

El Inves Spectrum+, a fondo.

Miguel Ángel Rodríguez Jódar (mcleod_ideafix, @zxprojects, <http://www.zxprojects.com>)

César Hernández Bañó (chernandezba, chernandezba ARROBA hotmail NOSPAM com, <https://sites.google.com/site/chernandezba/home>)

Artículo publicado originalmente en el número #11 de la revista Retrowiki Magazine. <http://www.retrowiki.es/rw10/page.php?3>

Introducción

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 pensarán: “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 Pentagón.

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 íntegramente 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 Taiwán).

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 incrustara 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, después 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 baratos, 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 prácticamente 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 sólo 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 convertidor 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

(fuente: <http://www.worldofspectrum.org/faq/reference/48kreference.htm#PortFE>)

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.

Legendas 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 RANDOMIZEUSR 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 RANDOMIZEUSR. 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 protegen el micro de sobrecargas; sin embargo, es necesario utilizar ciertos puertos para abrir esos buffers y esto 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 buffers 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 buffers 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 buffers. 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 attribute 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 solo 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 más difícil su lectura.

El siguiente descubrimiento sobre el Inves afectaba a la gestión de interrupciones. Recuerdo que tenía información de cómo activar las interrupciones IM2, y se comentaba que la CPU leía el vector de interrupciones desde la dirección $I * 256 + \text{valor bus de datos}$... 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 sería 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 más...

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 más de 48k (aunque sí que tenía más 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 más) 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 descriptado 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 vigila 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 más tarde, un amigo mío, 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 más 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 ZX Spectr.

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 reales

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:

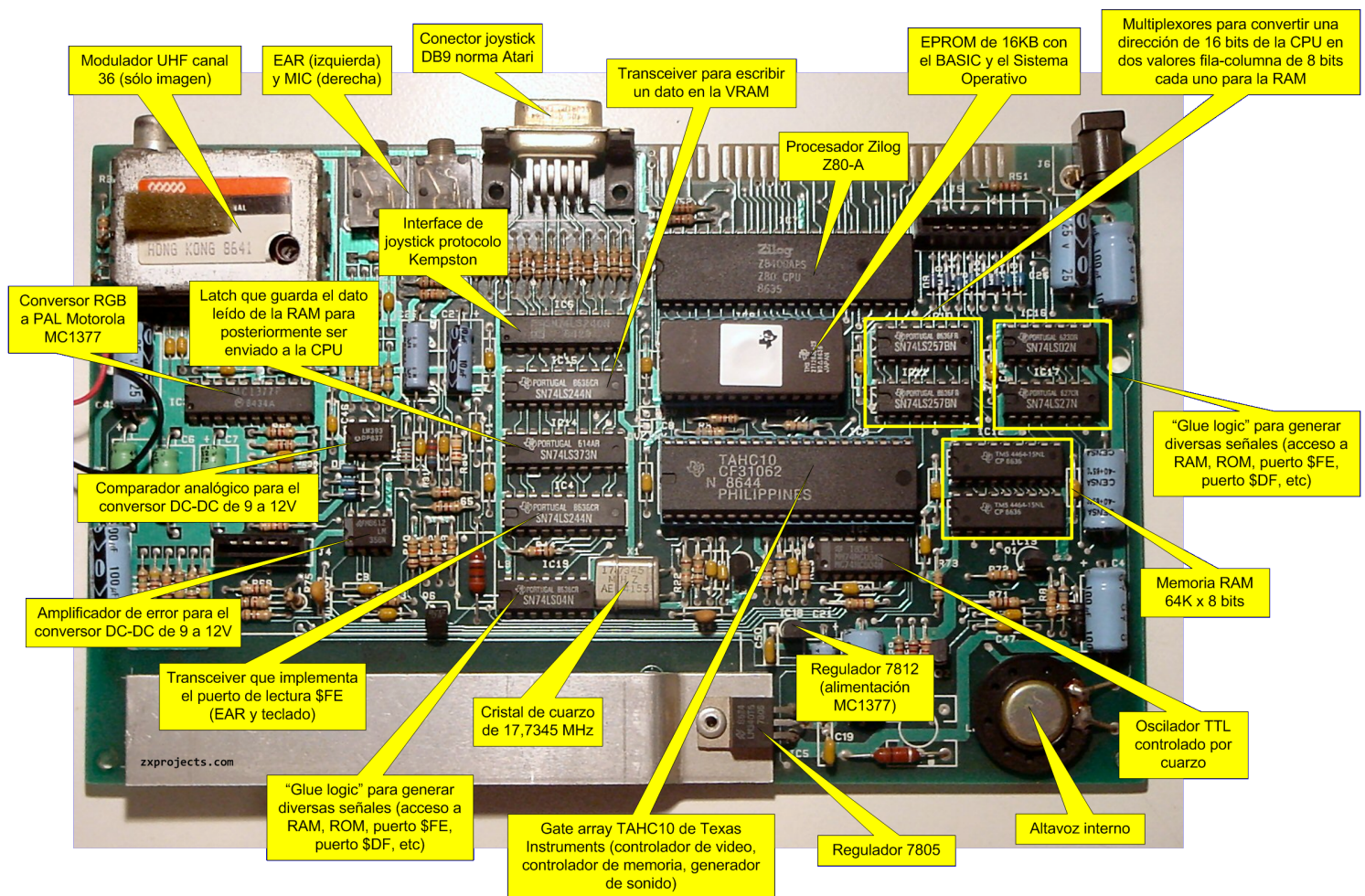
- Primer modelo de Spectrum 48K con 64K de memoria RAM real de 150ns. **Es posible escribir en cualquiera de las 65536 posiciones de memoria de esa RAM, y el dato se guardará**, aunque lamentablemente, cuando leamos alguna posición comprendida entre 0 y 16383, lo que obtendremos será lo que haya en ROM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

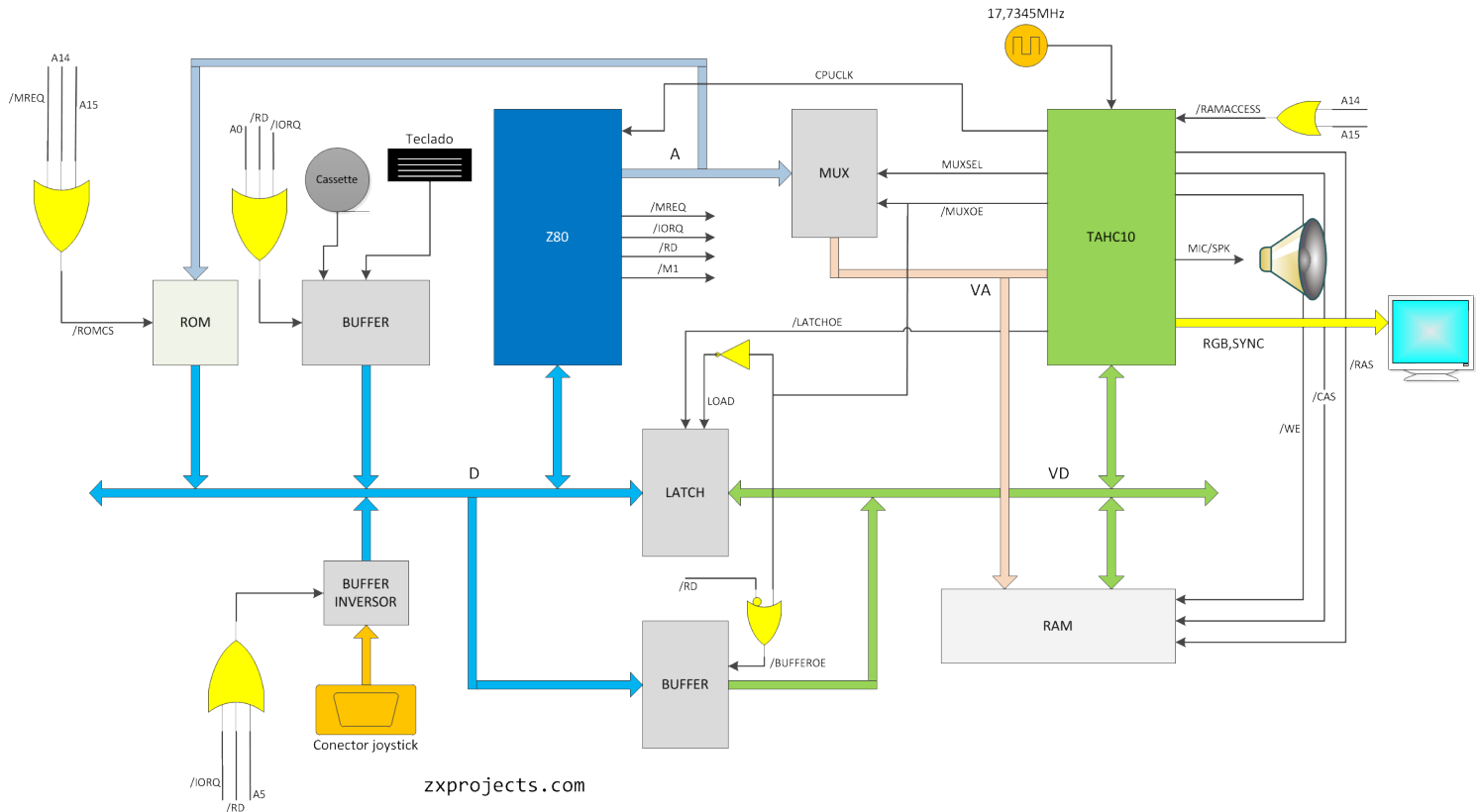
Esquema de bloques del ordenador

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:



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.



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 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 escritura 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.

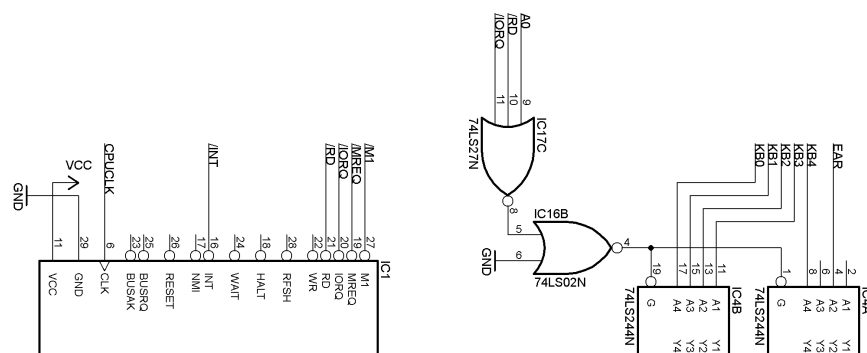
El circuito integrado específico TAHC10 será objeto de una sección aparte, pero comentar que se encarga prácticamente de lo mismo que la ULA en el Spectrum: generación de la imagen y del sonido, temporización y acceso a la RAM, generación de la señal de interrupción y reloj para el Z80, y generación 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.

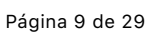
Para obtener la información que se presenta en las siguientes secciones se ha analizado un Inves Spectrum+ usando un analizador lógico de 32 canales, y un DivIDE para lanzar programas rápidamente, así como para poder ejecutar código arbitrario tanto en la zona de ROM como en la de RAM. El objeto principal de estudio ha sido el TAHC10, y en él se han usado la mayoría de las sondas del analizador.

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+

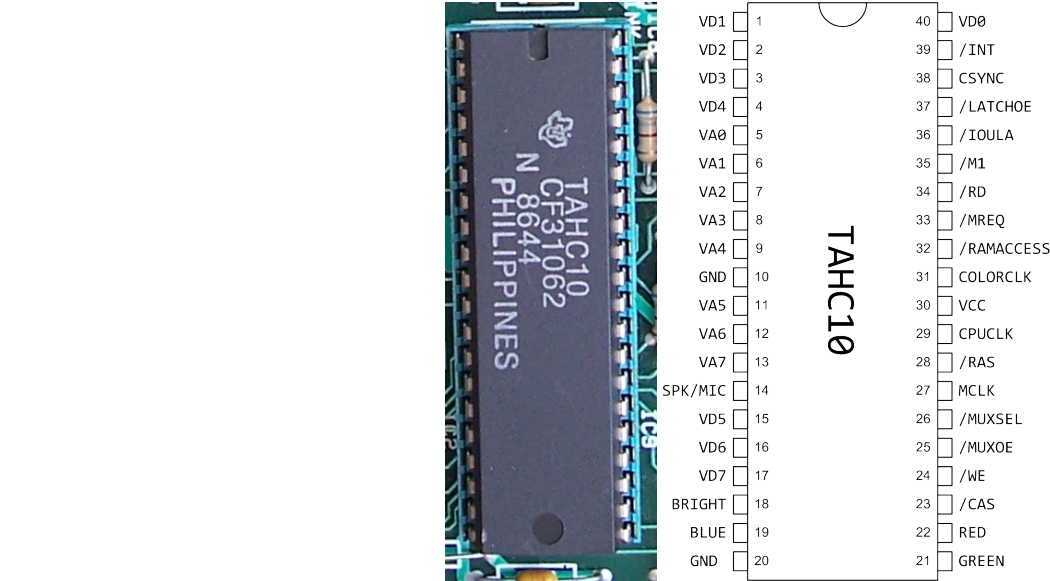
El TAHC10 de Texas Instruments

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 hacia 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:



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 convertor 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 direcciona. 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
/RAMACCESSIN		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 señalice 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 prácticamente 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.

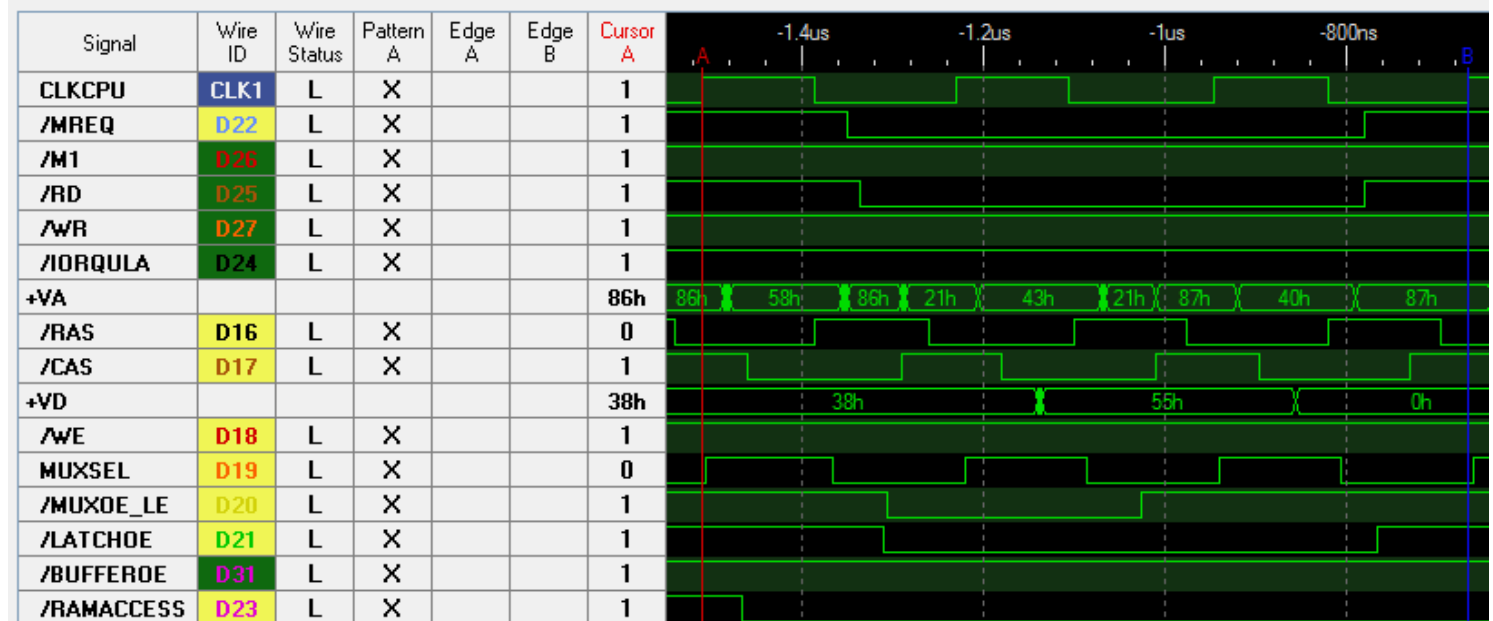
El mecanismo para la resolución de la contienda implica al propio TAHC10 y algunos de sus componentes. Al igual que el mecanismo de contienda desarrollado por Altwasser para la ULA del ZX Spectrum, el desarrollado para el TAHC10 denota un amplio conocimiento de la temporización de los buses en el Z80. La parte del circuito que nos interesa se ha destacado en la siguiente figura.



Página 12 de 29

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.

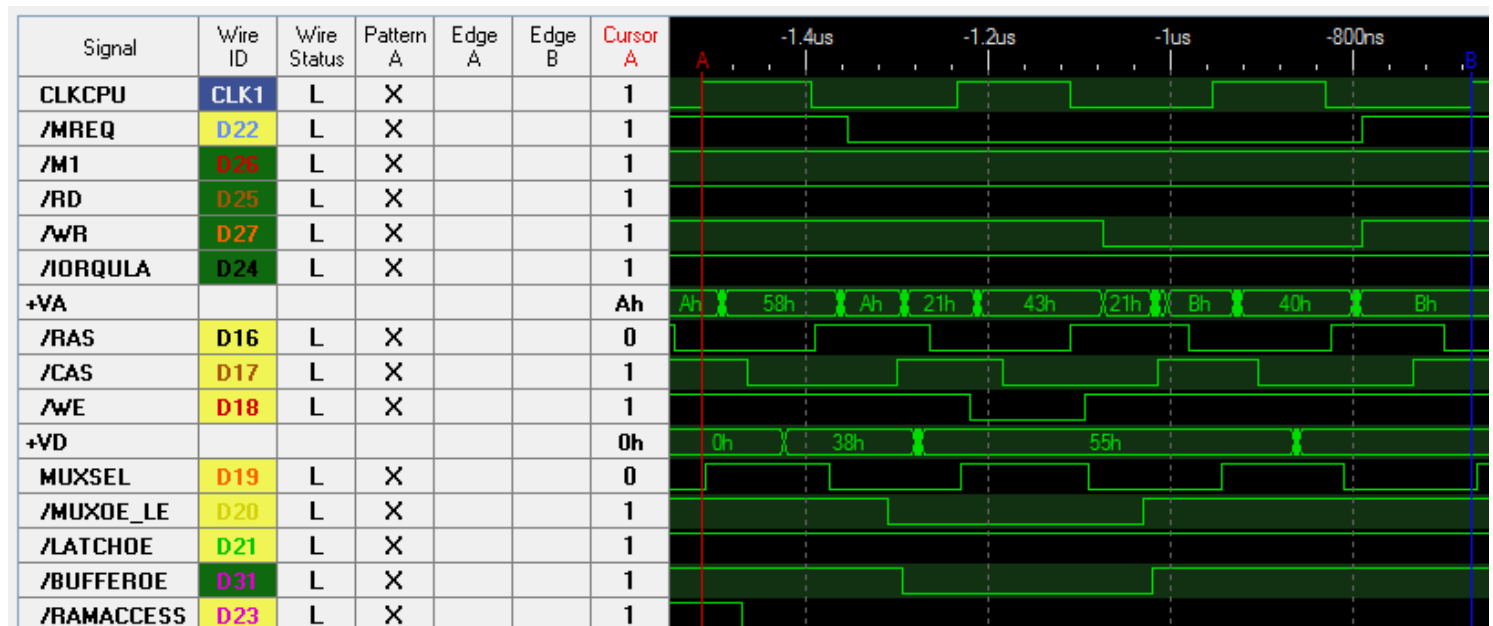


Cronograma de un ciclo de lectura de datos a memoria RAM

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).

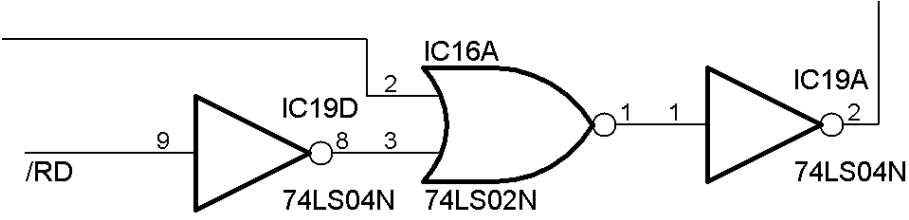
Cuando /MREQ y /RD se activan, lo hace también /MUXOE, permitiendo que la dirección multiplexada proveniente del Z80 llegue a la RAM. MUXSEL envía primero la fila y luego la columna, asegurándose de que ambas están presentes en el bus de direcciones de la memoria cuando se activan /RAS y /CAS respectivamente. /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

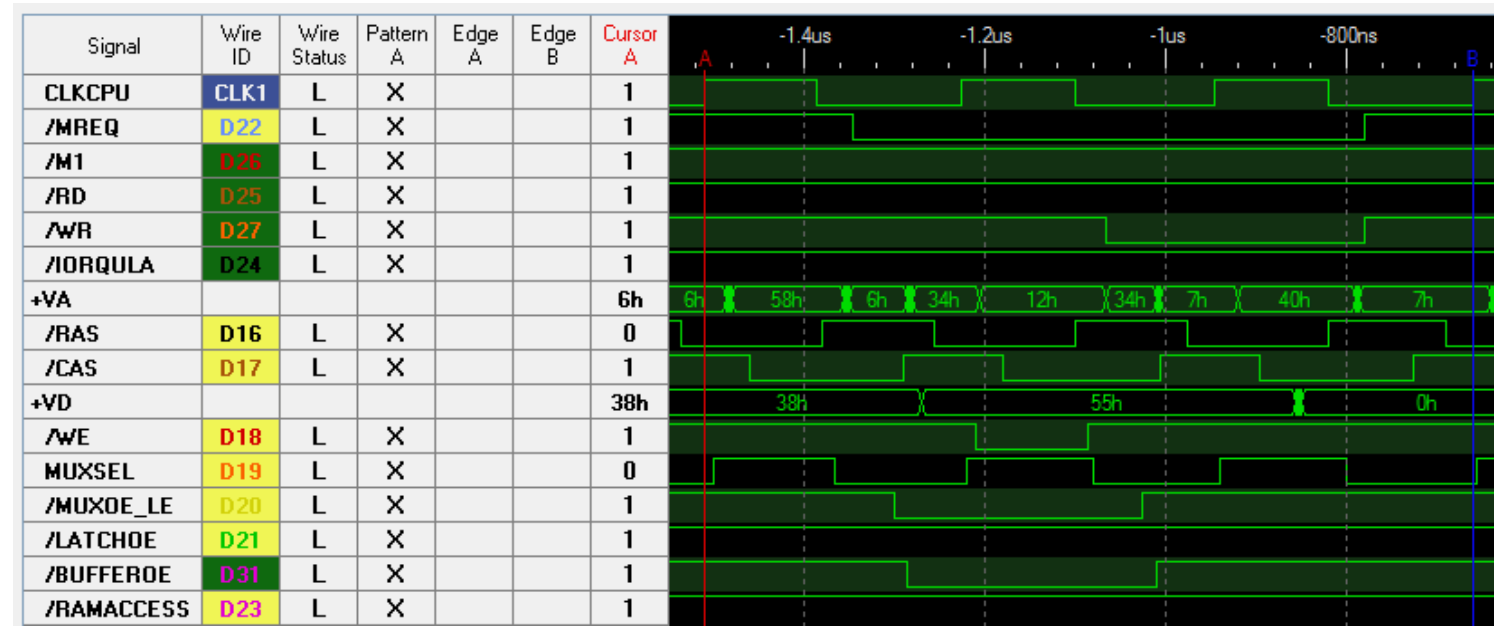
Como en el caso anterior, en cuanto se activa /MREQ y se sabe que esto es un acceso a RAM (se activa /RAMACCESS), el TAHC10 activa /MUXOE y comienza la selección de fila y columna a la RAM. En este caso, y como no se ha observado que se haya activado /RD, el TAHC10 adivina que esto es un ciclo de escritura 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.

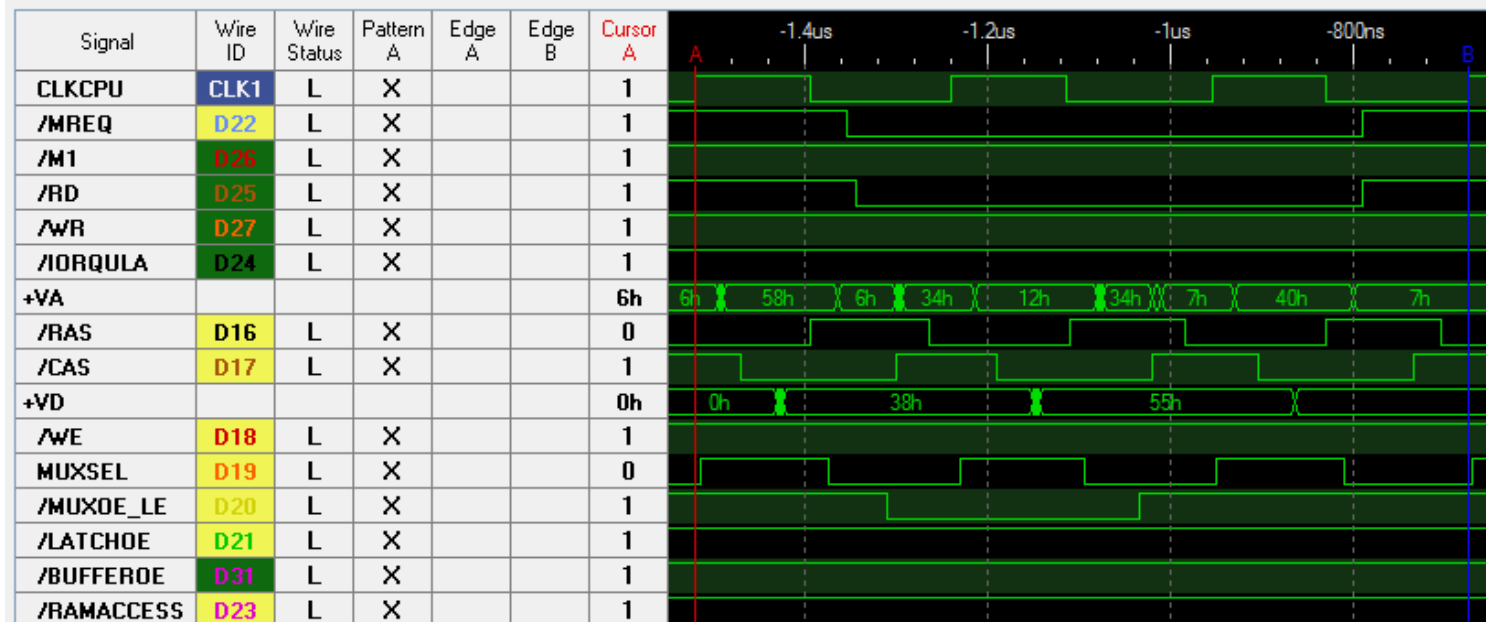
¿Y si la dirección de memoria en la que se quiere escribir pertenece al espacio de ROM? Esto es lo que se refleja en el siguiente cronograma:



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:

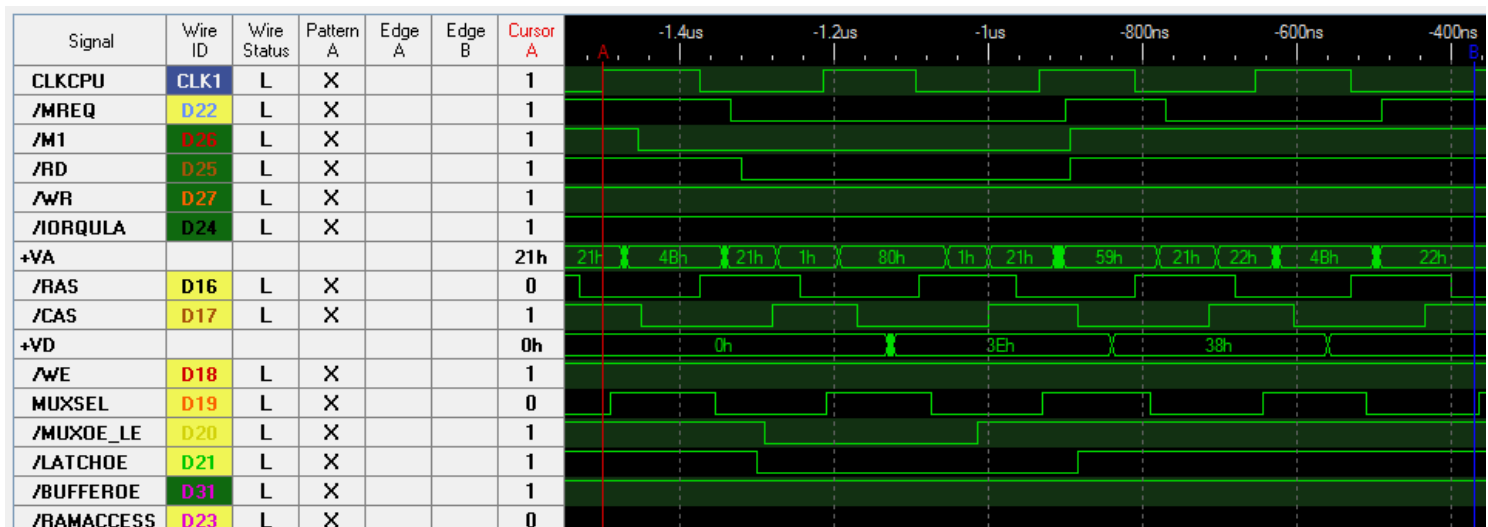


Cronograma de un ciclo de lectura de memoria en una dirección del espacio de ROM

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.

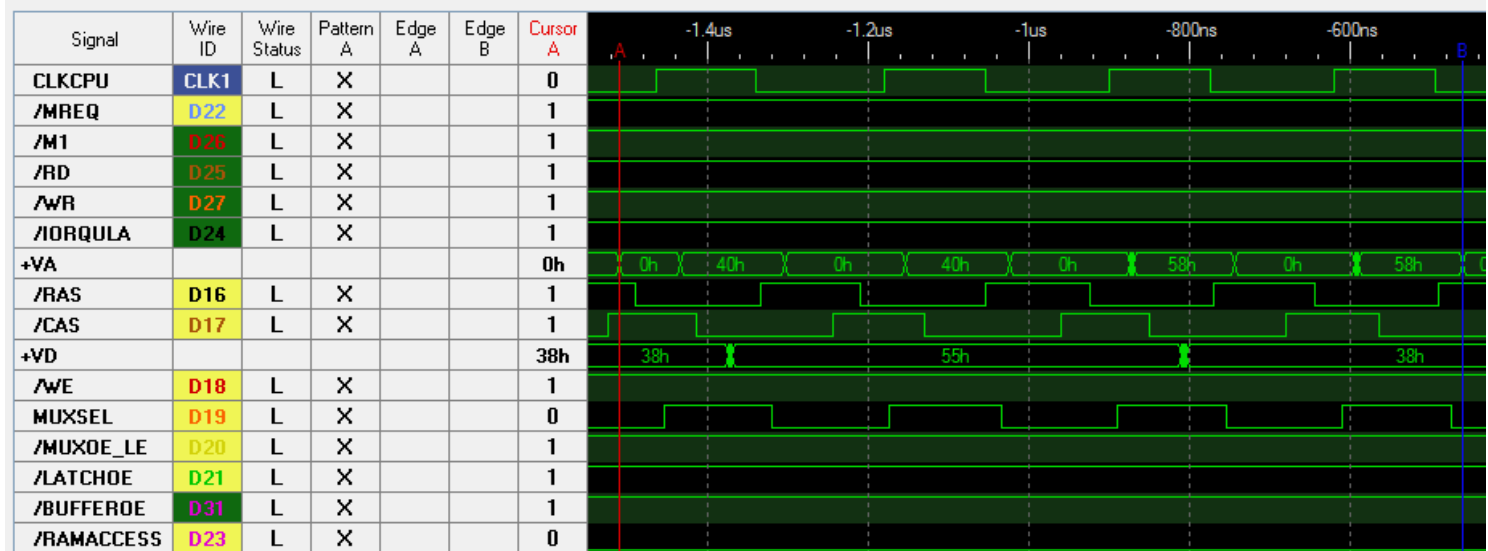


Cronograma de un ciclo de bus M1

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 dejándose 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:



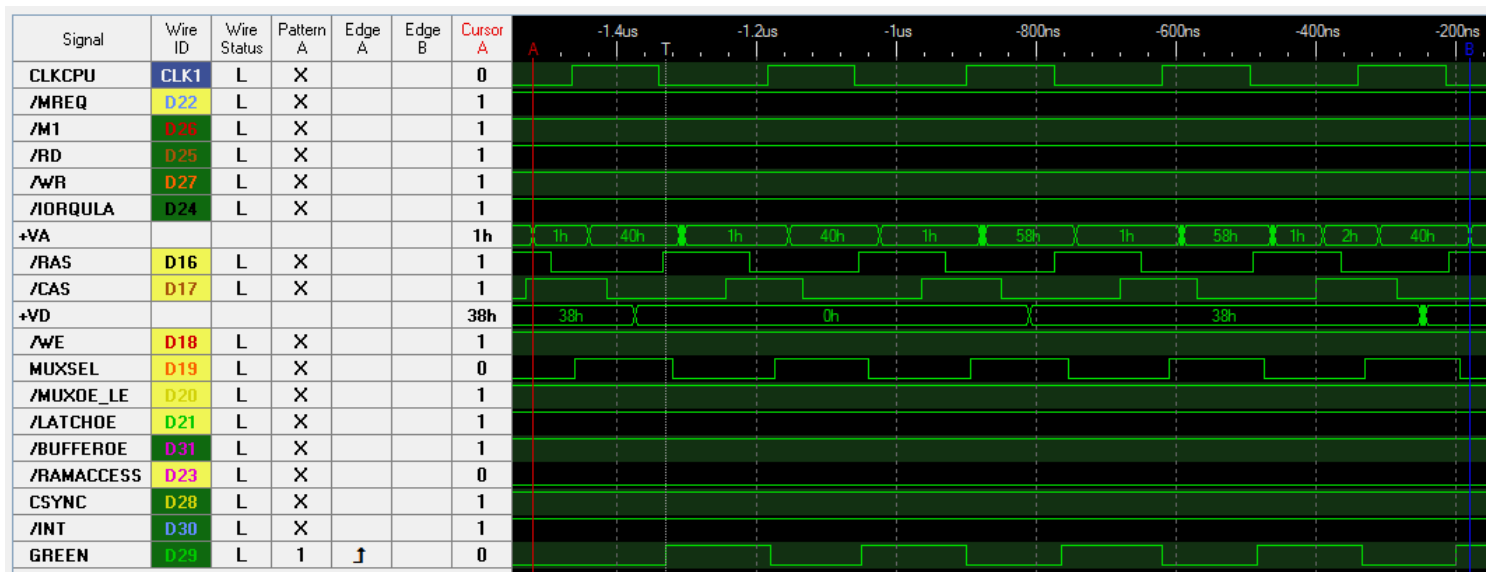
Cronograma de lectura de la dirección \$4000 y \$5800 por parte del TAHC10 para generar la imagen. El Z80 está desactivado.

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 columna \$40. El valor leído de ahí es \$55 que en binario es 01010101. Visualmente, una serie de píxeles 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 píxeles que ha leído en los ciclos anteriores.

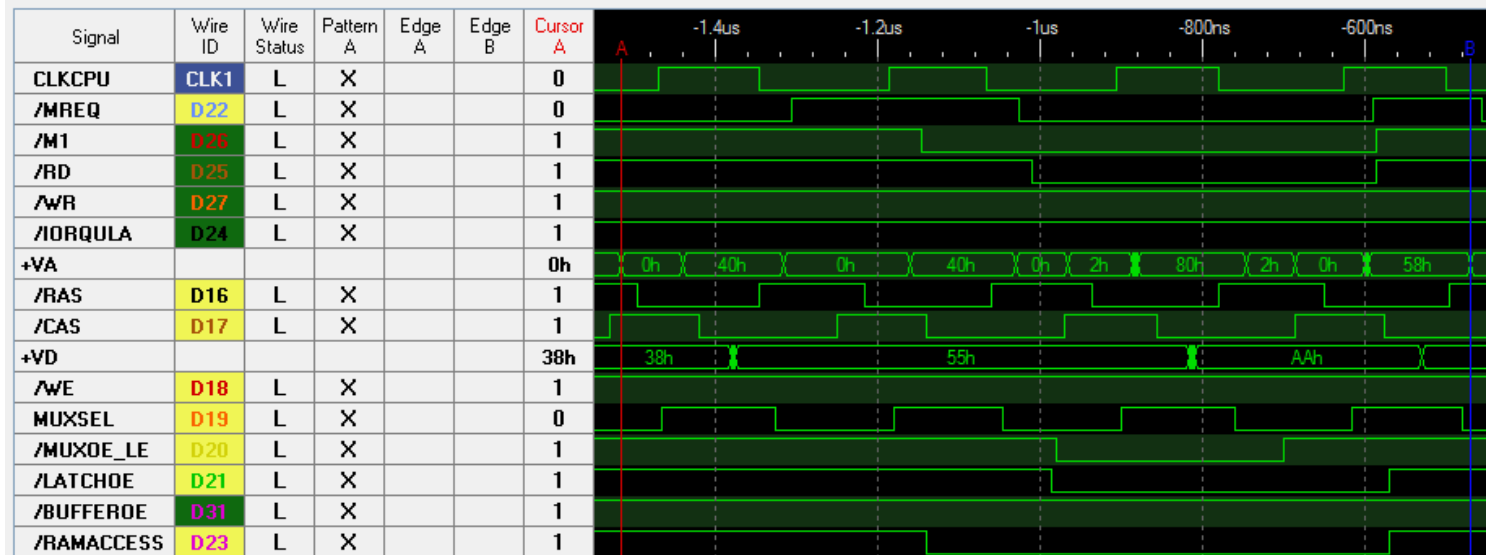


Cronograma de la temporización de los píxeles en el TAHC10

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.

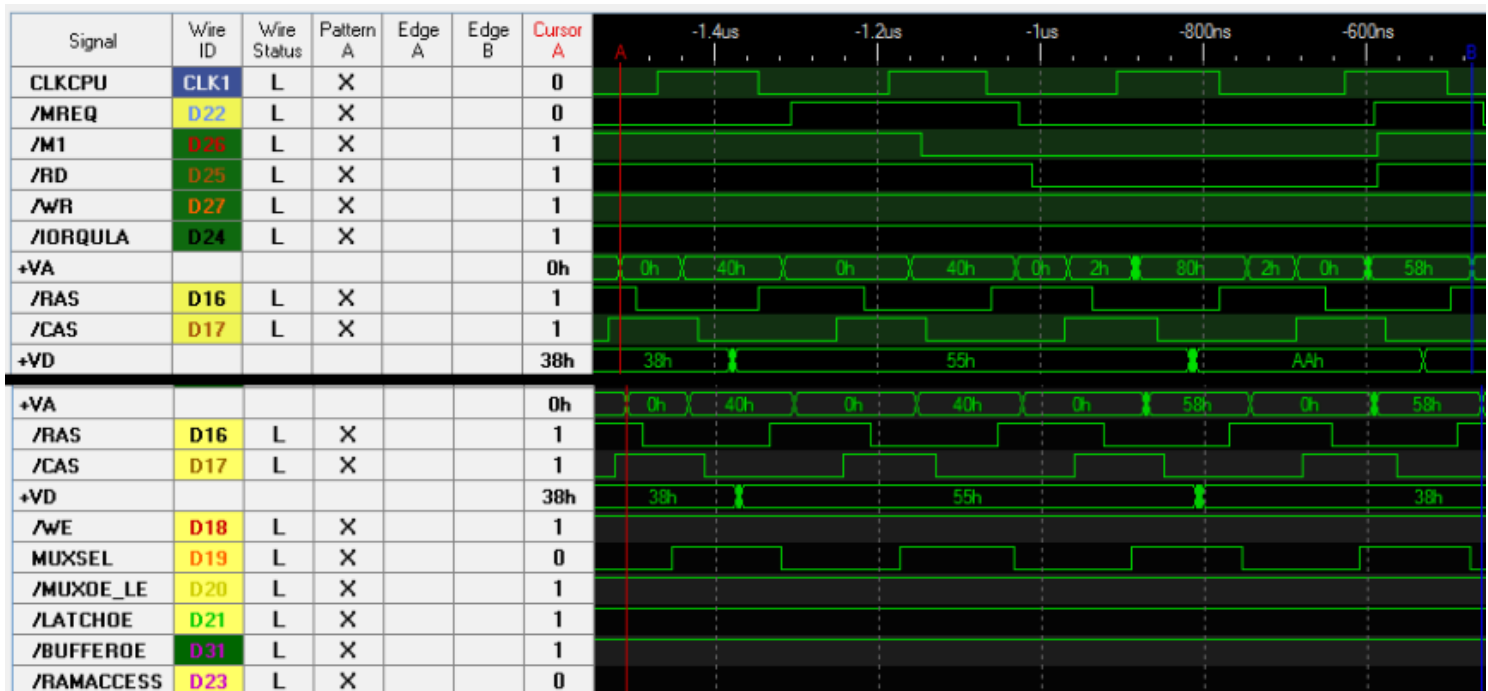
Pero, ¿por qué repite dos veces la lectura de cada dato si con leerlo una vez sería suficiente? Es ahora cuando entra la CPU en este juego:



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.



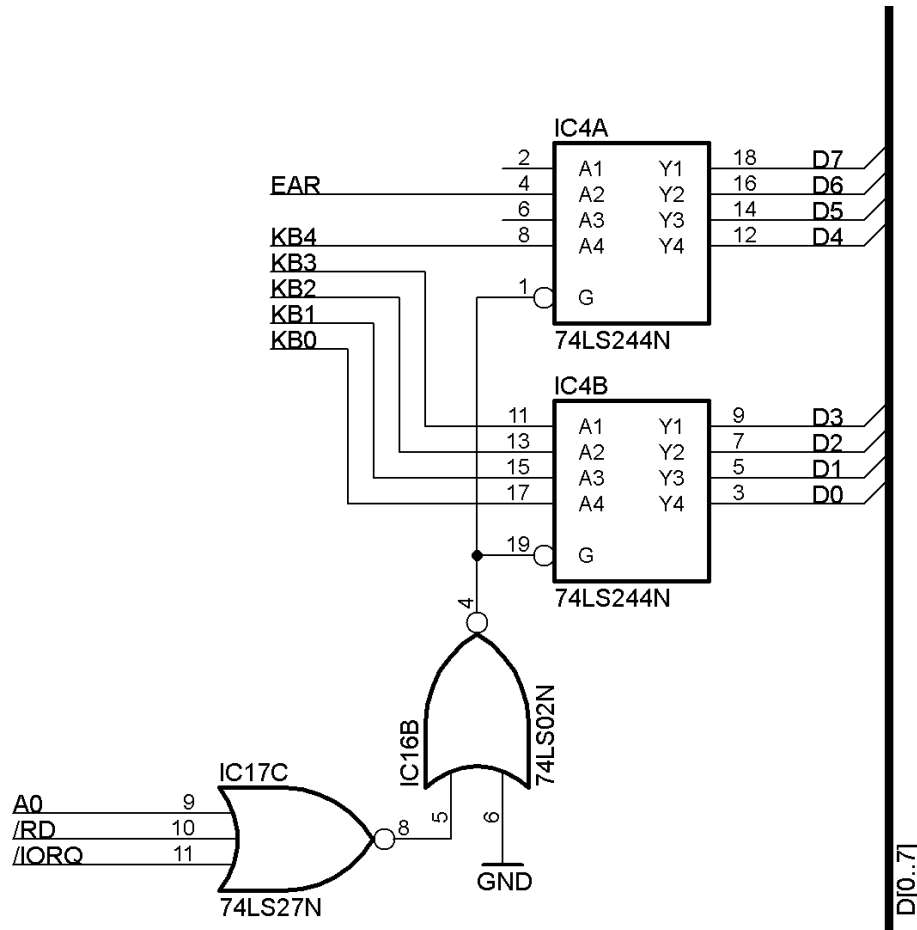
Comparación entre los ciclos de lectura a RAM por parte del TAHC10 compartiendo slots de acceso con el Z80 (arriba) y sin compartírselos (abajo)

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)

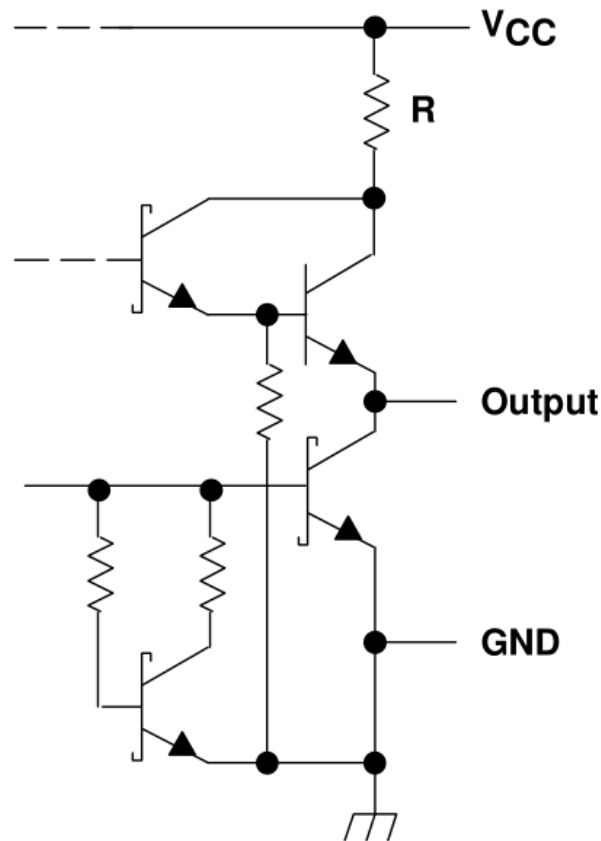
IC17 se emplea como decodificador para la condición anterior. IC16 se usa como inversor de forma que en el control de habilitación de triestado de IC4 tenemos la señal de habilitado solamente cuando se accede en lectura al puerto de E/S \$FE. De hecho, cuando se acceda en lectura a cualquier puerto de 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:

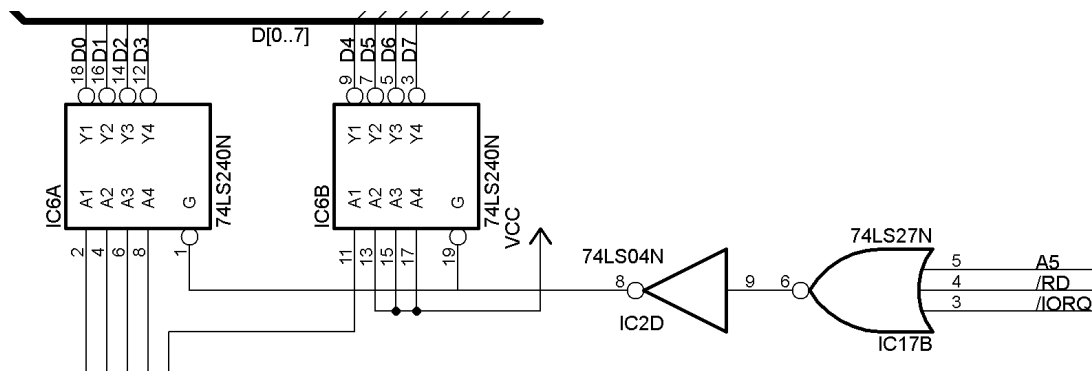


'LS240, 'LS241, 'LS244: $R = 50 \Omega$ NOM
'S240, 'S241, S244: $R = 25 \Omega$ NOM

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 TAH10. 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).

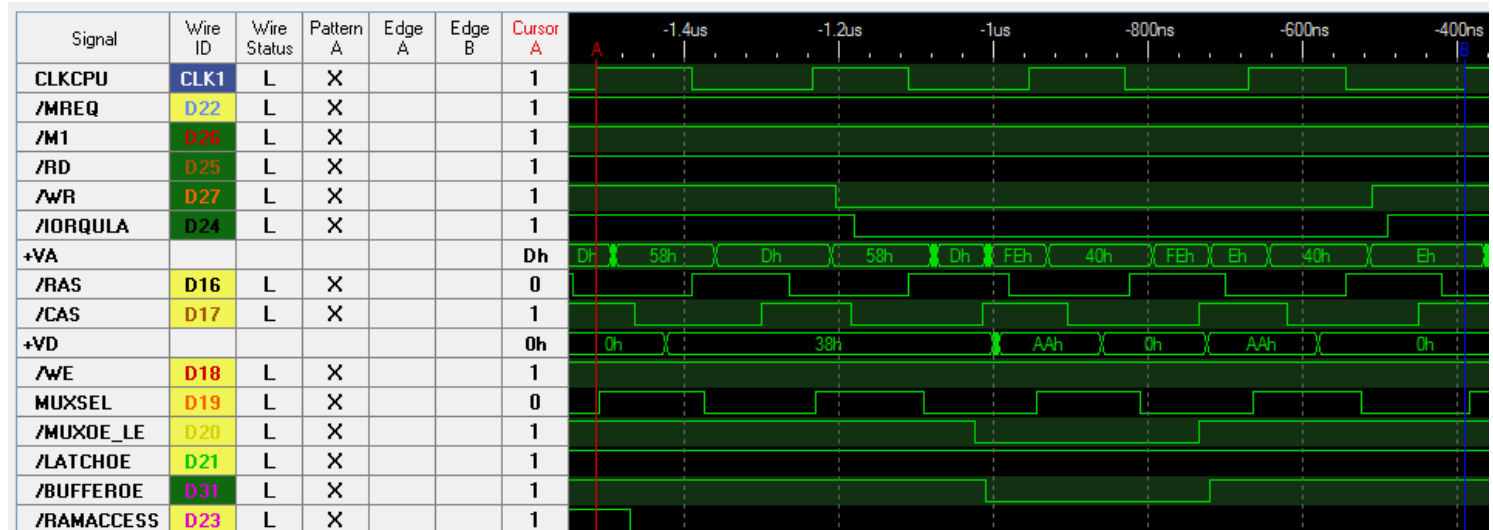
Puerto \$FE, en escritura. Este puerto está integrado dentro del TAHC10 por razones obvias: el color del borde y el altavoz se gestionan desde este chip. Dado que el TAHC sólo tiene un bus de datos y éste está aislado del bus de datos de la CPU, es lógico suponer que los buffers y latches que actúan 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.



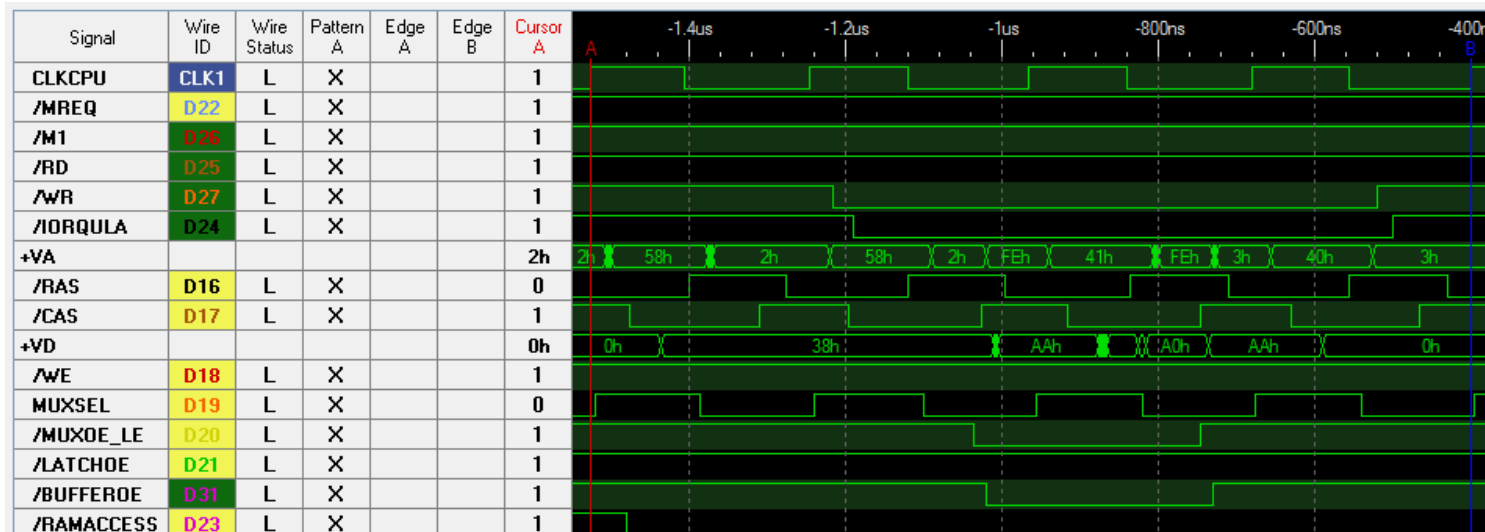
Cronograma de un ciclo de bus de lectura E/S al puerto \$40FE. En la dirección de memoria \$40FE se ha almacenado previamente el valor \$00

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).

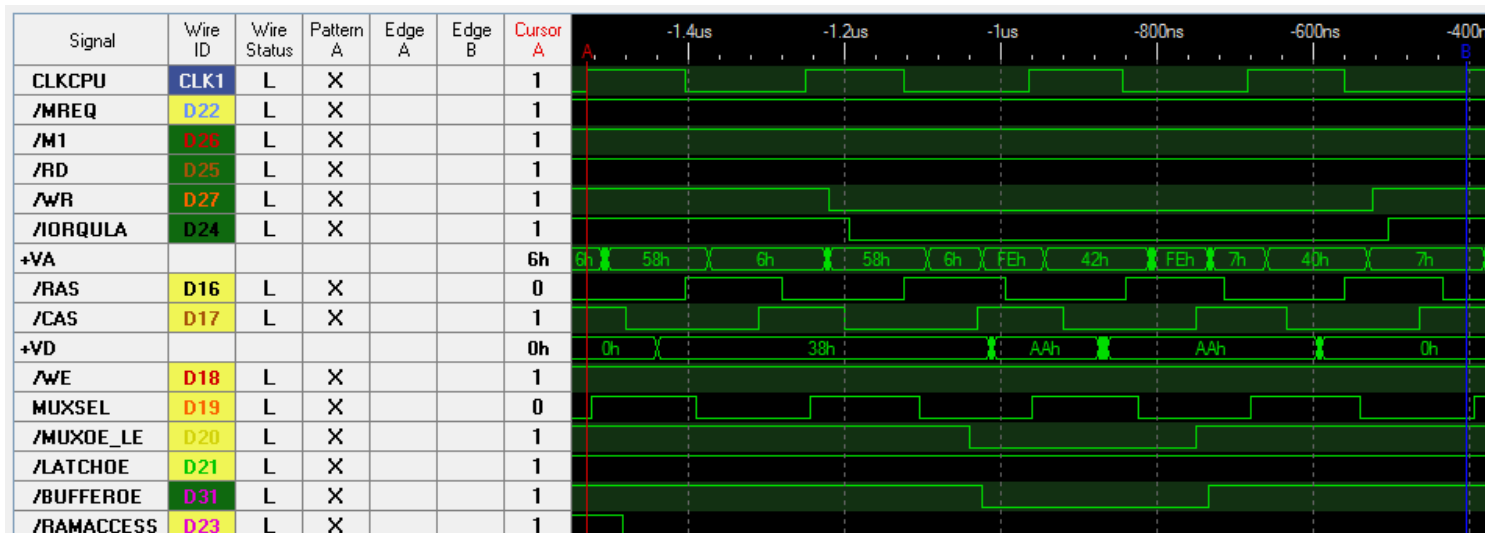
En el acceso al puerto \$41FE nos encontramos esta otra situación:



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

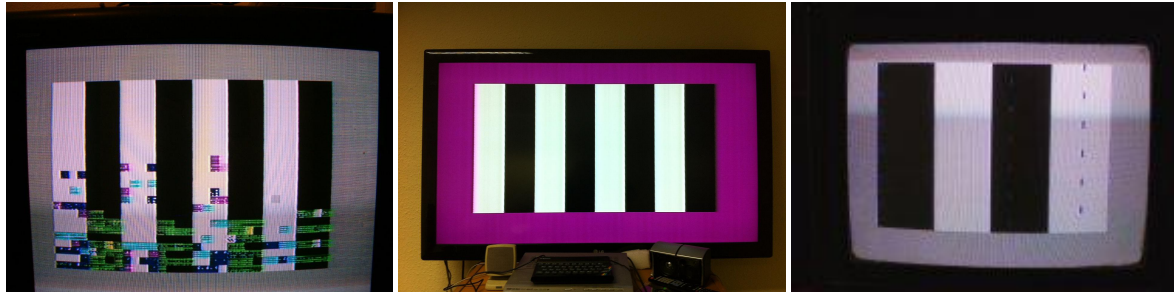
Hay un momento, dura unos nanosegundos, cuando después de que se active /CAS, la RAM actualiza su buffer interno para mostrar el valor pedido, en el que los glitches que se generan en sus líneas de datos hacen variar el valor que se muestra en VD, pero una vez que se estabiliza, el dato que vuelca, \$FF, 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 borrarla 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)

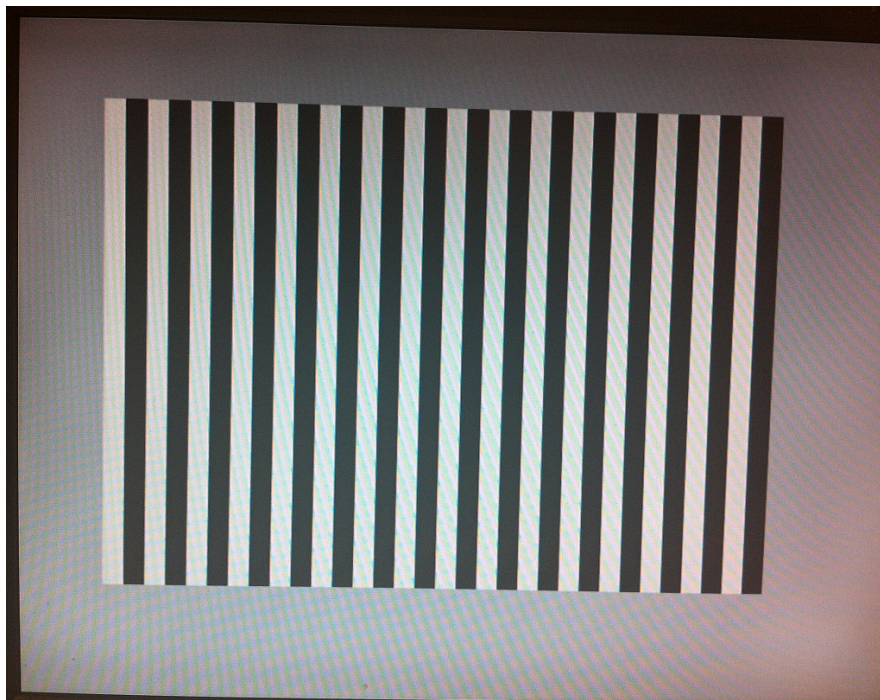
Según el fabricante y tipo de memoria que lleve, el patrón de franjas verticales es diferente. Pueden ser franjas muy anchas, de unos 8 caracteres de ancho como ocurre con algunos modelos +2A/3, o pueden ser franjas de unos 4 caracteres de ancho, como ocurre en la mayoría de Spectrum's de 16K y 48K.

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 prácticamente 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 conflicto 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.

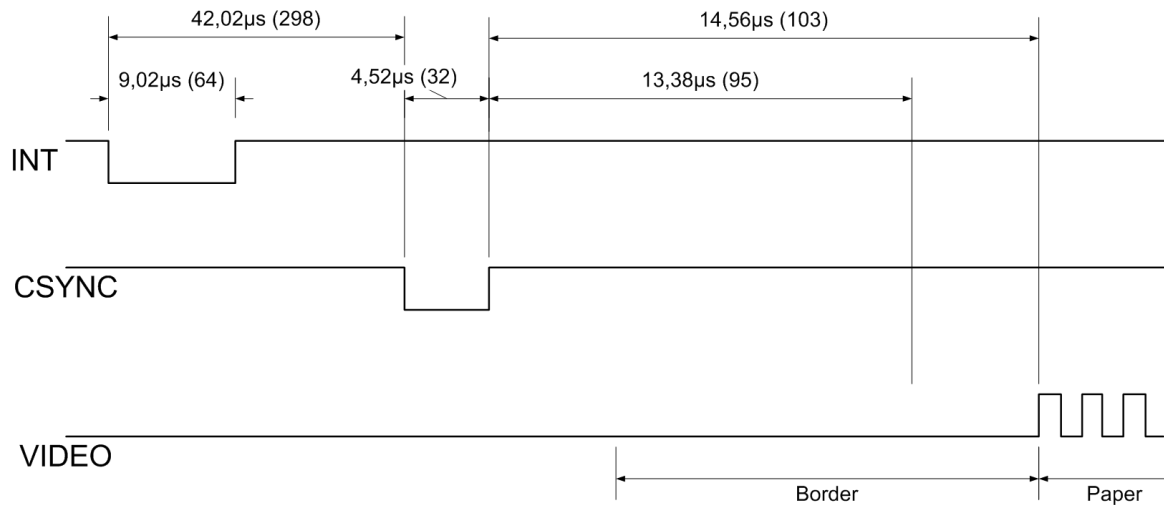
Claro está que si un programa realiza escrituras a ROM (por ejemplo como parte de una protección de carga para obtener un bloque de carga superior a lo que un copión de la época puede manejar) o bien por un efecto secundario de alguna rutina (por ejemplo para hacer un scroll vertical y no tener que chequear 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 retraso 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.

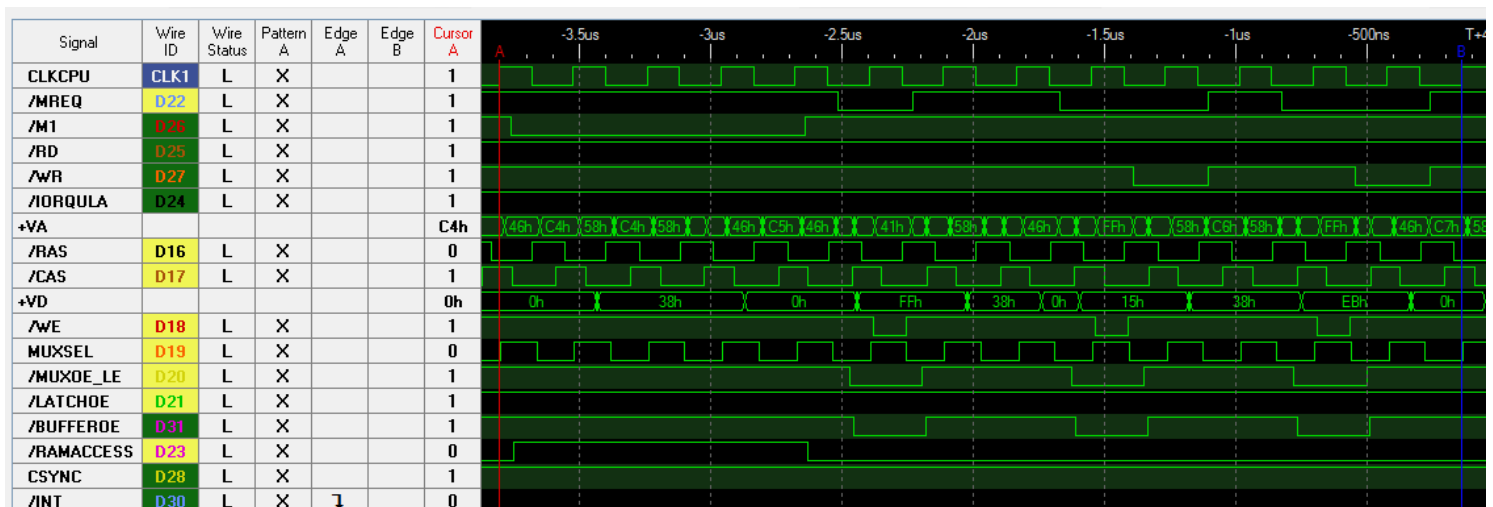
Los tiempos están dados en microsegundos, y entre paréntesis, en ciclos de reloj de pixel (ver más adelante la sección sobre la temporización de la pantalla). Si sumamos el tiempo en ciclos desde el flanco de bajada de INT hasta el momento en que se lee el primer byte de la memoria de pantalla (cota 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.

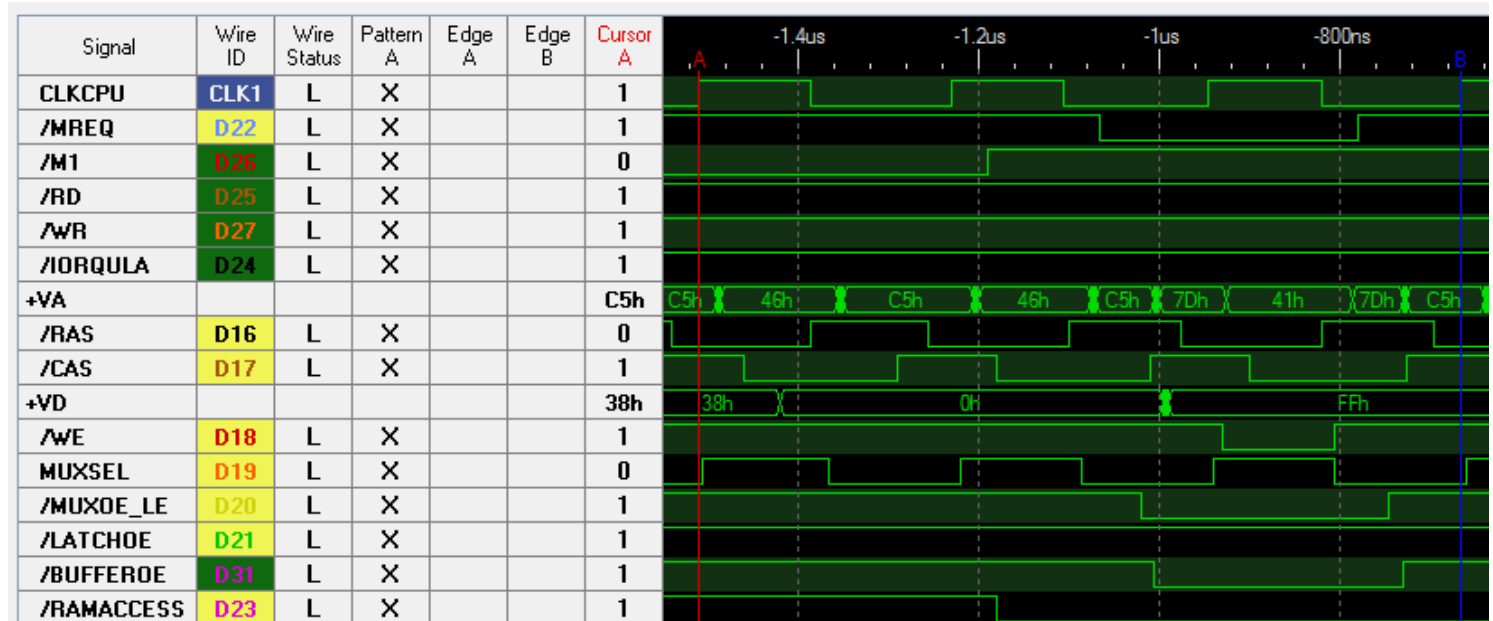
En el modo de interrupciones IM 2 se tiene además una tabla de vectores de interrupción. La dirección base de la tabla es de la forma \$xx00 donde \$xx es el valor del registro I. En el Spectrum, la parte menos significativa de la dirección la pone el dato que haya en el bus de datos en ese momento, 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 sabéis 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 adivinatoria 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 sobrescrito 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 sobrescritura 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). Automáticamente, 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=HiNAs0s0bI4>

No deja de ser curioso que este mismo programa, en un ZX Spectrum 48K, produzca el famoso efecto "ULA snow".

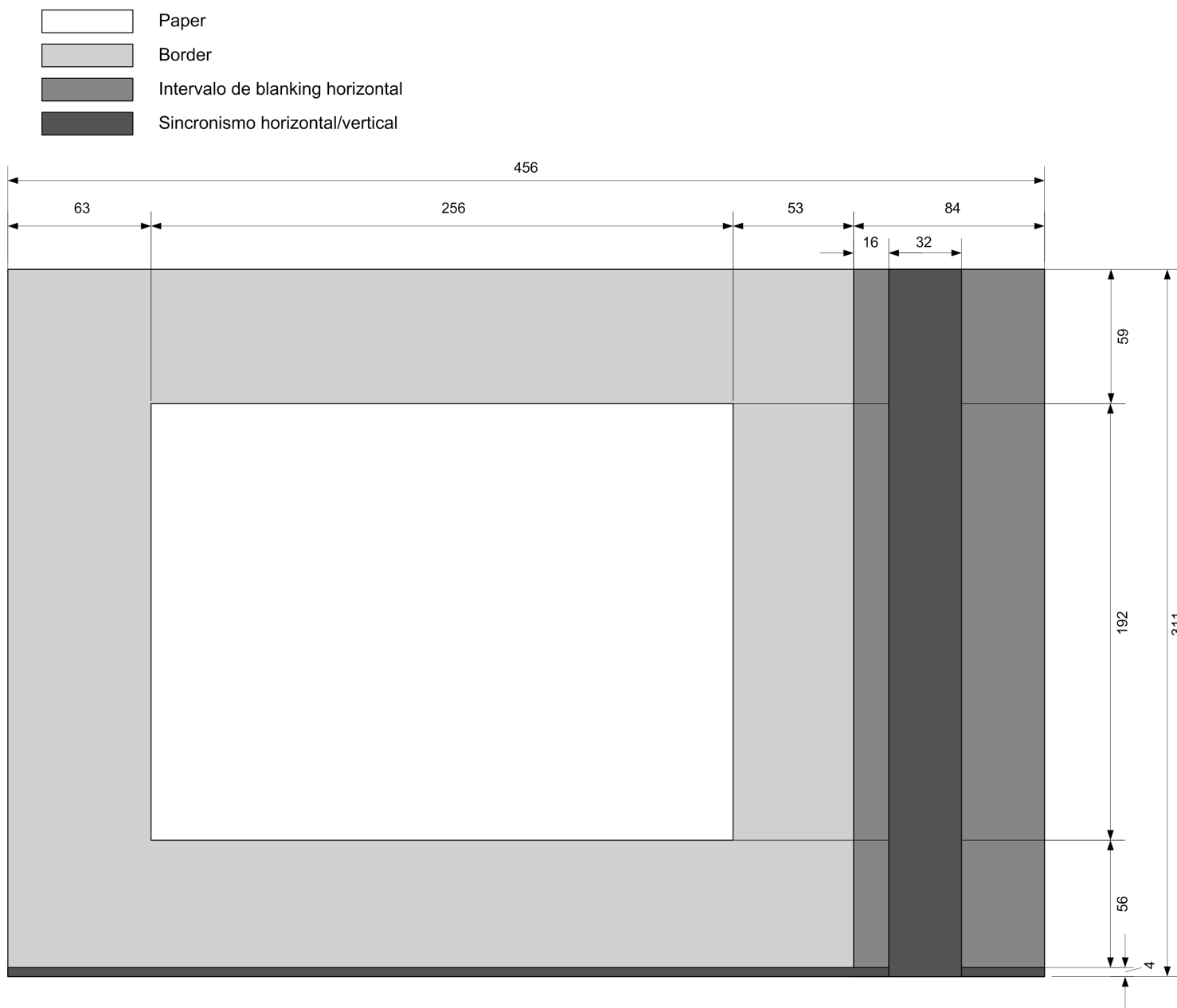
Temporización de la pantalla

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 radicalmente 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.

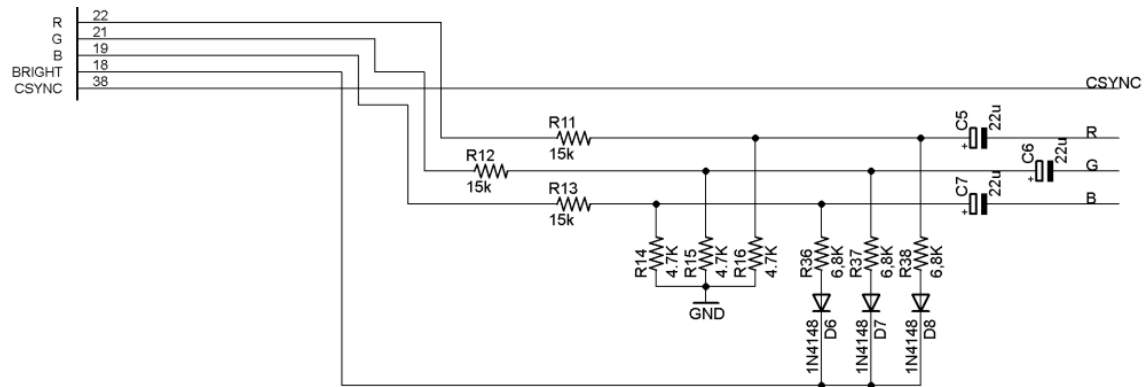
Salida de video compuesto por el conector de antena

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

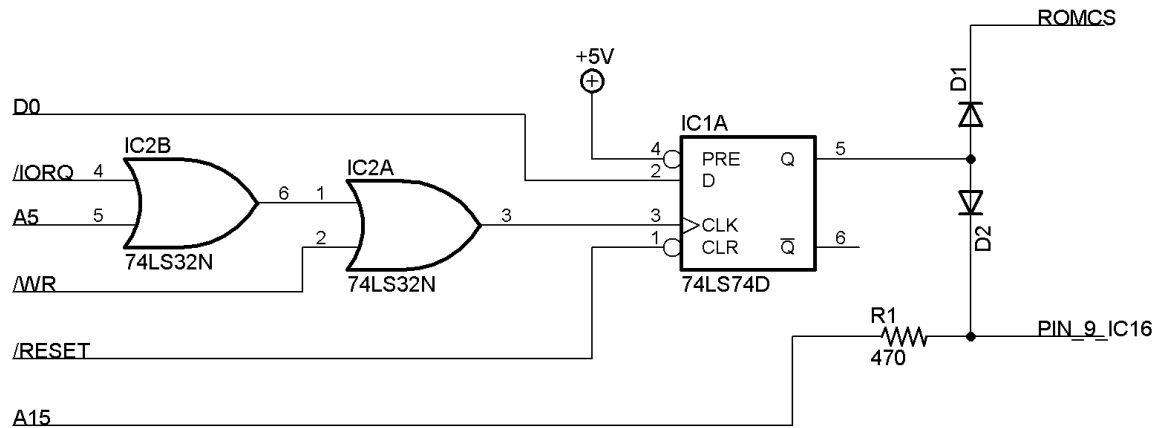
ret
```


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.

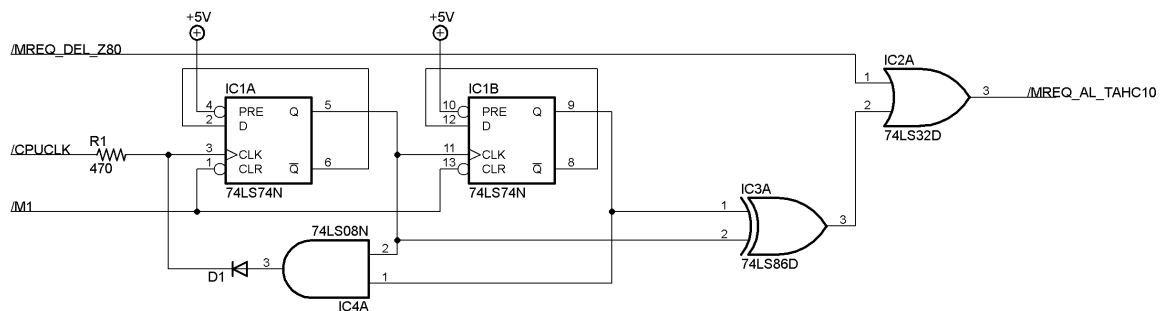
Este mod es compatible con un periférico que desactive a la ROM interna para poner la suya (ej: DivIDE, DivMMC, copiadores hardware, etc) siempre y cuando no se active el modo all-RAM.



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:

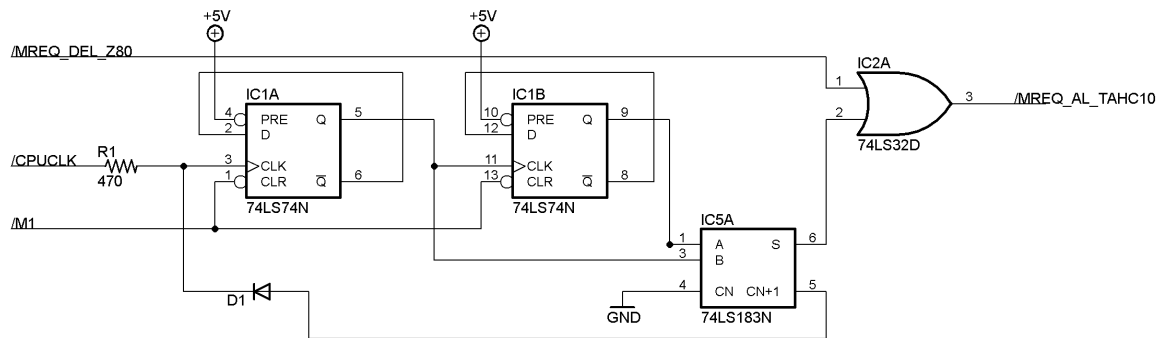


Aquí, los dos biestables 74LS74 forman un contador de 2 bits en cascada que cuenta ciclos de reloj. El contador se resetea a 0 con la señal /M1 y sólo cuenta cuando esta señal vale 1. El biestable A contiene el bit menos significativo del contador, y el biestable B, el más significativo. La señal de 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:

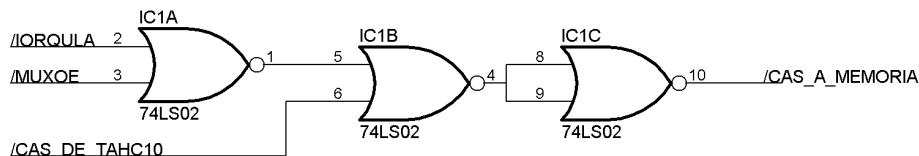


Reparación del fallo de enmascaramiento de datos en una operación de escritura al puerto \$FE

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 solamente 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 SFF para examinar qué está haciendo el TAHC10, y lo malo es que, a diferencia del +2A/3, es prácticamente 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 agri dulce. 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 cuadrado 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 Altvasser. 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.