribe el valor $00. En la dirección $41FE escribe $F0 y en la dirección $42FE escribe $FF. Según las pruebas de César, el valor que se escribirá en el puerto será el resultado de aplicar la operación AND de ese valor y el que estuviera escrito en memoria en la misma dirección que la que se usa como puerto.

Después de esto, y con las interrupciones deshabilitadas, se escribe sin cesar el mismo dato ($AA) a los puertos de E/S $40FE, $41FE y $42FE. El dato que veamos en el bus de datos VD será el mismo que vea el TAHC10. Nuestro objetivo es comprobar si, como sospechamos, la memoria es accedida a la vez que el puerto de E/S correspondiente.

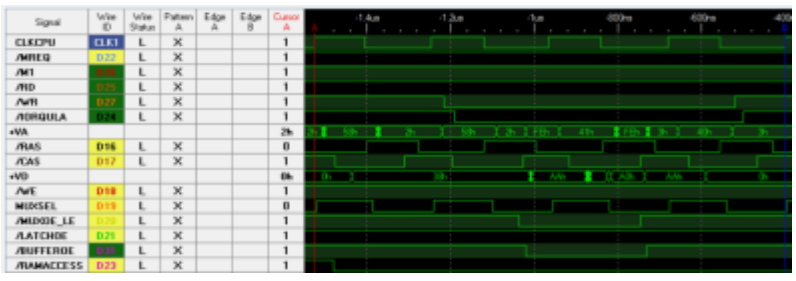


Este es el cronograma del acceso al puerto $40FE. En este caso, recordemos, la dirección de memoria $40FE tiene el valor $00.

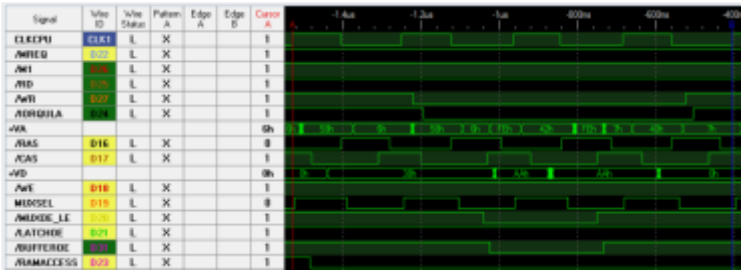
Cronograma de un ciclo de bus de lectura E/S al puerto $40FE. En la dirección de memoria $40FE se ha almacenado previamente el valor $00

La señal IORQULA se activa como resultado de haberse activado /IORQ y tener A0=0. El TAHC10 observa que /RD no ha bajado así que infiere que esto es un acceso de escritura al puerto $xxFE y se prepara para recibir el dato desde la CPU activando la señal /BUFFEROE. En el momento en que /BUFFEROE baja, ya se ve en el bus VD el valor $AA que la CPU pretende escribir al TAHC10. Mientras tanto, MUXSEL y /MUXOE también se activan, así que se produce un ciclo de lectura de memoria RAM en la misma dirección que se usa como puerto, como resultado del cual se consigue que también aparezca en el bus VD el valor que tiene la RAM en esa posición. En ese momento, unos nanosegundos después de bajar /CAS, en el bus VD

hay una mezcla del dato que viene desde la CPU y que es provisto por IC15, y el dato que viene de la RAM, provisto por ella misma.
Ya hemos visto qué tipo de salidas tiene IC15, un 74LS244, con un pullup de 50 ohmios y transistores bipolares. El chip de RAM TMS4464 es de tipo SMOS (MOS escalable canal N), lo que significa que su etapa de salida son transistores MOS con un pullup débil (un transistor NMOS en modo enhanced actuando como una resistencia), pero un pulldown (otro transistor NMOS) muy fuerte, casi un cortocircuito perfecto, mejor que el cortocircuito que pueda hacer a GND un transistor bipolar. En la práctica esto significa que un 0 que venga desde una línea de datos de la RAM "gana" a un 1 que venga de la línea de datos del buffer, y que un 1 que venga de la línea de datos de la RAM "pierde" frente a un 0 que venga de la línea de datos del buffer. La tabla de verdad de las señales que vienen de la RAM y del buffer es de hecho, la tabla de una operación AND. Esto significa que mientras que la RAM está dando el dato $00 (todos los bits a 0), este dato "gana" al anterior $AA, y así se muestra en el contenido del bus de datos VD. En cuanto a lo que lee el TAHC10, pensamos que abre su latch interno y lo deja en modo transparente durante el tiempo que está activo /MUXOE. La primera mitad de ese tiempo, alrededor de un semiciclo de reloj de CPU, el latch recibe el valor real que la CPU pretende escribir en el TAHC, y durante ese tiempo tanto la salida de MIC/SPK como el color del borde se actualizan correctamente. En la segunda mitad la RAM ya ha volcado su dato al bus VD y se combina con el dato que proviene de la CPU. /MUXOE se desactiva y este último valor combinado presente en VD es lo que ve en definitiva el TAHC10. El efecto es que durante aproximadamente un ciclo de reloj de pixel, el borde cambia de color al color correcto, para pasar después al color resultante de la combinación del valor de la CPU con el valor de la RAM. El sonido, de haberlo, es un pulso que dura unos 140 nanosegundos (medio ciclo de reloj).
En el acceso al puerto $41FE nos encontramos esta otra situación:



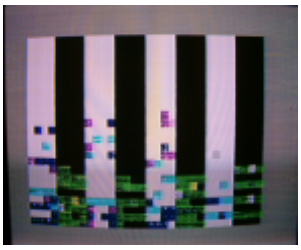
Cronograma de un ciclo de bus de lectura E/S al puerto $41FE. En la dirección de memoria $41FE se ha almacenado previamente el valor $F0
El valor leído de la RAM es $F0, que combinado con el valor que se pretende escribir, $AA, hace que el TAHC vea el valor $A0.
Por último, al acceder al puerto $42FE, el dato que enmascara al valor procedente de la CPU es $FF, lo que resulta en un dato sin alterar, tal como se observa en el cronograma:

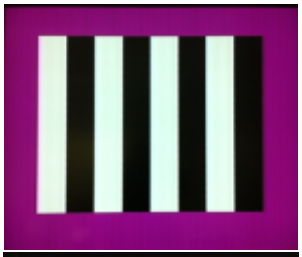


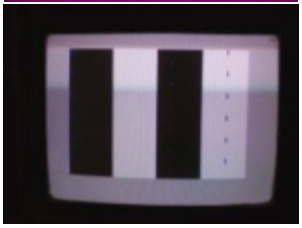
Cronograma de un ciclo de bus de lectura E/S al puerto $42FE. En la dirección de memoria $42FE se ha almacenado previamente el valor $FF

Hay un momento, dura unos nanosegundos, cuando después de que se active /CAS, la RAM actualiza su buffer interno para mostrar el valor pedido, en el que los glitches que se generan en sus lineas de datos hacen variar el valor que se muestra en VD, pero una vez que se estabiliza, el dato que vuelca, $FF, se combina con $AA para producir $AA, que es lo que ve el TAHC10.
Esta es la razón por la que los POKE's a la ROM provocan este comportamiento en el puerto $FE. De hecho y como se ha visto, que el POKE sea a ROM o a RAM es indiferente. Funciona también con ésta. La razón de por qué basta con pokear a direcciones de ROM estriba en que en la ROM del Inves los accesos de escritura al puerto $FE se hacen con la instrucción OUT ($FE),A. Esta instrucción pone en la parte baja del bus de direcciones el valor $FE logicamente, pero pone en la parte alta el valor que haya en A en ese momento. Dado que la ROM escribe en A un valor tal que en binario es 000SMBBB (S=speaker, M=mic, B=border), sabemos que dicho valor no será mayor de 31 ($1F), así que pokeando un 0 en todas las direcciones de memoria en las que la parte baja valga siempre $FE y la parte alta varíe desde $00 a $1F, obtenemos el efecto de color de borde forzado a negro, y silencio absoluto en el altavoz, cuando estamos en BASIC. Si desde él emitimos una instrucción tal como OUT 32*256+254,7, volveremos a ver el borde blanco. Si hacemos POKE 32*256+254,0 , el OUT anterior dejará de funcionar. De hecho, y dado que en un Inves recién arrancado, la gran mayoría de posiciones de memoria está a 0, casi

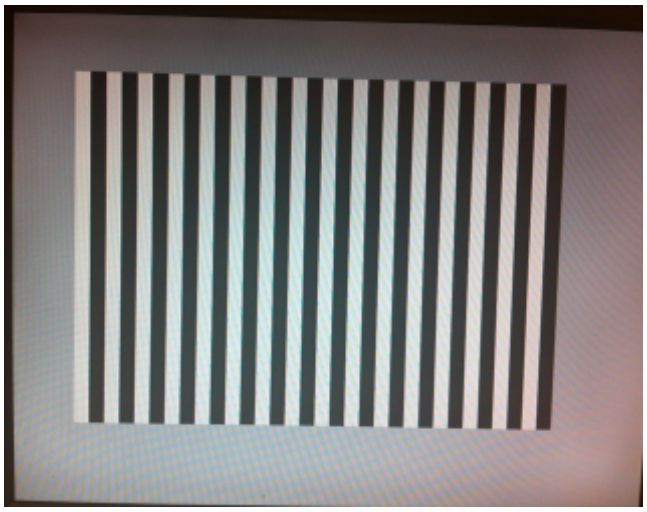
ningún OUT que pretenda escribir al puerto $FE usando en la parte alta un valor mayor de $3F funcionará.
Un comando NEW, RANDOMIZE USR 0, o el botón de reset, hacen que se vuelva a ejecutar la rutina de inicialización de la memoria, que es idéntica a la del ZX Spectrum. Esta rutina no toca los valores en ROM porque se supone que en la ROM no se puede escribir, por tanto no tiene sentido chequearla. Así, cualquier valor que se pokeara en ROM, allí seguirá, impidiendo que el color del borde cambie o el sonido funcione.
La pregunta es lógica: si para cualquier acceso de escritura al puerto $FE sucede esta combinación AND con el contenido de la memoria, ¿cómo es que un Inves recién encendido no muestra comportamientos extraños? Después de todo, el contenido de la memoria es aleatorio en el momento del encendido. ¿O no?
Cuando se enciende un Spectrum, sea de Sinclair, Amstrad o un Inves, si ya tenemos sintonizado el canal para que se vea la imagen desde el primer momento, y sobre todo si el equipo lleva apagado varios minutos, se puede ver durante una fracción de segundo una imagen característica, de franjas verticales, a veces negras y blancas, a veces de otros colores. Si el ordenador está estropeado y no puede iniciarse, el patrón queda fijo en pantalla y todos lo interpretamos como un Spectrum averiado. Un Spectrum “sano” sólo mostrará esta imagen por un momento, para seguidamente poner a negro toda la zona de paper, y a continuación, volver a borrarlo dejando el fondo blanco y en la parte inferior el conocido mensaje de copyright.







Patrones de imagen inicial del Spectrum hallados en varias webs, entre ellas, la página web de José Leandro Novellón (imagen de la derecha)
Según el fabricante y tipo de memoria que lleve, el patrón de franjas verticales es diferente. Pueden ser franjas muy anchas, de unos 8 caracteres de ancho como ocurre con algunos modelos +2A/3, o pueden ser franjas de unos 4 caracteres de ancho, como ocurre en la mayoría de Spectrum’s de 16K y 48K.
Esas franjas que se ven no son más que la interpretación en forma de imagen de los valores iniciales que contienen las celdillas de memoria. En una memoria DRAM, al contrario que las SRAM, quien guarda el valor de un bit es la capacitancia residual que se origina al fabricar un transistor MOS. La carga almacenada es muy pequeña y necesita ser amplificada y refrescada cada cierto tiempo.
Aquí no he encontrado información adicional, pero especulo con que bits que se guarden en posiciones adyacentes lo hagan con la polaridad invertida unos de otros, de forma que guardar un 1 lógico no siempre signifique cargar el condensador, y guardar un 0 no siempre signifique descargarlo. Como en muchas ocasiones hay grandes regiones de memoria con el mismo valor, estos cambios de polaridad podrían mejorar la inmunidad al ruido del propio chip.
Así, lo habitual es que tras un periodo prolongado con el equipo apagado, todos los condensadores que almacenan bits estén descargados, pero como el valor que se interpreta por los amplificadores es distinto según dónde esté ubicado el condensador, el resultado neto es que la memoria vuelca datos en forma de patrón regular de 1’s y 0’s.
El Inves monta casi todos los chips de Texas Instruments. Los chips de memoria también lo son, y su patrón de bits sin inicializar es del tipo 1111 0000 1111 0000 …. . Es decir: la dirección $0000 contiene un valor inicial que se interpreta como $F (los 4 bits a 1). La dirección $0001, el valor $0, y así sucesivamente. Al tener dos memorias del mismo fabricante, la secuencia que tenemos es $FF, $00, $FF, $00, etc. En la zona de memoria que guarda la información de la pantalla, esa secuencia hace que se vea el siguiente patrón al encender un Inves:

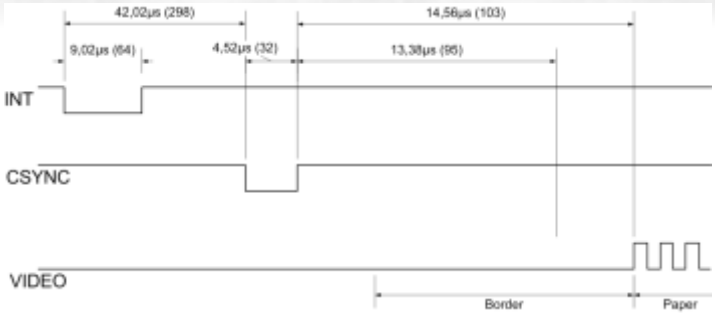


Patrón de valores iniciales de la RAM del Inves cuando aún no ha comenzado a ejecutarse la ROM

Y lo curioso de este patrón es que en las direcciones de memoria par guarda el valor $FF. Dado que el acceso al puerto $FE en escritura es un acceso a un puerto par, y dado que en practicamente todos los casos, la parte alta de la dirección de puerto será un valor comprendido entre $00 y $1F, la dirección de memoria que entra en conlicto con el puerto pertenece al espacio de ROM, a donde se espera que no se haga ninguna escritura. Es por esto que este fallo de diseño podría haber pasado inadvertido.
Claro está que si un programa realiza escrituras a ROM (por ejemplo como parte de una protección de carga para obtener un bloque de carga superior a lo que un copión de la época puede manejar) o bien por un efecto secundario de alguna rutina (por ejemplo para hacer un scroll vertical y no tener que chequear si la última fila se copia en ROM), hay posibilidad de que alguna de las direcciones de memoria “sensibles” ($xxFE) resulte alterada, creando el pequeño desastre en el bus VD que hemos visto.
Por otra parte, si se avería un Inves Spectrum y al repararlo se cambian los chips de memoria por otros que no cumplan con ese patrón de datos iniciales, es muy posible que desde el mismo momento del arranque, el color del borde no funcione como debiera, o el equipo se quede mudo.

## Interrupciones

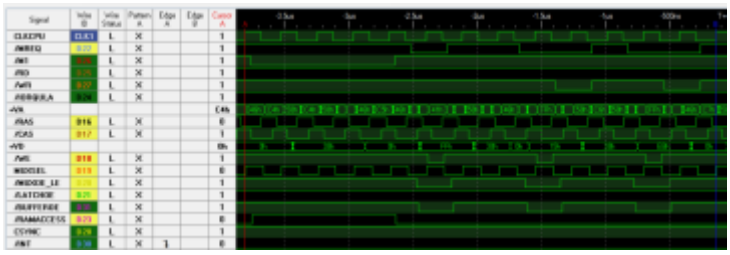
En el artículo de Oscar García Reyes ya se daba información sobre la diferente temporización de la interrupción del retrazo vertical. En el caso del Sinclair ZX Spectrum, la ULA dispara la interrupción 14336 ciclos de reloj de CPU antes de que comience a pintarse la primera línea de píxeles de “paper” en la pantalla. Durante todo ese tiempo, poco más de 4 milisegundos, la ULA no accede a la memoria de pantalla por lo que no existe contienda.
Chris Smith, en su libro “The ZX Spectrum ULA” describe las ecuaciones que dan lugar al pulso de interrupción, y su relación con los contadores de píxeles horizontal y vertical. En el caso del Inves, y aunque parece heredar algunas características de temporización del Spectrum 128K, el pulso de interrupción se produce en un momento muy raro: 212 ciclos de reloj antes de que el TAHC10 comience a pintar la primera línea de píxeles.
El valor de 212 ciclos se ha hallado usando el analizador lógico. Dado que la cantidad de tiempo a capturar era mucha no pude usar una resolución demasiado elevada. El resultado es que los valores en microsegundos no tienen la precisión que quisiera, y las cifras que están entre paréntesis pueden variar en +/- 1 ciclo. El cronograma al que se ha llegado es éste:




Cronograma de la señal de INT y su relación con la generación de video. Las trazas no están a escala.

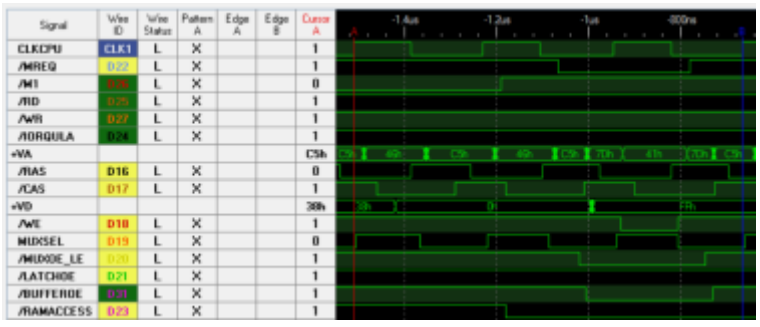
Los tiempos están dados en microsegundos, y entre paréntesis, en ciclos de reloj de pixel (ver más adelante la sección sobre la temporización de la pantalla). Si sumamos el tiempo en ciclos desde el flanco de bajada de INT hasta el momento en que se lee el primer byte de la memoria de pantalla (cota de 13,38 microsegundos), obtenemos 298+32+95 = 425 ciclos de pixel. En ciclos de reloj de CPU es la mitad: 212,5 ciclos.
A día de hoy no tengo explicación de por qué se eligió este momento en concreto. No parece haber una relación sencilla entre los valores de los contadores horizontal y vertical y el momento de la interrupción.
Lo que sí es más curioso, y que de hecho considero uno de los fallos de diseño más graves del Inves, es lo que ocurre durante el ciclo de aceptación de interrupción:
El Z80, en respuesta a una petición de interrupción enmascarable, y si las interrupciones no están deshabilitadas, realiza lo que se denomina un ciclo INTA (INTerrupt Acknowledge). Durante ese ciclo de bus, que puede durar más de una decena de ciclos de reloj, se realizan varias labores, siendo una de ellas la de guardar la dirección del programa que está siendo interrumpido, para actualizar el registro PC con la dirección de la rutina de interrupciones y saltar a ella. Esto significa realizar dos escrituras a memoria, con la parte alta y baja de la dirección de retorno.
En el modo de interrupciones IM 2 se tiene además una tabla de vectores de interrupción. La dirección base de la tabla es de la forma $xx00 donde $xx es el valor del registro I. En el Spectrum, la parte menos significativa de la dirección la pone el dato que haya en el bus de datos en ese momento, que debería ser $FF. A causa de que algunos periféricos “ensucian” el bus de datos poniendo cualquier dato durante el momento en el que el ciclo de INTA lee el valor del vector de interrupción, los programadores han optado casi desde el principio por usar una tabla de 257 valores, todos ellos iguales. César ya ha hablado de esto, así que ya sabeis que en el Inves usar la ROM

como tabla no es factible. Sólo puede usarse, si se quiere, una tabla en RAM. Y si no se quiere, puede suponerse que el vector siempre será $FF y por tanto sólo basta con poner la dirección de la rutina de interrupción en I*256+255, liberando así 255 bytes de memoria. Lo malo es que los juegos que optan por esa vía, y en los que esos 255 bytes contienen información importante para el programa, se bloquearán sin remedio. Esto es lo que ocurre:



Cronograma de un ciclo de INTA

Entre los cursores A y B se ha enmarcado el ciclo completo. La señal /INT ha bajado unos dos ciclos de reloj antes, y sólo cuando se ha terminado la instrucción en curso ha comenzado el ciclo de INTA. Para notificarlo, el Z80 baja la línea /M1 y la línea /IORQ. Aquí no tenemos sondada la línea /IORQ, y la línea /IORQULA es producto de varias entradas, no solamente /IORQ así que no se activa.
Recordando lo que hemos dicho hace unos párrafos, sobre que en el ciclo de INTA se realizan dos escrituras a memoria para guardar la dirección de retorno, podemos comprobar que están en el cronograma. O no. Un momento: no hay dos escrituras: ¡hay tres! Las dos últimas sí son escrituras reales: el TAHC10 se ha adelantado con éxito activando /WE antes de que el Z80 active /WR. Sin embargo, el primer pulso de /WE en el cronograma sucede sin que el Z80 haya emitido un ciclo de bus de escritura en memoria. ¿Qué ha pasado?




Cronograma de la primera parte del ciclo de INTA

En el cronograma se observa el último ciclo de reloj del ciclo M1 especial. En este ciclo no se activa ni /MREQ ni /RD, por lo que el TAHC10 ignora este ciclo y no lo tiene en cuenta, como sí hacía con el ciclo M1 ordinario. A causa de esto, el ciclo de refresco, que normalmente pasa ignorado para el TAHC, aquí se confunde con un ciclo de escritura en RAM: /MREQ está activa, /RD no lo está, y no hay constancia de un ciclo M1 anterior, así que el TAHC10 pone en marcha su maquinaria adivnatoria de ciclos de escritura en RAM… sólo que esta vez falla.

El ciclo de refresco, que es lo que tenemos aquí en realidad, pone en el bus de direcciones el valor de los registros I y R (parte alta y parte baja respectivamente). El TAHC10 piensa que esa es la dirección de escritura y realiza un ciclo RAS-WE-CAS. El valor que se escribe en el bus es el que hay en el bus de datos en ese momento, y como la CPU realmente no está poniendo ningún dato, lo que se lee y aparece como dato de la memoria es el bus "idle", o sea, $FF. En la figura, I=$41 y R=$7D. La operación de escritura hace que en la dirección $417D se escriba el valor $FF.
Este comportamiento es nefasto para la tabla de interrupciones en RAM. En cada interrupción, un byte de la tabla de interrupciones, aquel que corresponda al valor actual del registro R, será sobreescrito con el valor $FF. Eso siempre que no haya ningún periférico en el bus de expansión metiendo un valor diferente: en ese caso el valor que se escribe es indeterminado.
El registro R es de 8 bits, pero el Z80 sólo incrementa los 7 menos significativos. El bit más significativo puede cambiarse por software y su valor no se modifica en los ciclos de refresco. Por defecto vale 0, lo que significa que el efecto de sobreescritura automática de la tabla se produce sólo para los valores de R de $00 a $7F: la mitad de la tabla. La otra mitad, siempre y cuando no se haya modificado el bit 7 de R, permanecerá inalterada.

Como esto ocurre en el ciclo de refresco que se produce como parte del ciclo de INTA, no importa que no se esté en el modo IM 2. El efecto se producirá en cualquiera de los otros modos de interrupción. Este sencillo programa simplemente hace que I apunte al principio de la pantalla ( LD A,64 ; LD I,A ; RET ). Automaticamente, y sin que hagamos nada, empezaremos a ver cómo algunas líneas del tercio superior de la pantalla toman el valor $FF apareciendo rayas negras.

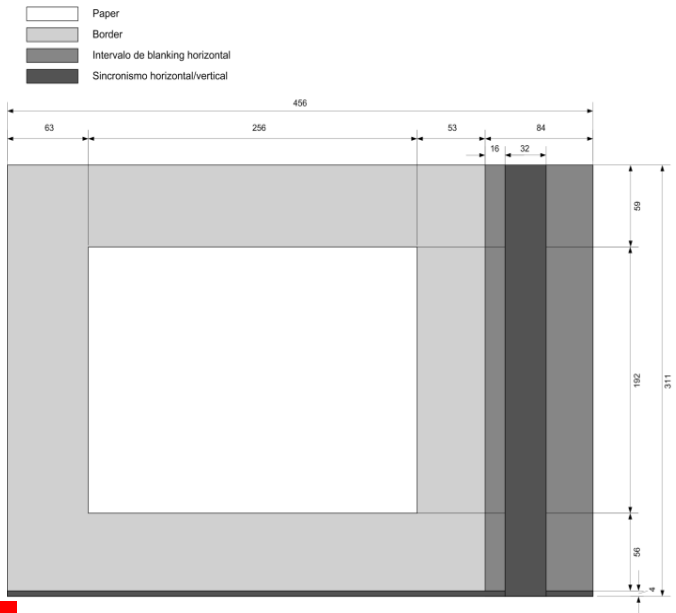
```
10 DATA 62,64,237,71,201
20 FOR n=23296 TO 23300: READ a: POKE n,a: NEXT n
30 RANDOMIZE USR 23296
```

Puede verse el efecto en este video:
https://www.youtube.com/watch?v=HiNAs0sQbI4
No deja de ser curioso que este mismo programa, en

un ZX Spectrum 48K, produzca el famoso efecto “ULA snow”.

## Temporización de la pantalla

El valor base con el que contamos para cualquier temporización en el Inves Spectrum+ es la frecuencia del reloj maestro: 17,7345MHz. Es la misma frecuencia que se usa en el Spectrum 128K y en el +2 gris. Del análisis que hizo Chris Smith, sabemos que la frecuencia de la CPU se obtiene dividiendo entre 5 esta frecuencia. La frecuencia de pixel (la cantidad de píxeles que se pintan por segundo) es el doble de la frecuencia de reloj de CPU. Esto nos da un periodo de pixel (el tiempo que se tarda en pintar un pixel) de 140,968 nanosegundos. Este periodo es también el periodo de los contadores internos del TAHC10 que determinan cuándo se pinta el borde, el paper, o hay señal de sincronismo.
Por otra parte, la duración de una línea de video en PAL es de 64000 nanosegundos. Una sencilla división nos da el número de cuentas por línea: 64000/140,968 ?  454. En el Inves, sin embargo, se hizo lo mismo que con el Spectrum 128K, y el número de cuentas es de 456. La razón la explica Chris en su libro en el capítulo 24, página 253, y fundamentalmente es para simplificar el diseño de algunas celdas de la ULA. En el Inves es probable que no se necesitara esa simplificación dado que la arquitectura del TAHC10 es radicamente diferente a la ULA de Ferranti, pero aun así, copiaron los valores del 128K. Esto significa que también se tienen 311 líneas por cuadro.
Tomando estos valores: 456 cuentas horizontales por línea, y 311 líneas (cuentas verticales) por cuadro, y tirando de osciloscopio digital, se ha llegado a la siguiente disposición de pantalla:




Cronograma de la temporización de un cuadro de video en el Inves Spectrum+

Todos los valores están dados en ciclos de reloj de pixel. Un ciclo de reloj de pixel, recordemos, dura 2,5/17,7345 = 140,968 nanosegundos.

## Posibles mods

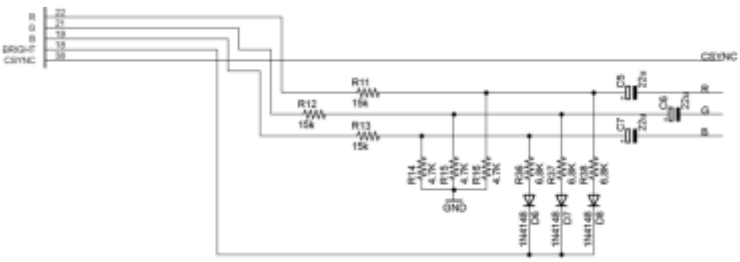
A continuación se describen algunas propuestas de modificaciones, algunas para mejorar la calidad de la salida de video, otras para tratar de corregir algunos de los fallos de diseño. Estas propuestas, sin embargo, no pretenden ser las mejores o las más fáciles de implementar, sino servir de guía a otras que sean más elaboradas. Por último debo añadir que no he probado aún ninguno de estos mods (salvo el de video compuesto) en hardware real.

### Salida de video compuesto por el conector de antena

Este mod es el más sencillo, y de hecho es calcado al que se usa en el ZX Spectrum. Me refiero a la versión con transistor y resistencia, aunque a juzgar por las características del codificador MC1377, bastaría con una conexión directa desde el pin 9 del MC1377 (IC3) a una resistencia de 75 ohmios, y de ahí a la salida de video compuesto, desconectando antes la entrada de video al modulador.

### Salida de video RGB

El TAHC10 genera una señal RGB-TTL, idéntica en niveles de tensión a que se genera en un ZX Spectrum 128K. Se puede optar por tanto por sacar una salida RGB digital y con un cable a euroconector como el que se usa para este modelo, disfrutar de una señal RGB.
Si se pretende usar RGB analógico, hay dos opciones: o bien tomar la señal RGB de las entradas del MC1377. Rojo en el pin 3, verde en el pin 4, azul en el pin 5, y sincronismos en el pin 2, o bien calcar el circuito de adaptación de RGB-TTL a RGB analógico que hay en el propio Inves y que se muestra a continuación. Las entradas a este circuito serían las señales RGB TTL generadas por el TAHC10, dibujado a la izquierda.



Si se opta por usar directamente las señales RGB TTL es conveniente intercalar un buffer para aislar al TAHC10 de la salida RGB. Un chip 74LS07 vale para esto. Si se hace así, añádanse las tres resistencias de 1K que conectan cada uno de los 3 canales a la señal de sincronismo. Esto es para bajar el nivel de esas señales durante un pulso de sincronismo vertical u horizontal y mejorar la estabilidad de la imagen y el balance de blancos.
Para calcar el circuito que usa el Inves para adaptar las señales RGB al MC1377 ha de tenerse en cuenta que el MC1377 requiere entradas RGB de 1V cada una y que su impedancia de entrada es de unos 10K. Por otra parte, las entradas RGB de un euroconector necesitan un nivel máximo de 0,7V para una impedancia de 75 ohmios. En el Inves, cada señal R,G ó B alcanza un nivel de 1,19V cuando es un color con brillo, y de unos 0,78V para el caso de un color sin brillo. El ratio brillo/no brillo es por tanto de un 65,55%. Si queremos conservar este ratio, y el máximo nivel de señal permitida para el euroconector son 0,7V , el nivel de señal para un color sin brillo debe ser de unos 0,458V.
Probablemente, lo más sencillo sea implementar alguno de los circuitos de Paul Farrow para el Spectrum 128K, dado que su ULA genera el mismo tipo de señales que el Inves.
http://www.fruitcake.plus.com/Sinclair/Spectrum128/SCARTCable/Spectrum128SCARTCableSpanish.htm

**Mod para mejorar la compatibilidad con interfaces de joystick y teclado**

Para evitar tener que "luchar" contra el 74LS244 a la hora de poner un 0 en el bus de datos, se puede usar un 74AS757, que es compatible pin a pin con el 74LS244. El AS757 tiene salidas en colector abierto, así que en cuando este buffer pone un 1, en realidad se está desconectando del bus. Si no hay ningún interface externo, el Z80 leerá el estado de bus "idle" para ese bit, que es 1.

**Modo all-RAM en el Inves**

Ya hemos visto que un POKE en el área de ROM realmente escribe en la RAM que está debajo, y también que cuando leemos un dato de memoria en el área de ROM, el TAHC10 evita que el dato que se lee de la memoria llegue al bus de datos del Z80.
Si volvemos a repasar el cronograma de la lectura desde RAM y lo comparamos con el de la lectura desde ROM, veremos que en ambos casos se recoge el dato de la RAM, pero en el caso de la lectura de ROM no se activa la señal /BUFFEROE.
Sospechamos que esto ocurre porque la señal /RAMACCESS no se ha activado. Esta señal indica al TAHC10 que el ciclo de acceso a memoria es a memoria RAM y es la única señal de entrada al TAHC10 que cambia en ambos cronogramas.
Así, si se cambia esa entrada para que siempre valga 0, el TAHC10 siempre habilitará la señal /BUFFEROE en una operación de lectura. A la vez, hay que evitar que la ROM se active cuando hay un acceso al rango $0000 - $3FFF. Esto se puede hacer fácilmente forzando un nivel alto en la señal ROMCS, que es el lado de la resistencia R54 conectado al pin 22 de la EPROM.

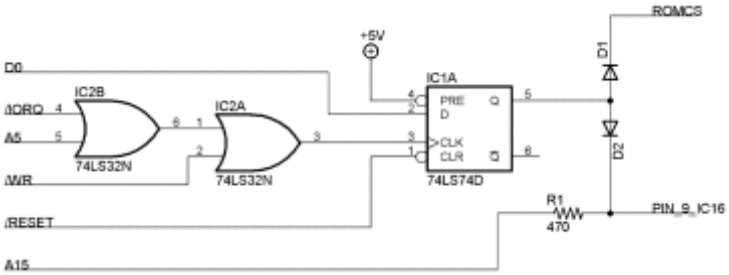
El mod por tanto podría implementarse de la siguiente forma:
•Cortar la conexión entre el pin 9 de IC16 y el pin A15 del Z80, y restaurarla de nuevo usando una resistencia de 470 ohmios. Esto nos permitirá forzar a 1 ese pin 9 sin alterar (significativamente) la señal del Z80.
•Conectar el cátodo de un diodo 1N4148 al pin 9 de IC16, y el cátodo de otro diodo 1N4148 al pin 22 de IC40. Los ánodos de ambos diodos se unen y van a parar a un borne de un interruptor. El otro borne irá conectado a 5V.
Cada vez que se accione el interruptor y se envíen 5V a esos dos diodos, tanto el pin 22 de IC40 como el pin 9 de IC16 tendrán nivel lógico 1. En IC40, la EPROM, esto hace que se desconecte del bus y no intervenga. En IC16 hace que la salida de la puerta NOR valga 0, activando permanentemente la señal /RAMACCESS, haciéndole creer al TAHC10 que el ciclo de bus actual es para la memoria RAM.
El modo de operación sería así: con el interruptor abierto, esto es, el Inves funcionando normalmente, se copiaría aquel programa que quiera usarse como ROM alternativa simplemente escribiendo en ROM. Una vez hecho esto, se pulsa RESET y se deja pulsado mientras se acciona el interruptor que activa el modo all-RAM. Al soltar RESET, el programa que se haya escrito en el espacio de ROM pasará a poder ser leído y ejecutado.

```
; Un programa que copia la ROM en sí misma (en el Inves,
; hace una copia en la RAM escondida)
ld hl,0
ld de,0
ld bc,16384
ldir
ret
```

Si se quiere tener control por software de este modo all-RAM en lugar de hacerlo con un interruptor, puede usarse un biestable, tal como un 74LS74.

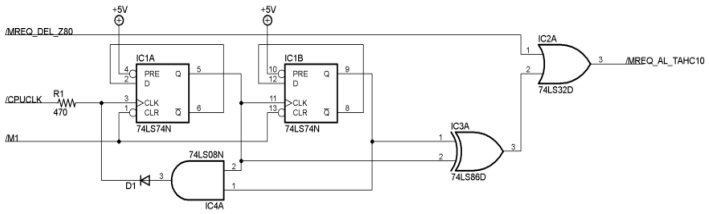
Este chip tiene dos biestables, pero sólo necesitamos uno. La entrada D del biestable que usemos se

conectaría al bit D0 del Z80. La entrada CLK se conectaría a un decodificador que se activara cuando se acceda a un puerto de E/S en escritura (el puerto tendríamos que elegirlo nosotros: un ejemplo es usar el bit A5, es decir, el puerto $DF, pero en escritura, ya que en lectura es el puerto de joystick Kempston). La salida Q se conectaría al punto de unión de los dos diodos (sus ánodos). Por último, la entrada CLR se conectaría a la señal de RESET del Z80. Si se quiere que tras un reset se siga usando la RAM como ROM, entonces sustituir la conexión de CLR a /RESET por una conexión a una célula de retardo: un condensador de 1uF desde CLR a GND, y una resistencia de 100K desde CLR hasta +5V.
Haciéndolo así, será necesario apagar y encender el Inves para restituir el uso de la ROM interna.
Para activar el modo all-RAM, una vez que la RAM que está en direcciones de ROM contenga el contenido deseado, hacer OUT 223,1 . Para volver a dar control a la ROM interna, hacer OUT 223,0
Este mod es compatible con un periférico que desactive a la ROM interna para poner la suya (ej: DivIDE, DivMMC, copiadores hardware, etc) siempre y cuando no se active el modo all-RAM.



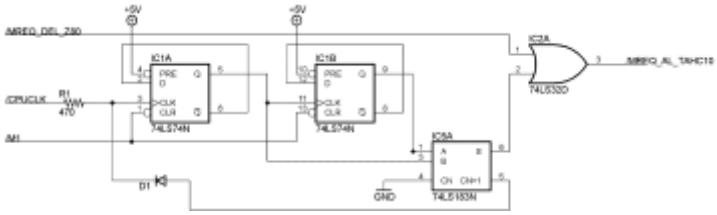

## Reparación del fallo de corrupción de la RAM durante un ciclo de INTA

El fallo ocurre porque justo después de un ciclo M1 especial dentro del ciclo de INTA hay un ciclo de refresco en el que /MREQ se activa. Este ciclo M1 especial no activa ni /MREQ ni /RD, con lo que el TAHC10 se confunde. Una solución consiste en callar a la señal /MREQ durante dos ciclos de reloj de CPU justo después de que ocurra un flanco de subida de la señal /M1. Esto elimina la señal /MREQ del ciclo de refresco, que de todas formas no se usa para nada en el Inves. Si no hay señal /MREQ, el TAHC10 no hará un ciclo de escritura en RAM.
El circuito reparador por tanto consiste en un monoastable que se dispare con un flanco positivo de /M1, de forma que en ese momento deje su salida a 1 durante dos ciclos de reloj de CPU (medidos en su flanco negativo) para a continuación volverla a su nivel de reposo, 0. La salida del monoastable iría a una puerta OR de dos entradas, donde la otra entrada sería la señal /MREQ original. La salida de la puerta sería la señal /MREQ que vería el TAHC10. Una posible implementación tendría esta pinta:




Aquí, los dos biestables 74LS74 forman un contador de 2 bits en cascada que cuenta ciclos de reloj. El contador se resetea a 0 con la señal /M1 y sólo cuenta cuando esta señal vale 1. El biestable A contiene el bit menos significativo del contador, y el biestable B, el más significativo. La señal de reloj del biestable A viene de la versión invertida del reloj del Z80, que se puede encontrar en el pin 10 de IC19. Los dos bits del contador alimentan una puerta XOR cuya salida contiene el valor llave que permite en la puerta OR que la señal /MREQ del Z80 pase inalterada a la salida del circuito, o bien que a la salida haya un valor 1. Esta salida se conectaría a la entrada /MREQ del TAHC10.
Al comienzo de un ciclo de bus M1, tanto si es normal como especial, la señal /M1 vale 0, reseteando el contador y haciendo que su salida sea 00 todo el tiempo que /M1 siga estando a nivel bajo. En este estado, la puerta XOR envía un 0 a la puerta OR, y /MREQ pasa inalterada.
En el momento en que /M1 pasa a valer 1, el contador puede comenzar a contar. El primer flanco negativo de la señal de reloj del Z80 hace que el contador pase de 00 a 01. En ese momento la puerta XOR cambia su salida a 1, haciendo que la salida de la puerta OR genere un 1 independientemente del valor de /MREQ. Con el siguiente flanco negativo de reloj la cuenta pasa de 01 a 10, y la salida de la puerta XOR no cambia. Con el siguiente flanco, la cuenta pasa de 10 a 11 y la salida de la puerta XOR cambia a 0, dejando pasar de nuevo la señal /MREQ inalterada. Al mismo tiempo, la puerta AND que también tiene en sus entradas los valores del contador cambia su salida a 1, haciendo que el diodo conduzca y forzando un nivel 1 en la entrada de reloj del contador. Esto inhabilita dicha señal de reloj en el contador, que por tanto deja de contar, dejando las salidas a 11 hasta que vuelva a ocurrir un cambio en /M1.
Este circuito puede simplificarse un poco: tenemos por una parte una puerta XOR y otra AND, cableadas

como en un circuito semisumador, así que podemos usar un chip que implementa este elemento. El circuito integrado 74LS183 implementa dos sumadores completos, de los cuales sólo necesitamos uno. Para usarlo de la forma en que queremos, basta con poner el carry de entrada a 0. El circuito modificado vendría a ser éste:

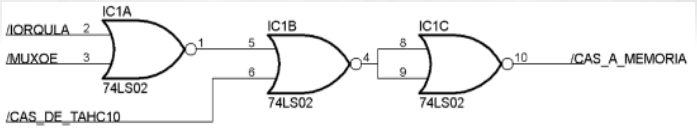



## Reparación del fallo de enmascaramiento de datos en una operación de escritura al puerto $FE

Este es un poco más truculento, aunque el circuito es más sencillo que para los otros mods. Mi propuesta consiste en enmascarar la señal /CAS que va a la RAM el tiempo justo y necesario para que el dato que viene del Z80 pueda ingresar dentro del TAHC10 sin que la RAM llegue a activarse. Digo que es truculento porque la RAM debe estar inactiva el menor tiempo posible, para evitar que se pierda el refresco de la pantalla. La RAM no entrega ningún dato al bus de datos a menos que /CAS baje, así que si evitamos eso, la RAM seguirá con su bus en alta impedancia.

Así, si nos fijamos en cualquiera de los tres cronogramas que hemos presentado con accesos de escritura al puerto $FE, vemos que hay un periodo de tiempo en el que /CAS está bajo, y que es cuando el dato de la memoria está interfiriendo con el que viene de la CPU. Al mismo tiempo que /CAS está a nivel bajo, lo está la señal /MUXOE y la señal /IORQULA. En ningún otro momento están esas tres señales activas al mismo tiempo.

Por tanto, el enmascaramiento consistiría en una puerta NOR de dos entradas: en una la señal /IORQULA y en otra la señal /MUXOE. La salida de esta NOR será 1 sólo cuando sus dos entradas sean 0. Esa salida será la máscara para /CAS : usando una puerta OR donde una de sus entradas sea la señal /CAS desde el TAHC10 y la otra entrada venga de la salida de la NOR anterior. La salida de esta nueva puerta OR iría a la señal /CAS de la RAM. En la práctica se pueden usar 3 puertas NOR de un chip 74LS02 como se ve en el esquema adjunto.




## Viabilidad de la implementación del puerto $FF (bus flotante)

Para cada scan de video de los que pertenece a la zona de paper, el TAHC10 comienza a leer la memoria de video correspondiente 9 ciclos de pixel antes de que el primero de los píxeles aparezca en pantalla. Asimismo, 9 píxeles antes de que termine la zona de paper, el TAHC10 deja de leer la memoria. En realidad esto no es así: el TAHC10 siempre está leyendo la RAM, aunque no haga nada con ella, y lo hace tanto si se está generando el borde como si es la zona de sincronismos. En la lista de cosas para hacer tengo el averiguar cuál es la secuencia exacta de direcciones de memoria que lee el TAHC10, y qué pasa cuando un scan ha terminado. Lo que sí puedo decir es que, a causa de que las lecturas no se interrumpen cuando se llega a pintar los bordes de la pantalla, es mucho más complicado implementar un mecanismo que copie el valor de lo que se está leyendo de RAM de video para que lo vea la CPU, de tal forma que cuando el TAHC10 no leyera datos la CPU obtuviera un $FF. Tanto, que la solución que se me ocurre pasa por replicar en circuitería externa parte del funcionamiento interno del TAHC10, sincronizado con éste último, y que diera a la CPU la información relevante cuando esta circuitería sepa que se está accediendo al borde o al paper. El circuito sería una máquina de estados controlada por dos contadores idénticos a los que lleva el TAHC10. Dependiendo de los valores de los contadores, sabríamos que estamos en zona de border o zona de paper, y enviaríamos a la CPU el valor $FF por el puerto $xxFF en el primer caso, y el valor que hubiera en el latch de la RAM en el segundo caso.

## Consejos para garantizar la compatibilidad del software de nuevo desarrollo (y de paso, poder arreglar algún juego existente)

Recogemos aquí tanto las indicaciones del artículo de Oscar García Reyes, como las que se derivan de los análisis que se han realizado sobre el Inves.

•No debe usarse la ROM como medio para encriptar el software, o el proceso de carga. Asimismo, no debe asumirse que la ROM tendrá ciertos contenidos, ni ciertas rutinas. Tampoco debe asumirse que tendrá un bloque con valores $FF lo suficientemente grande

como para albergar una tabla de interrupciones.

•En caso de tener que sincronizarse con las interrupciones, se ha de tener en cuenta que hay mucho menos tiempo para pintar sprites. Si no se respeta esto probablemente el juego siga siendo jugable, aunque con parpadeos. Por otra parte, al no haber contención las escrituras a memoria de video serán más rápidas que en un ZX Spectrum.

•Al no haber parón de reloj en el Z80 como resultado de la contención, los trucos que se emplean para sincronizar el programa con el barrido de pantalla, por ejemplo para conseguir que la rutina de interrupciones comience siempre en el mismo T-estado relativo al comienzo del barrido, no funcionarán.

•Para emitir sonidos por el puerto $FE, intentar usar la instrucción OUT (C),A ya que con ella se puede controlar qué valor tendrán los 8 bits más significativos. Hacer que estos 8 bits tengan siempre el mismo valor XX, y que en la posición de memoria $XXFE haya un valor $FF guardado.

•Asimismo, no usar los dos bits MIC y SPK para evitar que se silencie el altavoz. Se puede no obstante usar los dos bits para obtener más nivel en un ZX Spectrum, y usar un valor máscara en $XXFE, por ejemplo $F7, para forzar el bit de MIC a 0, y que se use sólamente SPK.

•Para poder usar rutinas de interrupción en modo IM 2 sin peligro de corrupción de memoria, generar una tabla en RAM en la posición que corresponda al valor elegido para el registro I, con todos los valores a $FF. Esto hará que el comienzo de la rutina de interrupción sea $FFFF y pueda usarse la conocida técnica que César ya ha comentado anteriormente. En este caso la corrupción de memoria en el ciclo de INTA no alterará en realidad el contenido de la tabla, pues lo que se escribe es precisamente el valor $FF.

•En el Inves no hay puerto $FF para examinar qué está haciendo el TAHC10, y lo malo es que, a diferencia del +2A/3, es practicamente imposible implementarlo por la forma en la que el TAHC10 lee la memoria, así que evitar siempre usar este puerto para sincronizarse con la pantalla.

## Conclusiones

Después de haber estado estos dos últimos años trasteando con el Inves (César lleva mucho más tiempo), la sensación es un poco agridulce. Es una máquina que da la sensación de que se ha hecho con una buena idea sobre el papel, pero una implementación con prisas que no ha dado tiempo a testear la máquina. No sabemos si el diseño del circuito interno del TAHC10 fue obra de ingenieros españoles, o si se hizo “outsourcing” a alguna empresa externa. Por un momento hasta he sospechado que el diseño fuera oriundo de la URSS, ya que, como apuntaba César, el Inves comparte alguno de los pintorescos comportamientos que se dan en alguno de los clones rusos: interrupción en un momento diferente que el ZX Spectrum, y ruidos o glitches en la salida de video (de esto hablaremos con detalle en otra ocasión).
Quien quiera que haya diseñado la circuitería del Inves se ha guiado por otros parámetros que no han sido el asegurarse la compatibilidad al 100%. Es como estos primeros emuladores que no emulaban efectos en el borde correctamente, o los efectos multicolor en el paper: la mayoría de los juegos iban bien y no se notaban esas “licencias” que se había dado el autor. Era, sencillamente, un trabajo de replicar el comportamiento sin entrar en detalles. Se suponía que teniendo un Z80, un mapa de memoria como el del Spectrum y una disposición de la memoria de pantalla como la del Spectrum, el juego debería ir sí o sí. Los autores de emuladores, César el primero, saben de sobra que no es suficiente.
Cosas como hacer un XOR de los dos bits de MIC y SPK apuntan a que realmente no se probó con muchos juegos. En esa época no había WOS, y aunque sí había dispositivos de carga rápida, estos eran, como mucho, unidades de disquete, así que si había un único prototipo de ordenador, era sobre él que tenían que probar todos esos programas, y eso lleva su tiempo.
Pero es que para un circuito que sólo pueda procesar señales digitales, precisamente la operación XOR es la única que permite algo parecido a una mezcla de ambas señales. ¿Qué ecuación booleana hace que dos señales puedan excitar de forma independiente un altavoz sin ahogarse una a la otra? Me imagino al ingeniero buscando como loco la manera de liberar un pin extra en el TAHC10 para poder enviar por separado MIC y SPK.
Curioso también es que revistas de referencia como Microhobby propagaran bulos sobre supuestos RANDOMIZEs de la muerte. ¿Nadie se atrevió a probarlo? No he llegado a ver ninguna nota de rectificación al aviso de “leyenda urbana” que hemos reproducido en el artículo. ¿Quizás no echaron demasiada cuenta al aparato? Eso parece, ya que a la vez que el Inves intentaba abrirse hueco en el mercado, a Microhobby se le caía la baba publicando suculentos artículos sobre las nuevas posibilidades

del 128K (y de paso deshidratando al personal).

La idea que me hago después de haber repasado los Microhobby's es que el mercado acogió con desgana la máquina, que llegaba muy tarde y sin un nicho de mercado que realmente pudiera ocupar. Para colmo cuando se vio que fallaba más que una escopeta de feria, se encargaron de enterrarlo: sólo hubo un número de Microhobby, el 108, en el que Investrónica puso publicidad de su ordenador. Desde entonces, no he encontrado ni un cuadrito de publicidad sobre él exceptuando el artículo de Primitivo sobre las interioridades del Inves, hasta el artículo de Oscar del número 139, y después de él, nada. Podrían al menos haber publicado algún mod, como el de RGB, que no necesita de unos conocimientos muy sesudos de la máquina. El puerto de joystick integrado permite algunas virguerías hardware adicionales, como por ejemplo conectarlo a una serie de sensores y usar al Inves como control central. Da igual: venía herido de muerte y sencillamente lo dejaron morir.

Cuando comencé el traceo de la placa del Inves (por cierto, aún no la he acabado), lo que me dejó completamente a cuadros fue que el diseño no usa en ningún momento la señal WR del procesador. La información de los cronogramas ha sido determinante para apreciar un detalle de genialidad por parte del anónimo ingeniero (o ingeniera): inferir que una operación es de escritura, cuando no tienes dicha información disponible de forma explícita, y además pudiendo terminar ese ciclo en mucho menos tiempo de lo que lo haría el Z80. Tenía que ser así, porque si no, el sistema de slots de tiempo que el TAHC10 genera no funcionaría. Aunque también pienso… ¿no sería más bien porque se quedaron sin pines para poder meter la señal WR? En este caso no lo creo.
El Z80 carece de lo que otros micros de la época, como el 6502, tienen: un comportamiento completamente coherente en cada parte del ciclo de reloj. En el 6502, durante la mitad del ciclo, accede al bus. Durante la otra mitad, no lo hace. Así es muy sencillo multiplexar en el tiempo este procesador con un sistema de video, y de hecho esta es la base sobre la que funciona el binomio 6510 – VIC2 en el Commodore 64: acceden a memoria en semiciclos alternos.
El diseño del Inves consigue restringir los accesos del Z80 a memoria de forma que el TAHC10 no pierde su oportunidad de leer la RAM para generar el video, y no necesita parar al Z80 para conseguirlo. El resultado: el Inves consigue lo que Sinclair y Amstrad no pudieron: un funcionamiento del Z80 sin cortes, y sin quebraderos de cabeza a la hora de calcular el tiempo de funcionamiento de una rutina. Y esto unos 4-5 años antes de que los rusos tuvieran el Pentagon y el Scorpion, y unos 20 años antes de que las memorias pudieran ser lo suficientemente rápidas (respecto a la velocidad del Z80) como para poder simular con ellas memorias de doble puerto y conseguir así un funcionamiento sin contienda en los clones de Spectrum por FPGA.
29 años después es muy fácil coger un analizador lógico USB, conectarlo a tu PC con un procesador miles de veces más rápido que el Z80, pincharlo a un lentísimo Inves, y descubrir con toda la comodidad y resolución de un monitor de 1080 puntos todas sus miserias. Coger después un software que hoy día se consigue gratis, pero que hace 29 años sólo existía (si existía) en los laboratorios de diseño y layout de los fabricantes de chips más reputados, y simular hasta resoluciones de picosegundos el funcionamiento de cualquier sistema, viendo en tiempo real dónde falla y donde no. Mostrar los puntos negros del Inves Spectrum+ "a toro pasado", con la tecnología de ahora, deja al trabajo de aquellos ingenieros en una situación, para mí injusta.
Faltó poco, la verdad, para que el Inves no se estrellara, al menos no tan flagrantemente. Más tiempo, más pruebas, y algo que los aficionados a la informática nos encanta(ba): que nos dieran todos los detalles de la máquina para poder aprovecharla a tope. El Inves se vendió como una caja negra (nunca mejor dicho) para el consumo. Y el consumidor habló: "gracias, pero no nos interesa".
Quizás nuestro anónimo ingeniero no sea un Richard Altwasser. Con todo, con la perspectiva de estos 29 años, y visto todo lo visto (lo bueno y lo malo) tiene todos mis respetos.
Por otra parte, yo (César), quiero manifestar que, pese a todas esas particularidades del Inves, es un ordenador al que tengo mucho cariño, dado que fue con el que aprendí a programar y me inicié en el mundo de la informática. Con el tiempo, fue aún mas gratificante ir descubriendo todos esos aspectos del Inves y cuales eran sus causas.

**Miguel Ángel Rodríguez Jódar**
**mcleod_ideafix**
**@zxprojects**
**http://www.zxprojects.com**

**César Hernández Bañó**
**chernandezba**
**chernandezba@hotmail.com,**
**https://sites.google.com/site/chernandezba/home**

# 10 Modificaciones y mejoras

## para el ZX Spectrum

**Seguramente me quedaré corto al afirmar que los ZX Spectrum marcaron un hito en la historia a nivel informático y electrónico, un producto de bajo coste y prestaciones impresionantes para la época. Mucha gente aprendió con ellos y aún hoy es evidente que seguimos descubriendo y aprendiendo. Desde su lanzamiento las cosas han cambiado mucho y hoy en día estamos en condiciones de rediseñar estos viejos ordenadores para dotarles de mayor fiabilidad, más prestaciones y un menor consumo. En este artículo vamos a ver qué modificaciones, mejoras y reparaciones podemos hacer a un ZX Spectrum de 48K para conseguir lo que podría llamarse un ZX Spectrum v2.0.**

La lista de modificaciones/reparaciones que presento a continuación no son definitivas, cada usuario puede añadir las suyas propias o incluso proponer mejoras; el propósito es que sirvan como un comienzo o fábrica de ideas.
Lista de las 10 modificaciones que he implementado:

1. Adaptación de un DIODO en la entrada de alimentación para evitar averías por cambios de polaridad.

2. Cambio del 7805 original y el disipador por un Step-down regulado a 5V que no se calienta y tiene una salida de 3A

3. LED testigo de alimentación4. Botón de reset

5. Potenciómetro de control de volumen

6. Interruptor RF/AV para conmutar entre salida de vídeo compuesto o salida de vídeo clásica

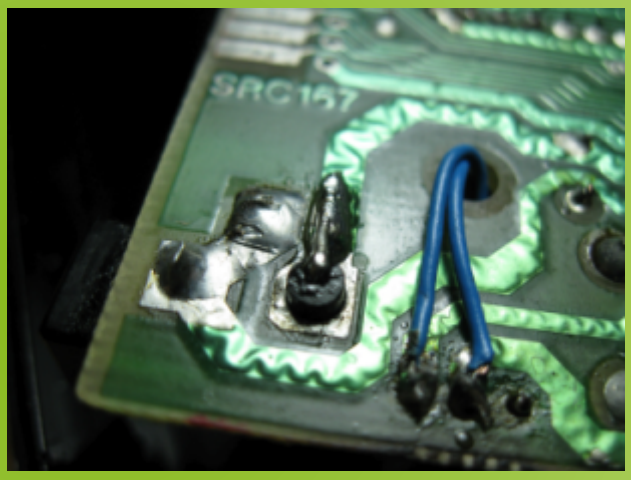
7. Sustitución de todos los chips de memoria por placas íntegras con memoria SMD.

8. Utilización de una ULA reparada
9. Cambio del PCF1306 por un circuito equivalente con componentes SMD

10. Generoso disipador de calor para la ULAAdemás de lo anterior, podemos añadir como mejoras:- 99% de los integrados en zócalos- Cambio de membrana- Sustitución de todos los condensadores electrolíticos
Vamos a ver cada uno de estos puntos por separado.

**1. Adaptación de un DIODO en la entrada de alimentación para evitar averías por cambios de polaridad.**

Ya sabéis que una de los fallos más comunes, por despiste o desconocimiento, es alimentar al Spectrum con otra fuente que tenga una clavija de alimentación que "encaje". Esto puede suponer una avería bastante grande porque estamos cambiado el positivo por el negativo. En cualquier caso, para solventarlo nada mejor que protegerse y que mejor que usar un diodo para tal fin.




En la primera imagen vemos que tan solo habría que levantar la patilla trasera (negativo) del conector de alimentación, doblarla hacia arriba y meter el diodo por el agujero que tiene la propia placa. La verdad es que el hueco es exactamente igual que el diodo que he usado y queda firme, sin holguras.

El diodo utilizado es un 1N4007.

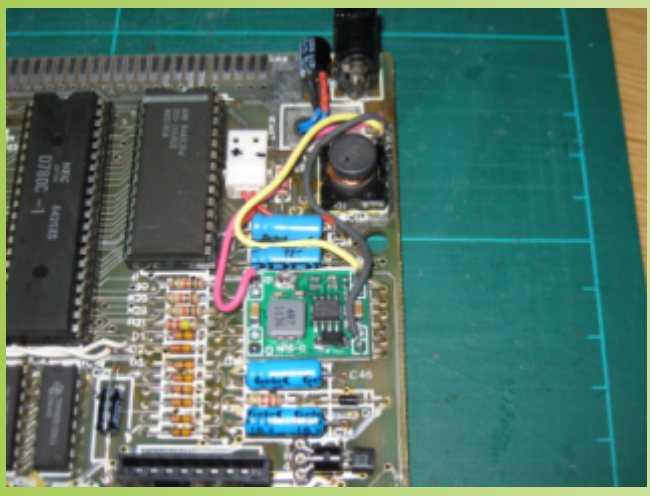
**2. Cambio del regulador de tensión 7805**
Este cambio es prácticamente imprescindible para evitar que el Spectrum se caliente en exceso en su interior, ya que el componente que más calor disipa es el 7805 en su trabajo por reducir la tensión de entrada de la fuente a los 5V necesarios para el funcionamiento de la electrónica.

Existen varias opciones para reemplazar el regulador original, personalmente he probado 3 tipos distintos.

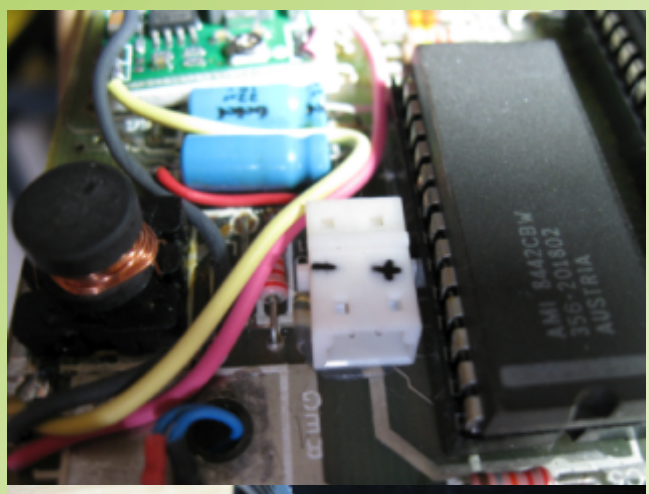
- TracoPower (búscalos con ese nombre en Internet o en tu tienda de electrónica). Se trata de reguladores comerciales que apenas se calientan y tienen una salida máxima de 1A.

- Reguladores no comerciales. Pueden adquirirse en tiendas online, generalmente de origen chino y que son equivalentes a los Tracopower por un precio algo más barato.

- Step-Down regulable. La mejor alternativa que he encontrado es un circuito regulador variable que tiene una salida máxima de 3A. Esta característica nos ayudará cuando conectemos varios periféricos al slot trasero del Spectrum. Con los reguladores anteriores he comprobado que en ocasiones el Spectrum no arranca normalmente y hay que realizar un reset.



**3. LED testigo de alimentación.**
Sin duda lo más sencillo de implementar y es muy útil para saber si el ordenador está encendido y a su vez evitar que conectemos o desconectemos periféricos con él en marcha. La tensión la podemos sacar de ambos polos de C34. Le he puesto un conector que resulta muy útil para poder conectar y desconectar el LED cuando hay que abrir el Spectrum. Al cerrarlo se vuelve a conectar y listo.

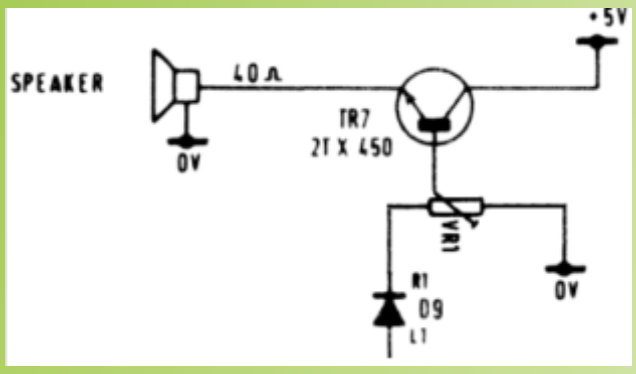



El diodo LED utilizado es de color rojo, 3mm de diámetro y protegido por una resistencia de 1K.

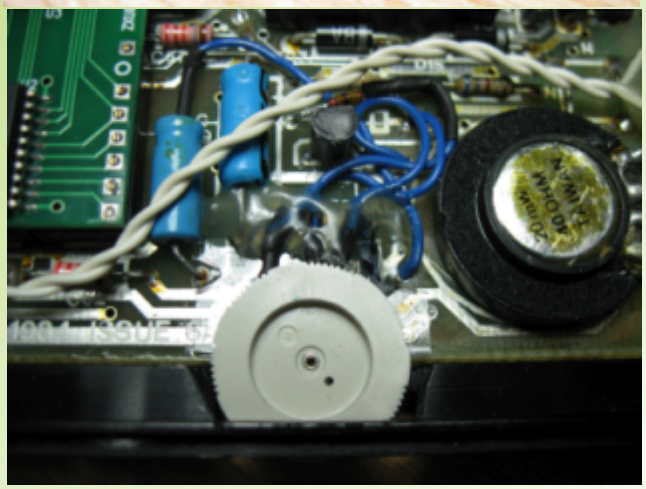
**4. Botón de reset.**
Yo diría que es imprescindible por muchas razones pero la fundamental es la de no tener que estar quitando y poniendo el conector de alimentación del Spectrum. El botón de reset alarga la vida de los componentes y del ordenador en general porque no hay que quitar tensiones de todos los integrados y componentes para al momento volver a alimentarlo todo. Las memorias sufren bastante por este motivo y además, es muy cómodo.


En un Issue 6A (como es el caso) el botón de reset viene de serie, pero siempre podemos sacarlo de los bornes de C26 o C27.

**5. Control de volumen.**
Seguro que más de una vez, en el silencio de la noche, habéis notado que el ZXSpectrum chillaba demasiado con algún juego o durante la carga de un programa...Pues eso se acabó con la implementación de un control de volumen regulable.



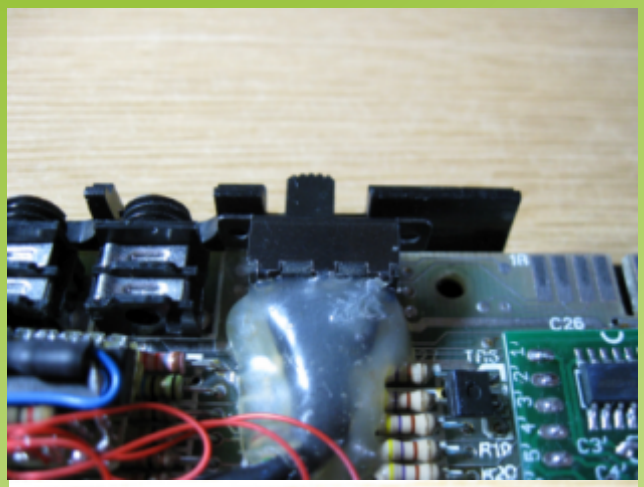

En la ubicación que he encontrado queda discreto.

La idea ya la planteé hace mucho tiempo en el foro amigo de Speccy.org y finalmente la pude poner en marcha. Por cierto, el potenciómetro es reciclado de una vieja unidad lectora de CD.

**6. Interruptor RF/AV**

Me gustaba la idea de poder obtener desde el mismo conector RCA de la salida original tanto vídeo compuesto o RF. La forma que he ideado es la incorporación de un interruptor que permita conmutar entre uno y otro modo según la tele que vaya a usar. El montaje es muy fácil, es como hacer la modificación de vídeo compuesto clásica que puede encontrarse en múltiples páginas web y foros, pero en vez de unir el cable con la señal de vídeo directamente al polo central del conector, le metemos un interruptor y podemos conmutar entre una y otra señal.

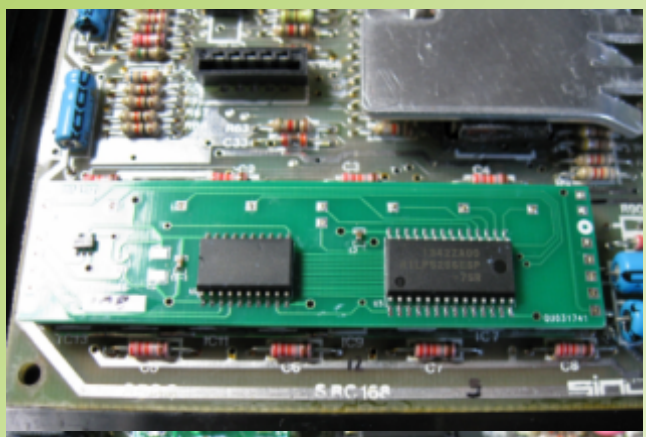


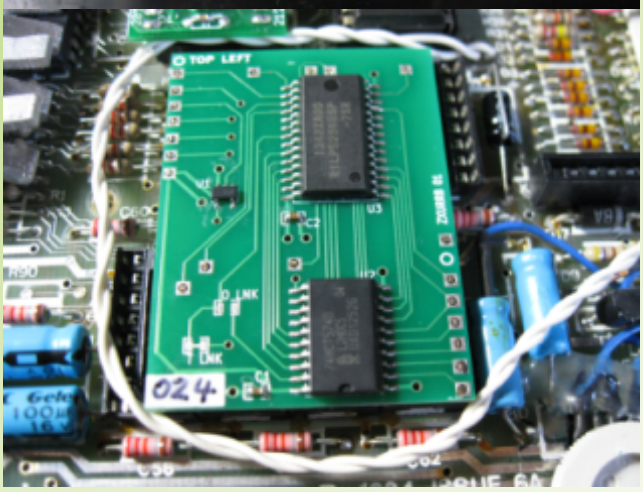

Como véis, lo he situado al lado de los conectores de audio del Spectrum. Hay que mecanizar un poco la carcasa para poder manipular el interruptor pero lo he dejado de tal manera que no moleste al conectar cualquier interface al puerto trasero.

7. Cambio de memorias

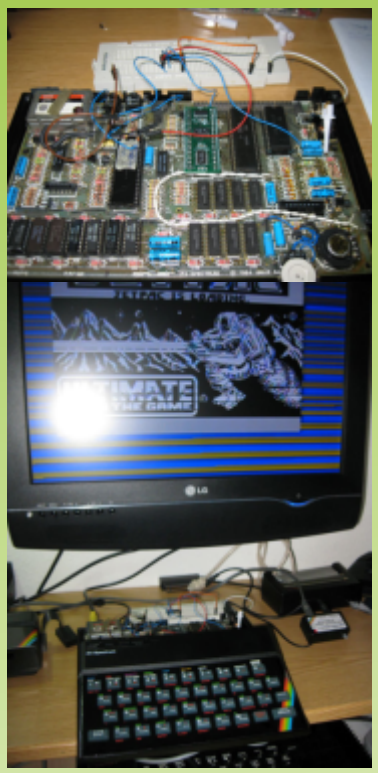
Ya desde hace tiempo han surgido algunos sustitutos de las memorias del Spectrum, sobre todo de la memoria baja y es que los 4116 escasean y además son muy propensos a fallos y errores. La solución es cambiarlo por integrados de memoria modernos que únicamente necesitan 5v para funcionar y por supuesto, son más estables y no generan calor, lo que redunda en un consumo más eficiente de la energía. En mi caso he cambiado tanto la memoria baja como la memoria alta, el único requisito es que hay que sacar todos los chips de memoria y sustituirlos por zócalos con el fin de poder insertar esas nuevas placas.

Pueden conseguirse en distintas páginas web, como alguna popular de subastas o como en mi caso, a través de la web: http://zx.zigg.net/.



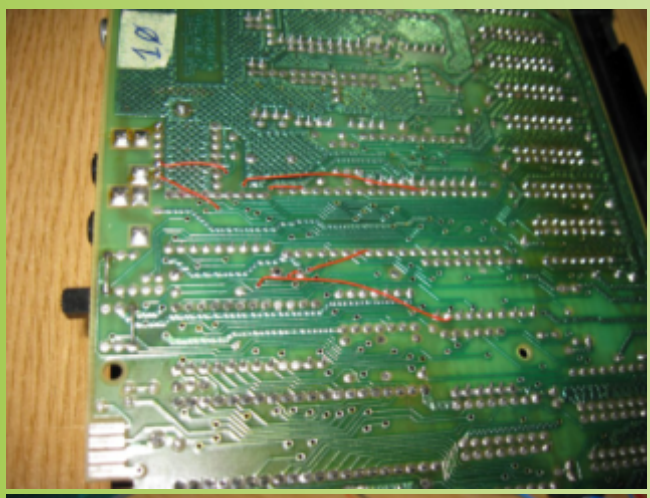


8. Uso de una ULA reciclada
Todos sabemos que el punto débil de un Spectrum en cuanto a repuestos es la ULA, el único componente que no puedes sustituir por ningún otro integrado comercial. Sabemos que hay proyectos que permiten su realización pero hoy por hoy no se pueden adquirir comercialmente.
Lo que intenté es hacer uso de una de esas ULAs que están semi-estropeadas, en esta ocasión la ULA parecía funcionar bien pero no cargaba de casete. Obviamente no podemos acceder a su interior para repararla, pero es posible construir el "trocito" de ULA que controla la carga del casete fuera de ella y que se puedan volver a cargar programas.
Esta solución la ha aportado mcleod_ideafix en el foro Speccy . Su gran conocimiento acerca de este integrado le permite aportar ideas muy ingeniosas. Desde aquí mi agradecimiento.Para su reparación usé un integrado 74HCT125.Al principio monté el integrado y los cables con la ayuda de una placa de pruebas:



Posteriormente usando ya el integrado, monté el 74HCT125 en una placa perforada para poder soldar con más comodidad, recorté los cables lo justo para evitar que quedaran demasiado largos y lo que hice fue coger los puntos de soldadura debajo de la placa y pasar los cables por lo agujeros, así queda más recogido.

Toda la información técnica podemos leerla en el post original del foro Speccy:
http://foro.speccy.org/viewtopic.php?f=8&t=3707
9. Cambio del PCF1306
José Leandro tiene en el trastero

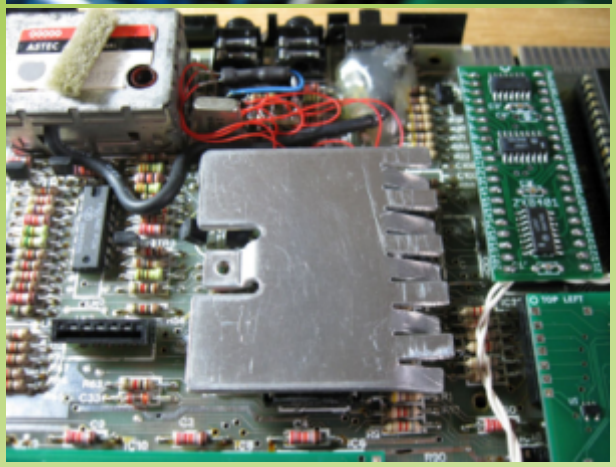


(http://trastero.speccy.org/cosas/JL/pcf1306p/PCF1306P.html) un artículo donde explica cómo determinó los componentes que hay dentro del integrado PCF1306 que montan las Issue 6. Un trabajo excelente sin duda y lo mejor es que queda muy bien realizado en componentes SMD (tiene otra versión con componentes discretos). Resulta imprescindible levantar el integrado de 40 patillas y poner un zócalo.Queda de la siguiente manera:

Ahora si tenemos alguna avería en alguno de esos integrados se pueden sustituir como hacíamos en ISSUES anteriores.
10. Disipador de la ULA
No cuesta nada

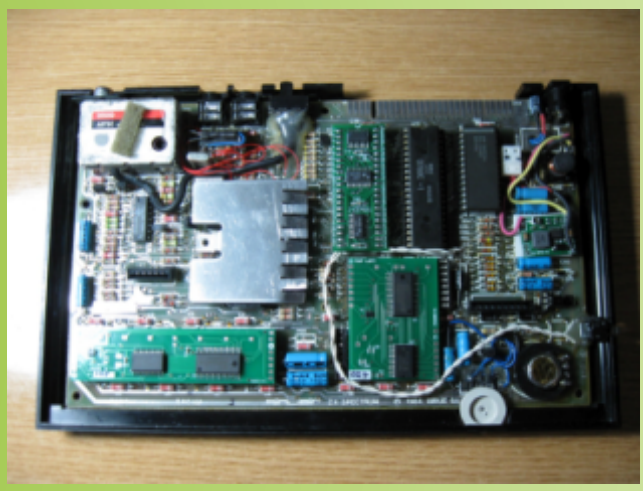
y alargaremos la vida de este componente. Simplemente recortar un trozo de aluminio del tamaño adecuado (max. 1,5mm de altura), hacerle algunos cortes tipo rejilla para mejorar la disipación, colocarlo encima de la ULA con pasta térmica y 4 gotas de pegamento rápido en las esquinas. No se moverá y hará que la temperatura de trabajo descienda notablemente. Si la ULA se estropea será por otro motivo, pero no por calor.

Y finalmente este es el aspecto que tiene mi Spectrum con las 10 modificaciones anteriores.



Mejoras implementadas

En cuanto a las mejoras que comentaba al principio, una de ellas era el cambio de membrana. Muchas veces este cambio resulta imprescindible porque las membranas originales se han degradado con los años. Afortunadamente son fáciles de encontrar, fabricadas en materiales más duraderos y de más calidad. Otra mejora que podemos hacer para facilitar la reparación y comprobación de componentes es sacar los integrados y montarlos de nuevo en zócalos. Por último, resulta casi imprescindible sustituir todos



los condensadores electrolíticos de la placa; después de 30 años en su mayoría no funcionan al 100% y aparte de poder provocar alguna avería mejorará notablemente la calidad de la imagen en el televisor.

Hay dos cosas más que no he nombrado y es la incorporación de un condensador de 100uF y 25V para evitar el rizado en la entrada de la alimentación (se puede ver en las primeras fotos). Tan solo habría que montarlo en paralelo con la salida del regulador. Por otra parte el TR4 lo he puesto en zócalo, el más susceptible de quemarse por culpa de alguna memoria baja en mal estado, aunque eso ahora con la instalación de las placas que sustituyen las memorias, creo que será muy poco probable.

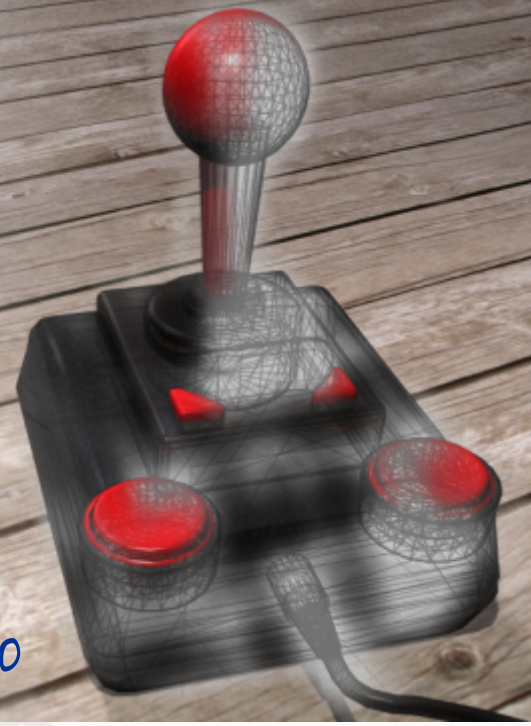
Final

Realizar modificaciones de este tipo no solo son útiles para alargar la vida de los componentes y conseguir un Spectrum más estable, también resulta muy gratificante ver que el trabajo y tiempo invertido han dado muy buenos resultados.

Quedan para más adelante otras ideas como la sustitución de la ROM original por una múltiples con varios sistemas.

***Fernando** Fernández (fermars)*
***28/10/2014***
***fermars@gmail.com***

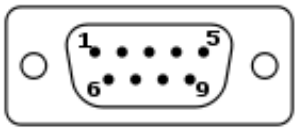
# Los entresijos de un interface Kempston



## Picoteando Hardware con Wilco

***Hola amigos. Llevaba tiempo pensando en escribir un artículo para esta fantástica revista a la que yo tanto aprecio, y al final me he decidido.***
***Como muchos de vosotros sabéis, yo no soy un hombre de juegos. Me gusta mucho más destripar el funcionamiento de las cosas, ya sean estas de hardware o de software, así que, a la hora de pensar sobre un tema para mi artículo he decidido intentar transmitiros a vosotros un poco de ese ansia por destripar las cosas.***
***Éste que estáis empezando a leer pretende ser el primero de una serie de artículos sobre cómo funcionan determinados circuitos de hardware para nuestras retro-máquinas.***
***He elegido el interface Kempston, porque se trata de un interface muy sencillo, con un chip y unas pocas resistencias y diodos, y sobre todo un interface muy fácil de entender, lo que lo convierte en ideal para iniciarse.***
***De hecho, he de confesaros que este fue el primer interface de hardware que me esforcé en analizar y entender y que me sirvió de primer paso para comenzar con el hardware del Spectrum.***

**¿Como funciona un interface Kempston?**
Como sabéis, un interface Kempston permite conectar cualquier joystick de la norma Atari a nuestro Spectrum.
La norma Atari consiste en 5 switches (las cuatro direcciones más disparo), conectados a un conector DB9 con el siguiente esquema:




| | | |
|---|---|---|
| **Pin 1** | JOY 0 | Arriba |
| **Pin 2** | JOY 1 | Abajo |
| **Pin 3** | JOY 2 | Izquierda |
| **Pin 4** | JOY 3 | Derecha |
| **Pin 6** | BOTON | Fuego |
| **Pin 8** | GND | Signal Ground |

Cuando el jugador mueve el joystick hacia una de las direcciones o pulsa el botón de disparo, uno de estos switches es pulsado. Esto hace que se cortocircuite con la masa (pin 8), el pin correspondiente a la dirección que se haya accionado con el joystick.
Del resto se encarga la electrónica que convierte este cortocircuito en un número que es pasado al bus de datos y después leido por el software.

**Puertos de Entrada/Salida**
Pero ¿cómo hace el software para comunicarse con el hardware del spectrum y leer este número que nos llega desde el bus?. La respuesta es simple, para esta tarea se utilizan los puertos de entrada/salida.
Cuando un periférico quiere enviar un dato al procesador, siempre lo hace mediante un puerto de entrada ( o de lectura), mientras que cuando el procesador quiere enviar un dato a un periférico se utilizará un puerto de salida (o de escritura)
En el Z80 se puede disponer de un total de 65536 puertos de entrada y otros tantos de salida, y cada uno de ellos viene identificado por un número.

El proceso mediante el cual el Z80 lee un dato desde un periférico es el siguiente
El Z80 envía una instrucción de lectura de puerto (instrucción “IN”), con el número del puerto a leer como parámetro.
Después pone el número del puerto a leer en el bus de direcciones.
Activa la patilla IORQ del z80
El hardware del interface comprueba que la línea IORQ del bus está activa, lee el bus de direcciones y comprueba si el número que hay allí corresponde con su dirección de puerto.
Si es así, el interface deposita el dato en el bus de datos.
Finalmente, el Z80 lee el dato que ha puesto el hardware en el bus de datos y retira la señal IORQ.

**Lectura del interface Kempston desde Basic**

En el punto anterior hemos visto cómo se produce la comunicación entre un interface y el procesador. También hemos visto que se necesita asignar un número puerto para poder llevar a cabo la comunicación. En el caso del interface Kempston, el puerto que asignaron los ingenieros que lo diseñaron fue el puerto 31.
Para llevar a cabo la lectura de un puerto desde Basic del Spectrum, existe la función IN, a la cual hemos de pasarle el número del puerto que queremos leer, y esta función nos devuelve el valor que pone el interface en el bus de datos.

Por ejemplo, el siguiente programa en BASIC mostrará en pantalla de manera indefinida el valor devuelto por el joystick.

```
10 PRINT AT 0,0; IN 31; “   “
20 GOTO 10
```

El valor que muestra el programa variará dependiendo de hacia donde tengamos la palanca del joystick, conteniendo dicho número toda la información que necesitamos para saber dicha posición desde nuestro programa.

Para representar esa información es suficiente con 5 bits (tantos como switches hay en el joystick) codificados en los bits 0..4 del valor devuelto por IN y teniendo en cuenta que si el bit está a 1 significa que el switch correspondiente está pulsado.
Los otros tres bits sobrantes deberían estar siempre a 0.

| | BIT 4 | BIT 3 | BIT 2 | BIT 1 | BIT 0 |
|---|---|---|---|---|---|
| DIRECCIÓN | FUEGO | ARRIBA | ABAJO | IZQUIERDA | DERECHA |
| VALOR DECIMAL | 16 | 8 | 4 | 2 | 1 |

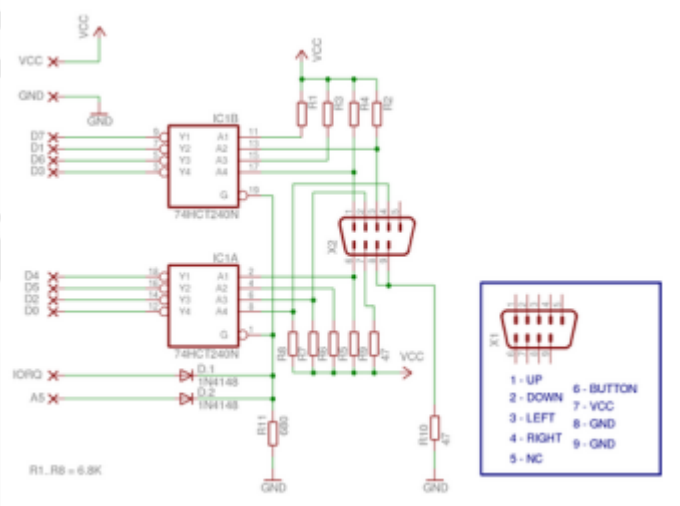
Si tenemos pulsado más de un switch de dirección, el valor devuelto será la suma del valor correspondiente a todos los switches pulsados.

Por ejemplo, si tenemos el joystick en la diagonal derecha/arriba y el botón del disparo pulsado, el valor devuelto por IN será 1+8+16 = 25.

**Vale, todo eso ya lo sabíamos, pero ¿cuándo vamos a entrar en materia?**
**Comencemos con el hardware del interface.**

Después del impepinable rollo anterior creo que ya podemos empezar con la descripción del interface propiamente dicho.

Empecemos con el esquema de la criatura:




Como hemos explicado antes, el hardware tiene que hacer dos cosas:

1º Interpretar cuando se está haciendo una llamada al puerto 31, para lo cual deberá comprobar que está activa la línea IORQ, y saber leer el contenido del bus de direcciones para compararlo con el número 31.
2º Devolver un número que contenga el estado del joystick codificado tal cual hemos explicado antes.

Una cosa importante que debemos tener en cuenta de la línea IORQ, es que es activa en baja. Esto significa que su lógica es inversa a como podríamos pensar, ya que está activada cuando está a 0v, y desactivada cuando está a +5v.

Por otro lado, para comprobar si en el bus de direcciones tenemos el número 31, deberíamos comprobar todas y cada una de las líneas de direcciones (A15..A0) para que tengan el siguiente contenido (0000000000011111) Esto es lo que llamamos decodificación completa e implicaría un montón de electrónica.
Para ahorrar dinero en hardware se utiliza lo que se llama decodificación parcial, que no es ni más ni menos que decodificar solo parte de la dirección, lo que ahorra en hardware aunque desperdicia gran cantidad de puertos.
En el caso del interface Kempston solo se comprueba A5=0, lo que implica conflictos con cualquier hardware que use un puerto que tenga A5 a cero.
Afortunadamente tenemos muchos puertos disponibles y no tantos interfaces desarrollados para Spectrum, por lo que no hay conflictos conocidos con el interface de joystick Kempston.

Utilizando decodificación parcial, lo que debe de hacer el circuito es comprobar si IORQ está activa y si A5=0, por lo que si IORQ fuera activo en alta, la lógica necesaria para comprobar este puerto de forma parcial sería la siguiente:

IORQ AND NOT A5

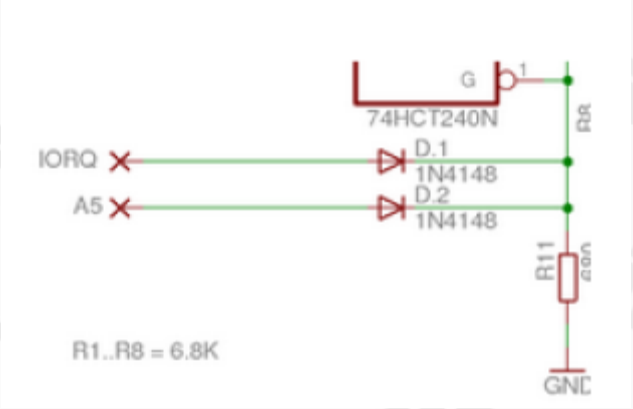
Aunque como IORQ es activa en baja, en realidad deberíamos utilizar lo siguiente:

NOT IORQ AND NOT A5

O lo que es lo mismo:

NOT(IORQ OR A5) si queremos el resultado en activo en alta y IORQ OR A5 si decidimos usar el resultado como activo en baja, es decir 0V = activo.
Lo que resulta extremadamente sencillo de comprobar por hardware.

Echémosle un vistazo a la parte del circuito que se encarga de decodificar el puerto.




Básicamente se trata de una puerta OR hecha con diodos y resistencias que tiene como entradas las señales IORQ y A5.

La salida de esta puerta OR nos dará una señal que estará a 0V siempre que se esté haciendo una llamada al puerto 31. Si quisiéramos utilizar una señal activa en alta, habría que invertir esta señal utilizando una puerta NOT, pero dependiendo de cómo vayamos a utilizarla luego, es muy probable que nos sirva tal cual está, con lo que nos ahorraremos algo de electrónica.

En el circuito vemos que la salida de esta puerta OR va conectada a las patillas 1 y 19 de IC1, que es un buffer octal triestado.
Estas patillas (1 y 19) son, ni más ni menos, que la activación de la salida de las dos mitades del buffer (se pueden activar de manera independiente), y tienen la particularidad de que son activas en baja, es decir si aplicamos 0v activamos las salidas, por lo que nuestra señal decodificada nos viene como anillo al dedo para utilizarla como activación de este buffer.
Al conectar la salida de nuestro OR a esas dos patillas, en el momento que detectamos una lectura del puerto 31 se activarán las salidas del chip.

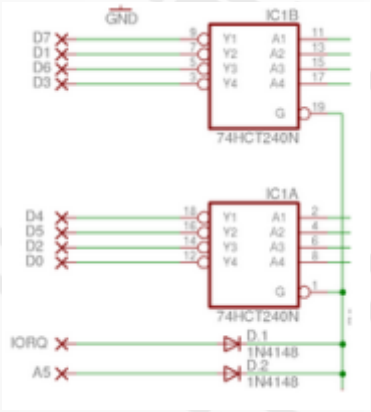
Ahora que ya tenemos decodificado el puerto, vamos a entrar a ver el funcionamiento de la copia del estado del joystick en el bus de datos.

El componente principal que realizará dicha tarea es el buffer, dual, inversor octal triestado (Ic1).
Inversor porque invierte el estado de las entradas en las salidas.
Dual, porque tiene dos mitades que se pueden activar o desactivar de manera independiente.
Aunque nosotros lo usaremos como si fuera simple.

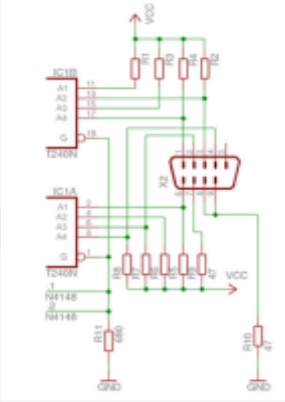
Octal porque consta de 8 bits.
Triestado porque sus salidas pueden estar en tres estados diferentes: 0V, +5V y alta impedancia.

Cuando las patillas 1 y 19 del chip están a +5V las salidas estarán en estado de alta impedancia (o sea, que es como si se desconectaran de las líneas a las que van conectadas).
Cuando las patillas 1 y 19 están a 0V, las salidas pasarán a tener el estado inverso de las entradas. Es decir si la patilla 2 está a 5V, la patilla 18 estará a 0V.

En nuestro esquema, tenemos conectadas las entradas a los switches del joystick, las salidas al bus de datos y las dos patillas de habilitación (1 y 19) a nuestra puerta OR. Por tanto, mientras no se esté realizando ninguna consulta del puerto 31, las salidas del chip están en estado de alta impedancia, por lo que no afectarán al estado del bus de datos, y en cuanto se detecte una llamada a dicho puerto, el buffer invertirá las señales que le llegan desde el joystick volcando el resultado sobre el bus de datos




Centrémonos ahora en las señales provenientes del joystick.

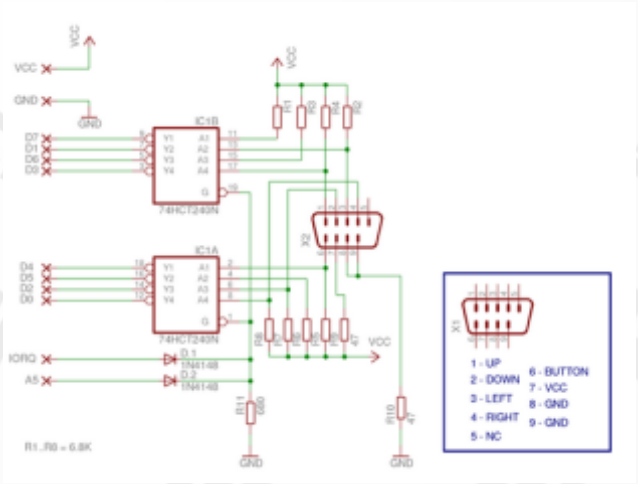


Como puede verse en el esquema, las entradas del buffer (pines 2,4,6,8,11,13,15 y 17) están conectadas a cada uno de los pines del puerto del joystick, y a +5V a través de una resistencia de 6k8.

Cuando conectamos un joystick a dicho conector, cada uno de los switches correspondientes a cada dirección queda conectado por un lado pin que le corresponde en el conector, y por otro lado a los pines 8 y 9 que a su vez están conectados a masa.
En estado de reposo, todos los switches están abiertos, por lo que las entradas del buffer estarán sólo conectados a +5V, tensión que les llegará a través de las resistencias R1..R9.

Cuando movamos el joystick, por ejemplo hacia arriba, el switch conectado al pin 1 del conector DB9 cerrará el circuito con masa a través de los pines 8 y 9 del DB9, por lo que el pin 17 del buffer, que está conectado con el pin 1 del joystick (arriba) y que antes estaba a +5V, cambiará a 0V, ya que la corriente que le llegaba desde R4, se derivará a masa a través del microswitch, dejando a 0V la entrada (pin 17) y a 5v la salida (pin 3). Como el pin 3 está conectado con el bit D3 del bus de datos, dicha línea quedará con un 1 lógico.

El resto de los pines funcionan de la misma manera, aunque hemos de hacer notar que el diseñador del circuito no ha mantenido el orden natural de las líneas de datos para facilitar el trazado del circuito impreso. Esto no tiene ninguna importancia mientras mantengamos la correspondencia entre cada señal del joystick con cada línea del bus de datos (D1=arriba, D0=abajo, D3=izquierda, D2=derecha, D4=fuego)




De esta manera todos las entradas que no tengan su microswitch activado quedarán a 1 mientras las que no quedarán a 0, formando entre todas el dato a devolver al z80.

Y eso es todo por hoy, espero que no os haya resultado muy complicada la explicación.
Os animo a que me preguntéis cualquier duda que tengáis a través del foro. Será un placer para mí responder vuestras cuestiones.

Pido que seáis benévolos si habéis detectado alguna inexactitud, y os animo a que hagáis crítica constructiva para poder mejorar mis artículos en próximas entregas.

Un saludo y hasta la próxima!!

***Wilco2009.***

# COMO HACER MI PROPIA RECREATIVA ARCADE


**A la mayoría de los aficionados a los videojuegos de la vieja escuela nos presentaron los videojuegos en las recreativas, las recreativas eran donde estaban las novedades, donde se veían los mejores gráficos y el mejor sonido.**
**Es por ello, que para muchos de nosotros siempre fue un sueño disponer de una recreativa. Con los años la recreativas han ido desapareciendo, las novedades vienen en las consolas y aunque han quedado obsoletas en el mercado comercial siempre tendrán un hueco en un rincón de nuestro corazón.**

Mucha gente compra maquinas y bien las restauran exteriormente solamente o bien le meten dentro un pc, para poder cargarla de emuladores. Incluso se hacen el mueble que alberga la maquina ¿pero como hacer esto? ¿Es fácil? ¿que componentes necesitamos? ¿que software hay disponible? Son muchas preguntas que nuestro invitado ha ido resolviendo y que las publica en su libro, pasemos a leer esta interesante entrevista, con David Provencio.

¿Quien es David Provencio? hablamos de ti, ¿Como se te ocurrió esto de un libro llamado: '¿Cómo hacer mi propia recreativa arcade'?

- Hola, soy técnico en telecomunicaciones, especializado en electrónica y desde hace unos años trabajo como autónomo, con la crisis me he ido reconvirtiendo :). Sobre el libro decir que inicialmente iba a ser un pequeño manual de pocas páginas, pero después lo fui ampliando y mejorando hasta que al final pensé: esto tengo que editarlo y sacarlo en papel, y ya es cuando me lancé a la aventura de sacar un libro.

Cuéntanos el objetivo del libro, ¿Que se encontrara el que lo compre? Por que he visto que tienes la posibilidad de comprarlo con material extra...

- El libro tiene casi 200 páginas a color y explica de manera visual todos los pasos para hacerte tu propia 'recre', dando un repaso a todos los aspectos que se tocan: Software, hardaware, construcción del mueble, conexiones. La idea es usar un ordenador antiguo que tengamos por casa y que no demos uso y reconverito para jugar a estos juegos retro que piden pocos recursos.

Desde hace poco, además de vender el libro con la plantilla de la máquina más el DVD, hemos añadido unos packs de joysticks y botones para 1 o 2 jugadores, para que la gente cuando adquiera el libro pueda tener facilmente el material arcade necesario, sólo tendrá que poner el ordenador y hacer el mueble con la ayuda del libro.

¿Quienes lo hicieron posible? tu solo, tuviste colaboradores, socios...

- El libro está escrito por mí pero por supuesto el mérito es también de mi pareja y familia que me


ayudó en correciones de estilo y formato. Para que quedase un libro fácil de leer y entendible por cualquiera. Además ellos me siguen ayudando mucho con la distribución, en ferias, envíos por correo, etc..

¿tuviste apoyo de algún organismo publico?

En realidad no pedí apoyo, para este caso no ví ninguna que aplicase. La inversión inicial para sacar un libro de este tipo: a color, con plano en tamaño A1 y un DVD es bastante grande, pero me podían las ganas de hacerlo y ví que había un hueco de mercado por cubrir.

Si me han ayudado de manera indirecta en algunas ferias, ya que muchas veces estas ferias retro reciben ayuda para el recinto, etc.. de organismos o ayuntamientos. Seguro que este tema lo conoces tú mucho mejor ;)

¿hay alguna anécdota ocurrida en la elaboración del libro?

Pues sí, lo presentamos el año pasado en RetroMadrid y llegarón los libros impresos de Valencia 3 días antes de empezar, así que estaba cruzando los dedos para que llegase a tiempo y cruzandolos otra vez para que estuviese todo bien, color.., maquetación... al final todo correcto y se presento a tiempo.

¿Tienes pensamiento de sacar otros libros enfocados a la retro informática y los videojuegos?

- Me han pedido que haga uno para crear Pinballs o hacer la 'recre' en vez de con un ordenador que sea con una Raspberry pi que está muy de moda. Seguramente cuando termine con la distribución de la tirada física del libro, lo lanzaré en formato digital y agregaré estas peticiones a modo de suplemento... pero siento decir que en papel no sacaré más libros, requiere mucha dedicación hacerlo y aún más venderlo!!

Llevando un poco el tema a tu persona, ¿Eres aficionado a la electrónica, retro informática y los videojuegos?

- Por supuesto, me encanta jugar aunque cada vez tengo menos tiempo. Me gustan los juegos arcade por que me recuerdan como a todos a cuando jugabamos en los recreativos. La electrónica también me apasiona, tengo en casa la rasberry pi y un arduino, pero como digo, el problema es que tengo poco tiempo para querer hacer tantas cosas...

Si puede ser dinos tus dos máquinas retro preferidas, microordenador o consola y el por que.

- En mi caso destacaría el Spectrum de 128k que tenía, fue un regalo de comunión y yo creo que ahí empecé... también destacaría la psp, que no es que sea muy retro que se diga, aunque bueno cada vez lo es más, la tengo especial cariño y todavía juego con ella cuando me voy de veraneo a la playa, es una todo terreno.


Y ya que estamos dime 3 juegos retro que destacarías en tu vida, ya se que decir solo 3 es injusto, pero puestos a destacar 3, ¿cuales sobresalen en tu cabeza?.

Hmm... veamos... en primer lugar "Caveman Ninja" me encanta lo bien que está hecho y el toque de humor que tiene, me lo he pasado un montón de veces y no me canso. Después el famoso "Ghosts'n Goblins" ¡¡madre mía que juego más difícil!!, me acuerdo cuando jugaba en Benidorm y me duraban los 5 duros menos 2 minutos. Finalmente, destacaría con mucho cariño el juego "Fred", lo jugaba en el spectrum cuando era peque, trataba de un explorador que debía salir de un laberinto y esquivar a los fantasmas y bichos que iban apareciendo. Con el tiempo descubrí que el juego fue desarrollado por Españoles, que buenos recuerdos...

Una pregunta de la difíciles ¿Que opinas del panorama presente y futuro de esta afición?

Es difícil sí, jeje, veo que hay más gente de la que parece que le gusta el tema gamer retro, y también que en las ferias, y mira que yo vendo allí el libro, se llena cada vez más de stands de venta y se convierten estas reuniones de amigos en un mercadillo en donde se especula con los precios... no es algo que me preocupe ciertamente pero es lo que veo.

Para finalizar, Dinos donde adquirir el libro...

El libro se puede comprar en la Web oficial de www.mimaquinarecreativa.com ó enviando un correo a mimaquinarecreativa@gmail.com, se peude pagar por transferencia o por PayPal.

También se puede adquirir en las ferias retro a las que asistimos y que vamos anunciando previamente en la web oficial del libro. Si vaís a alguna podréis ver y probar la recre que se hace con el libro, que siempre llevamos una para que la gente juegue.

¿Quieres añadir algo?

Daros las gracias por permitirme dar a conocer el libro, y agradecer a todos los que nos han realizado comentarios y enviado fotos de las recreativas que han creado con ayuda del libro!!. Es lo mejor de haber realizado este proyecto, los buenos comentarios que hemos recibido y las fotos que nos mandan. Así que gracias a todos los que se han interesado por el proyecto.

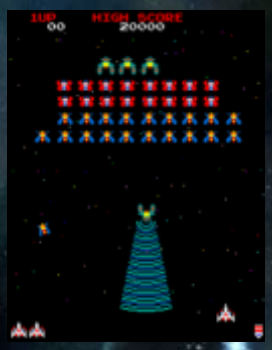
Muchas gracias y un saludo.

# Galaga '88

## El eslabón perdido entre el matamarcianos monopantalla y el matamarcianos con scroll vertical moderno

El género de matamarcianos es casi tan antiguo como la propia existencia de los videojuegos. Hace sus primeras incursiones y comienza a definirse como tal desde los orígenes del videojuego, a comienzos de los 60, en los primeros superordenadores para grandes instalaciones de trabajo y universidades, como los PDP-1. Así, el que es considerado como el primer juego interactivo para ordenador, es a su vez considerado el primer matamarcianos. Hablamos de Spacewar!, desarrollado en 1961 por Steve Russel, en el MIT. En él, dos naves deben tratar de destruirse entre sí, mientras orbitan en torno al campo gravitatorio de una estrella, tratando de no caer en ella. La trayectoria de sus disparos se verá alterada, curvada y ralentizada por el campo gravitatorio de la propia estrella. Este juego no será publicado en máquinas recreativas hasta comienzos de la década de los 70. Le seguirán tantos otros, al principio en gráficos monocromos, sólo más tarde se introducirá el color. Pero no será hasta la aparición de Space Invaders, en 1978, en Japón, título para recreativas, y posteriormente videoconsolas y ordenadores domésticos, de gran éxito, que supone un gran salto adelante en el desarrollo, diseño y difusión de los videojuegos, cuando comenzamos a tener de forma más o menos definida las características básicas del género.

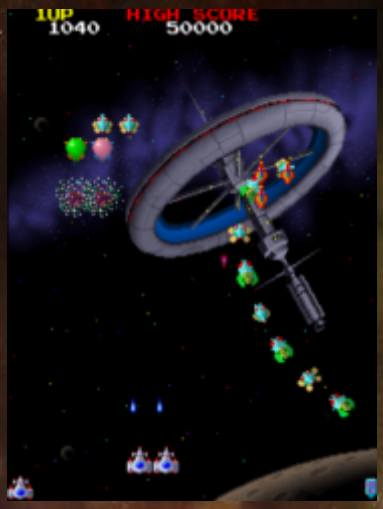
Comienzan a surgir así los primeros matamarcianos monopantalla, con estética, planteamiento y mecánica a menudo derivada del Space Invaders original. Surgen variantes y derivados como Galaxian, en 1979, desarrollado por Namco para recreativas, ordenadores y consolas de 8 bits, que incorpora la innovación de que el enjambre de naves invasoras avanza gradualmente hacia la posición del jugador, a modo de kamikazes, pudiendo alcanzar fácilmente al mismo si no son detenidos antes de llegar al fondo de la pantalla. Y luego más tarde, en 1982, sugirá Galaga, de la misma compañía, concebido como secuela de éste. En él la variedad de enemigos y patrones de movimientos de éstos se complejiza enormemente, a la par que comienzan a introducirse las primeras nociones de decorados, en fondo vacío espacial rociado de polvo cósmico y estrellas. Los enemigos se lanzan a muy mayor velocidad hacia el fondo de la pantalla, describiendo hipérboles y vuelos en picado hacia trayectorias más o menos aleatorias pero cercanas al jugador y disparando proyectiles mientras lo hacen, pero lo harán turnándose, de manera consecutiva o concatenada, a distintos intervalos. Algunos requerirán uno y otros 2 ó más disparos. Le seguirá Gaplus en el 84.




El juego del que vamos a hablar hoy es una reedición de este Galaga original, considerado uno de los primeros matamarcianos, que hará aparición, de la mano de la misma compañía, a las recreativas de 16 bits hacia finales de los 80. Se trata de Galaga '88.

Los jugadores que lo recordamos comenzamos a verlo en los muebles arcade en España ya a comienzos de los 90. Supuso una revolución entre la comunidad de jóvenes jugadores por su componente altamente adictivo, su colorido, viveza de sonidos, y variedad de decorados, así como mayor variedad de sprites enemigos, o por el reto que suponía ir avanzado hacia sucesivas pantallas, la variedad de movimientos...

A diferencia del Galaga original, este Galaga '88 incluye varios efectos, como el efecto de salto al hiperespacio o velocidad de la luz entre fase y fase, así como distintos decorados de una fase a otra, con lo que comienza a superarse la propia concepción inicial de este concepto de juego de "Matamarcianos monopantalla" a "multipantallas" y semiscroll.

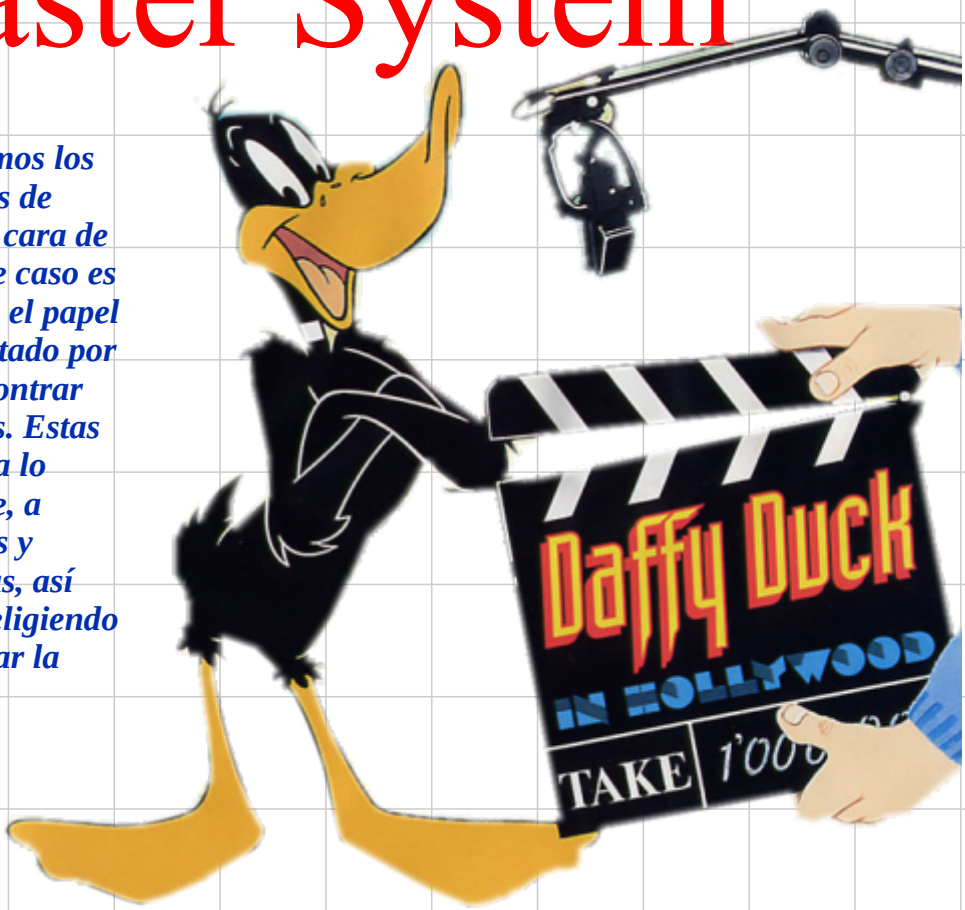



Destaca por su gran jugabilidad en lo que se refiere a la fluidez del movimiento y manejo y la agilidad del disparo, la posibilidad de elegir entre dos modalidades de juego, una con nave simple, que nos otorgará mayor cantidad de vidas, y una con nave doble, que nos proporcionará un único intento; esto se ve contrastado con su dificultad relativa, que resulta considerable en las primeras partidas pero irá siendo superada a medida que logremos superar las primeras pantallas y avancemos en el juego. Se desarrollan las ideas de ítem y de "bonus" a partir de la tercera pantalla, y se incorpora una nueva modalidad de fases, en niveles avanzados, que nos permiten avanzar en scroll vertical a través de un mapeado con multitud de elementos y trampas estáticas o móviles, y la idea de jefe final en cada "zona" o lote de actos. Un juego de una gran sencillez a la par que belleza de diseño, con contadas pero acertadas y originales músicas de introducción y en determinadas fases de fondo, y derroche de efectos sonoros que harán buen uso de los chips de síntesis FM incorporados junto al chip principal de 16 bits Motorola 68000 en la placa de la recreativa arcade. Hoy fácilmente disponible a través de emuladores Mame 32 y Mame Plus y adaptaciones a PC Engine, Game Gear, ordenadores X68000 de 32 bits y la Virtual Console de Wii.

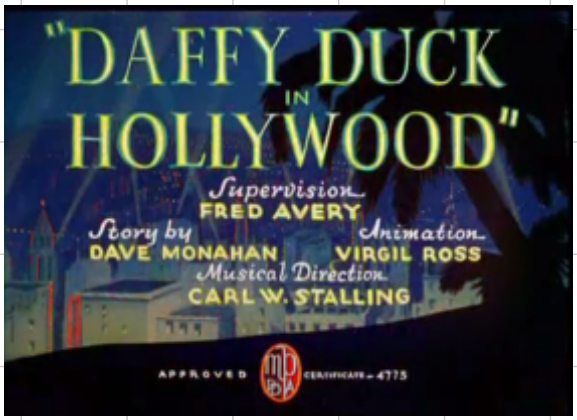
**Roberto Mérida Fernández ("HrodXpectrvm")**

SEGA®

# Master System




***Muchos de nosotros conocemos los juegos basados en personajes de disney. Aquí tenemos la otra cara de los dibujos animados, en este caso es el pato Lucas que se mete en el papel de detective privado y contratado por SAM, tiene la misión de encontrar las estatuillas de oro robadas. Estas estatuillas están escondidas a lo largo de varios platos de cine, a donde iremos en helicópteros y descenderemos en paracaídas, así comienza nuestra aventura eligiendo en que plato de cine comenzar la búsqueda...***




El juego que hoy nos ocupa coge su nombre de una producción de Merrie Melodies en 1938, para la Warner Bros. Pictures. Cortometraje de dibujos animados en tecnicolor de unos 8 minutos de duración donde un Pato Lucas de imagen primigenia pone de los nervios a un director de cine con sus locuras.

Pasemos a tema, La versión que vamos a comentar aqui es la creada para la master system, aunque existe una para megadrive con mejor acabado, no es la que nos ocupa. Hablamos de la versión 8 bits Manejamos al pato Lucas, que con su pistola de burbujas busca unas estatuillas a lo largo de unos niveles localizados en distintos platos de cine.

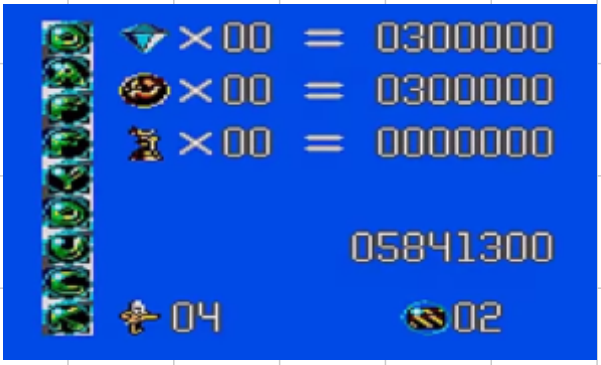
El juego en si es un plataformas 2D clásico, iremos explorando los distintos escenarios y al encontrar algunos objetos nos darán paso libre para ir mas alla y poder terminar el nivel. Antes de comenzar cada nivel nos deja elegir donde vamos a jugar, manejamos un helicóptero que nos dejara en el escenario que de deseamos jugar, al acabar todos, pasaremos de nivel.

El juego es del año 93 así que pertenece a la ultima época de la master system, esto se traduce en que esta explotada al máximo, tanto en gráficos como en la música.

Así podemos disfrutar de gráficos animados en la intro.
Otro detalle inaudito unos años antes es tener la posibilidad de tener los textos en varios idiomas y en especial en español. Se ve que en esta época ya se empezaba a tener en cuenta el mercado de habla hispana.


Nuestro personaje cuenta con algunas ventajas para enfrentarse a diversos peligros en su búsqueda, a parte de su bazoca de burbujas, tenemos la posibilidad de ir mejorandola cogiendo distintos objetos a lo largo de los escenarios. Estos objetos van saliendo del suelo al pasar sobre ellos y se van elevando poco a poco, así que hay que estar atento y no ir muy rápido para no dejarnos atrás nada que después no haga falta.

Otro objeto seria una especie de varita mágica que hace que aparezca una especie de ángel de la guarda que nos proporciona un toque adicional, de esta manera si nos dan no nos mataran, pero el ángel desaparece, aunque tendremos la posibilidad de encontrar otro mas adelante.

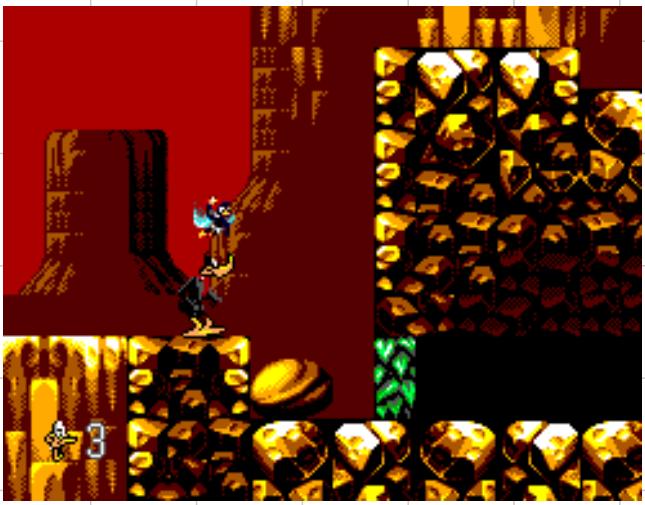



Por el escenario nos vamos encontrando letras que podemos ir acumulando al finalizar la pantalla nos darán puntos extra. las letras forman las palabras Daffy Duck.

Otros objetos serian los diamantes, estatuillas y unas bolas negras.

El control del personaje es el habitual en un plataforma aunque destacaremos dos puntos, uno que se puede colgar de algunas plataformas y lianas y otro que si dejamos el botón 1 pulsado lucas correra como loco.

Los escenarios, pues tienen un gran acabado y detalle, para una consola de 8 bits es lo mejor que se puede ver, un buen aprovechamiento de la paleta y de los titles en todo el mapeado.




Comenzamos en el desierto, esquivando catus y


enemigos.
En la segunda nos adentramos en un cementerio con fantasmas y pasadizos...


Para la tercera nos movemos por el bosque encantado de Robin hood

En la siguiente pantalla nos vamos a una especie de bosque de bambú, donde podemos distinguir algunos templos japoneses. Nos enfrentamos a unas especies de tortugas con pinchos y lo que parecen monos o ardillas...quien sabe...

Esta pantalla nos traslada al siglo 24 y medio, aunque mas parece que nos hemos trasladado a una película de serie b de los 50 sobre Marte.

Ahora nos encontramos en una especie de ruinas de una civilización en lo mas profundo de la selva.

Dependiendo del nivel en el que hayamos juagado pues este juego tiene la posibilidad de ver varios finales.

Hay tres finales a este juego. El final bueno conseguido al jugar el juego en dificultad difícil y recoger todas las Cartoon Movie Awards de oro en el camino. El primer final malo es por completar el juego en fácil o dificultad media, y el otro final malo es por terminar el juego en difícil sin la recopilación de todos los Cartoon Movie Awards de oro.


**DAFFY DUCK IN HOLLYWOOD**

Master System
También disponible en Genesis Sega Game Gear

Género: Plataformas

El Pato Lucas se embarca en una aventura para ir a Hollywood, está decidido a ganar un óscar. Ayúdalo a lograrlo a través de diversos y entretenidos episodios

Desarrollador: Probe
Distribuidor: Probe

Fecha de lanzamiento: 1 / Junio / 1993

# SAM Coupé

**Lanzamiento dicembre 1989**

**CPU:** Zilog Z80 a 3,56 MHz.
**RAM:** 256 kb de serie. Ampliable hasta 4,5 MB
**Gráficos:** 256 x 192, 512 x 192 128 colores
**Rom:** 32 kb (BASIC, BIOS y arranque de disco)
**Almacenamiento:** Disco 3,5
**Puertos:**
RF de salida de vídeo (UHF, canal 36), salida de vídeo SCART RGB, serial socket / red, conector de joystick (estándar Atari), interfaz paralela, entrada de audio, salida de audio

MILES GORDON TECHOLOGY plc

# Inves Spectrum+

**Lanzamiento Diciembre 1986**

**CPU:** Zilog Z80A 3,5 Mhz.
**RAM:** 64kb. solo usa 48 Kb.
**Gráficos:** 8 colores con dos niveles de intensidad
**Sonido:** ULA Texas Instruments
**Rom:** 16 Kb. Incluyendo Sistema operativo y BASIC
**Almacenamiento:** cinta de audio, 1500 baudios
**Puertos:** Expansión, joystick


investronica

# RetroWiki
## Magazine

Retrowiki Magazine te necesita, no te vamos a pedir dinero... Retrowiki magazine la hace gente como tú, así que no te lo pienses más ponte en contacto con nosotros en retrowiki.es y colabora con tus artículos.

Gracias.


REGISTRATE Y PARTICIPA

www.retrowiki.es

RW
Magazine