Tutorial para la generación de archivos TSX

por Natalia Pujol (@NataliaPC @ishwin74)
2018.11.16 Primera versión
2019.02.09 Segunda versión

ÍNDICE:

1.- Nociones técnicas básicas de codificación de datos en cinta
- **1.1.-** Codificación de bits
- **1.2.-** Codificación de bytes
- **1.3.-** Bloques de datos
  - **1.3.1.-** Bloques cabecera
  - **1.3.2.-** Bloques de datos
- **1.4.-** El nuevo bloque #4B

2.- El programa makeTSX

3.- Pasos para realizar una buena conversión a TSX
- **3.1.-** Digitalizar la cinta a formato WAV
- **3.2.-** Usando el programa makeTSX
- **3.3.-** Control de errores
- **3.4.-** Restauración manual de la señal
- **3.5.-** Verificación de los datos
  - **3.5.1.-** Verificación de cintas con bloques MSX
  - **3.5.2.-** Verificación de cintas con bloques Spectrum

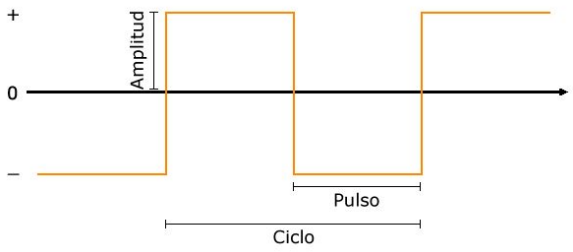
1.- Nociones técnicas básicas de codificación de datos en cinta

La mayoría de los antiguos ordenadores de 8 bits, para almacenar datos en cintas magnéticas, se ciñen a la técnica *FSK* (modulación por desplazamiento de frecuencia). Esta técnica se basa en el cambio de la frecuencia de la señal para definir dos o más símbolos. Más concretamente, en el caso que nos ocupa, serían dos: ceros y unos.

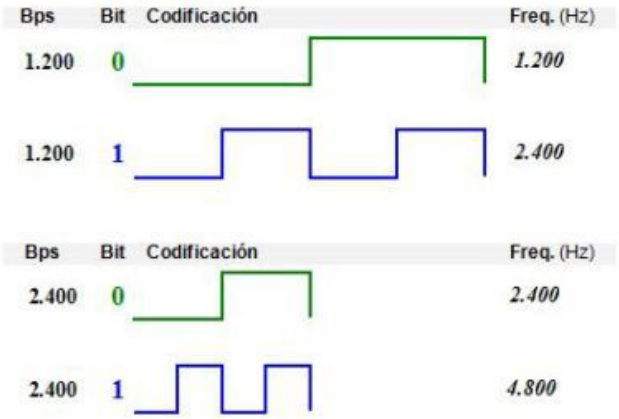
Una vez comprendido esto, cada sistema usaba su propio sistema de codificación: frecuencias, repeticiones de pulsos, sistemas de paridad y bits de control distintos. Aquí veremos el usado por los ordenadores *MSX* que son los que nos ocupan.

1.1.- Codificación de bits

Para codificar bits (ceros y unos), se emplean pulsos. Un pulso es un estado de la señal (estado bajo o alto), y dos pulsos crean un ciclo, tras el cual se obtiene de nuevo el estado de señal inicial.



Como podemos ver en la siguiente tabla, para codificar un bit a 0 se usan 2 pulsos mientras que para un bit a 1 se usan 4, durando ambos bits lo mismo[1] debido a su distinta frecuencia. Esto es así en el sistema *MSX*, a diferencia de la codificación *Spectrum* o la de los *SVI-318/328* por ejemplo, en la que un bit 0 y un bit 1 no tienen la misma duración.



1.2.- Codificación de bytes

En *MSX*, para codificar un byte se usa la convención *LSB* (*Least Significant Bit*) la cual consiste en que la aparición de bits se produce desde el bit menos significativo al más significativo, al contrario que en Spectrum que se usa *MSB* (*Most Significant Bit*) que invierte el orden de aparición.

Además de *LSB* se usan una serie de bits de control rodeando al byte llamados *bits de inicio* y *bits de parada* (*start/stop bits*) que permiten detectar errores y cierto nivel (poco) de autocorrección. En este caso concreto se usa un bit de inicio a cero y dos bits de parada con valor uno. En total, para un byte, tendríamos 11 bits de la siguiente forma:

```
0 b0 b1 b2 b3 b4 b5 b6 b7 1 1
```

Este tipo de codificación entra dentro del estandard *KCS*[(2)] (*Kansas City Standard*) que es la que adoptó el sistema MSX para grabaciones en cinta cassette. Hay diversas formas de codificación *KCS* incompatibles entre sí, por ejemplo los de los siguientes sistemas: *MSX*, *ABC 80*, *Acorn/BBC/Electron*, *MicroBee*, *Dragon/CoCo*, *SVI-3x8*, ...

1.3.- Bloques de datos

Los bytes obtenidos se agrupan en bloques de datos separados por silencios. Estos bloques cuentan con una serie de pulsos piloto que indicarán al ordenador qué velocidad en baudios se está utilizando en ese bloque y ayuda a sincronizar la lectura con esa velocidad.

En *MSX* hay dos tipos de pulsos piloto: largos (para los bloques cabecera) y cortos (para los bloques de datos). Ambos se codifican como si fueran una serie de **unos** consecutivos (con su misma frecuencia), pero se diferencian en el número de pulsos que contiene cada uno.

A continuación una tabla con información sobre los distintos pulsos piloto:

Baudios	Tipo de piloto	Frecuencia pulsos	Núm. de pulsos	Duración
1200	Corta	2400	7680	~1.5 seg
1200	Larga	2400	30720	~6.1 seg
2400	Corta	4800	15872	~1.6 seg
2400	Larga	4800	63488	~6.3 seg

Por otra parte, hay 3 tipos estándar de bloques de datos: *BINARY*, *ASCII* y *BASIC*. Se diferencian por un byte *ID* que se repite 10 veces en sus bloques cabecera y por la estructura de sus bloques de datos.

1.3.1.- Bloques cabecera

Composición de un bloque cabecera:

#		
1	Serie larga de Pulsos piloto	Ver tabla anterior para el núm. de pulsos
2	10 bytes con un ID que indica el tipo de bloque que vendrá a continuación.	BINARY (0xD0), ASCII (0xEA) , BASIC (0xD3)
3	6 bytes con el nombre del bloque	El nombre que aparecerá en el “Found:”

1.3.2.- Bloques de datos

Todos los bloques de datos inician con una serie corta de pulsos piloto seguida de los datos propiamente dichos.

Los de tipo ***BINARY*** tienen 6 bytes al principio indicando las direcciones de inicio (2), final (2) y ejecución (4) del bloque. Seguidamente se encuentran los datos propiamente dichos.

Los bloques ***ASCII*** contienen un programa en *BASIC* en formato *ASCII* que se dividen en bloques de 256 bytes. El último bloque se rellena con bytes *0x1A* hasta llegar a los 256.

Y los bloques ***BASIC*** contienen un programa en *BASIC* tokenizado seguido de 7 bytes *0x00* para indicar el final de fichero.

1.4.- El nuevo bloque #4B

ID 4B - Kansas City Standard			
longitud: [00,01,02,03] + 4			
Posición	**Valor**	**Tipo**	**Descripción**
0x00	N+12	DWORD	Longitud del bloque sin estos cuatro bytes (regla de ampliación)
0x04	-	WORD	Pausa tras este bloque en milisegundos
0x06	-	WORD	Duración de un pulso del tono GUIA en T-states
0x08	-	WORD	Número de pulsos en el tono GUIA
0x0A	-	WORD	Duración de un pulso del bit CERO en T-states
0x0C	-	WORD	Duración de un pulso del bit UNO en T-states
0x0E	-	BYTE (mapeado)	Bits 7-4: Número de pulsos en un bit CERO (0=16 pulsos) Bits 3-0: Número de pulsos en un bit UNO (0=16 pulses)
0x0F	-	BYTE (mapeado)	Bits 7-6: Número de bits de arranque Bit 5: Valor de los bits de arranque Bits 4-3: Número de bits de parada Bit 2: Valor de los bits de parada Bit 1: Reservado Bit 0: Orden de los bits en el byte (0 para LSb, 1 para MSb)
0x10	-	BYTE[N]	Flujo de datos

En los archivos *TSX* se añade un nuevo tipo de bloque a la especificación *TZX 1.20*. Debido a esto, los ficheros *TSX* se definen como *TZX* revisión 1.21.

Este bloque con *ID 0x4B* y creado por Blackhole, permite codificar bloques de datos que sigan el estándar *KCS*, por lo que podrán incluirse datos de todos los sistemas que lo adoptaron.

Para el caso concreto de los *MSX* la configuración del bloque *#4B* quedaría así para una cabecera a 1200 baudios:

Valores de un bloque #4B para MSX	
0x06 Duración pulso tono guía	729 T-States
0x08 Número pulsos tono guía	30720 / 7680 T-States
0x0A Duración pulso bit CERO	1458 T-States
0x0C Duración pulso bit UNO	729 T-States
0x0E Configuración de un BIT	0x24 (0b00100100) *0010* = 2 pulsos para un CERO *0100* = 4 pulsos para un UNO
0x0F Configuración de un BYTE	0x54 (0b01010100) *01* = 1 bit de arranque *0* = bits de arranque son 0 *10* = 2 bits de parada *1* = bits de parada son 1 *0* = *Reservado* *0* = Orden de bits LSb

2.- El programa makeTSX

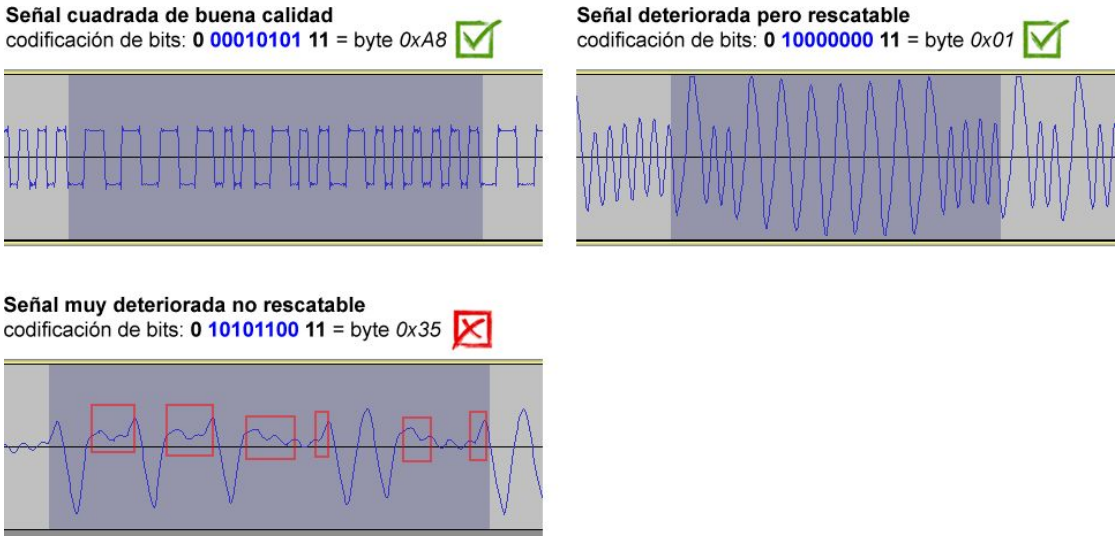
Una vez definido el formato *TSX* con el nuevo bloque *#4B*, debíamos de empezar a crear ficheros *TSX*, pero a pesar de contar con todas las herramientas propias de los *TZX*, necesitábamos una herramienta que fuera capaz de extraer bloques *MSX #4B* de un WAV y general los *TSX* con ellos. Y así nació ***makeTSX***[3].

Este programa es muy similar a *WAV2CAS* o *MAKETZX*, ya que permite extraer datos de un archivo *WAV*, siempre que la calidad de la digitalización o el deterioro de la cinta lo permitan.

Lo ideal es procesar un audio con pulsos cuadrados y definidos, pero pueden presentarse diversos problemas por deterioro de la cinta tales como:

- Ondas redondeadas (sinusoidales).
- Desplazamiento arriba o abajo del centro de los pulsos.
- Reducciones de la potencia de la señal (amplitud) confundiéndose con el ruido base.
- etc...

A veces pueden darse incluso varios de estos a la vez, complicando el rescate de la cinta.



Como se puede ver en las imágenes, los datos serán rescatables siempre que la transición entre pulsos atraviese el cero (línea horizontal central). En caso contrario (como aparece en los recuadros rojos) habría que editar las ondas a mano para arreglarlas, cosa que lleva mucho tiempo y puede llegar a ser desesperante.

Para poder ver este tipo de gráficas recomendamos usar algún programa de edición de audio como ***Audacity*** o similar.

3.- Pasos para realizar una buena conversión a TSX

A continuación pasaremos a enumerar los pasos y métodos para conseguir un *TSX* correcto y validado en la medida de lo posible.

3.1.- Digitalizar la cinta a formato WAV

El objetivo de este punto es, a partir de una cinta, conseguir un archivo *WAV* lo más correcto posible. Esto es: obtener **pulsos cuadrados definidos** y **minimizar el ruido** en los silencios.

Para conseguir pulsos cuadrados habrá que desactivar cualquier de tipo de filtro/preprocesado que posea el reproductor de cassette, el driver de sonido de tu PC, o el programa de captura de audio.

Para lograr minimizar el ruido de fondo hay que conseguir encontrar un equilibrio con el volumen de reproducción del cassette. Se debe encontrar el punto en que la señal sea lo bastante alta como para que los pulsos estén bien definidos y sus transiciones crucen la línea central del cero, pero lo bastante baja como para que la señal no se sature o aumente

demasiado el ruido de la cinta o del deterioro propio del tiempo. Esto suele verse sobre todo en los silencios entre bloques de datos.

Si por deterioro de la cinta o la acción de algún filtro se obtiene una señal sinusoidal aún pueden ser rescatables los datos si la onda cruza correctamente la línea de cero en cada transición.

3.2.- Usando el programa makeTSX

El uso básico de programa para convertir un *WAV* a *TSX* es el siguiente:

```
makeTSX -wav Nombre Fichero.wav -tsx Nombre Fichero.tsx
```

Existen otros parámetros que nos serán útiles para controlar los errores como **-v** que nos informará bit a bit y byte a byte de todo lo que se va encontrando en el archivo *WAV*.

Para más información sobre todos ellos podéis ejecutar ***makeTSX*** sin parámetros y os saldrá una lista completa.

3.3.- Control de errores

En los bloques *MSX* se usa un **sistema predictivo** de control de errores. Esto significa que cuando nos encontramos con una serie de pulsos que no se pueden identificar como ceros o unos, se utilizan los *bits de control* (start/stop bits) de ese byte y los siguientes para intentar determinar el valor correcto de ese bit.

Por desgracia no siempre es posible determinar el valor correcto de este bit defectuoso, por lo que ante este problema ***makeTSX*** nos preguntará por la acción a tomar.

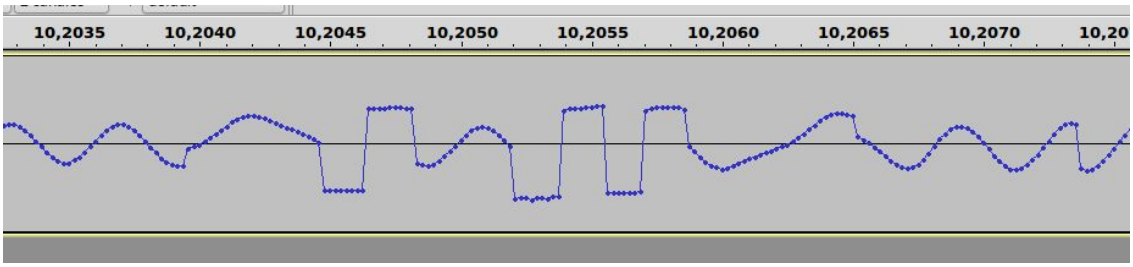
En los bloques *Spectrum* es algo más complicado el control de errores ya que no existen los *bits de control*, sin embargo suelen tener una modulación de frecuencia mucho más precisa por lo que se emplea el método de comprobar los tiempos de un ciclo completo (2 pulsos) para determinar si estamos ante un cero o un uno. También se puede usar como comprobación el último byte del bloque que es la suma checksum del resto de bytes para saber si la conversión ha sido correcta o no.

El programa ***makeTSX*** está preparado para leer bloques Spectrum sólo si son estándar. Para los casos en que sean bloques no estándar o con algún tipo de error no reconocido por el programa, lo recomendable es usar ***MAKETZX.EXE*** para extraer esos bloques en concreto y añadirlos al TSX final usando la utilidad ***ZXBlockEditor***. El programa *MAKETZX* está especializado en extracción de bloques *Spectrum*, por lo que lo hará mucho más fiablemente en caso de problemas de señal.

3.4.- Restauración manual de la señal

Es posible editar el *WAV* acudiendo al milisegundo donde se produjeron los errores (indicado en la conversión usando **-v**) y ver si se puede restaurar ese pulso para solucionarlo. Para ello usaremos el programa ***Audacity*** anteriormente citado y que es freeware.

A continuación vemos una señal en forma de onda deteriorada y posteriormente corregida a forma cuadrada para que sea más fácilmente reconocible por ***makeTSX***.



3.5.- Verificación de los datos

Una vez obtenemos un archivos *TSX* sin que se hayan detectado errores tenemos que pasar a su validación, ya que el haber obtenido el fichero no significa al 100% que sea fiable.

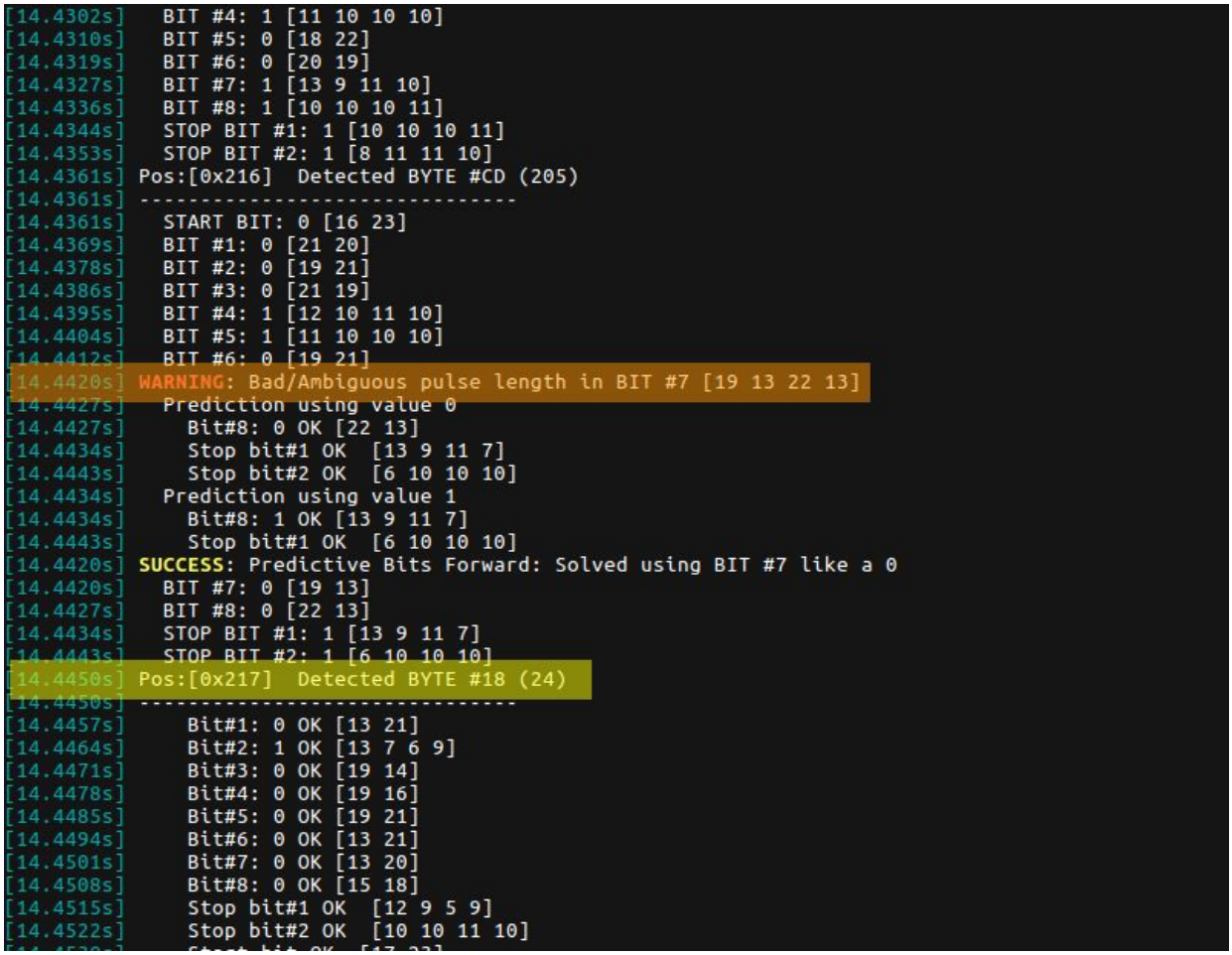
Lo ideal sería contar con varios *WAVs* de distintas cintas por cada juego a convertir, pero eso puede ser bastante complicado de conseguir, por lo que obtaremos por métodos indirectos de verificación.

3.5.1.- Verificación de cintas con bloques MSX

Para juegos en que todos los bloques de datos sean MSX se puede usar el método de compararlo con los datos de su archivo CAS correspondiente. Para ello convertiremos el archivo *CAS* a *TSX* usando el script ***cas2tsx.php***[4], tras lo cual pasaremos a abrir ambos (nuestro *TSX* y el *CAS* convertido) con ***ZXBlockEditor*** y comparar los *CRC* de los bloques uno a uno para detectar discrepancias.

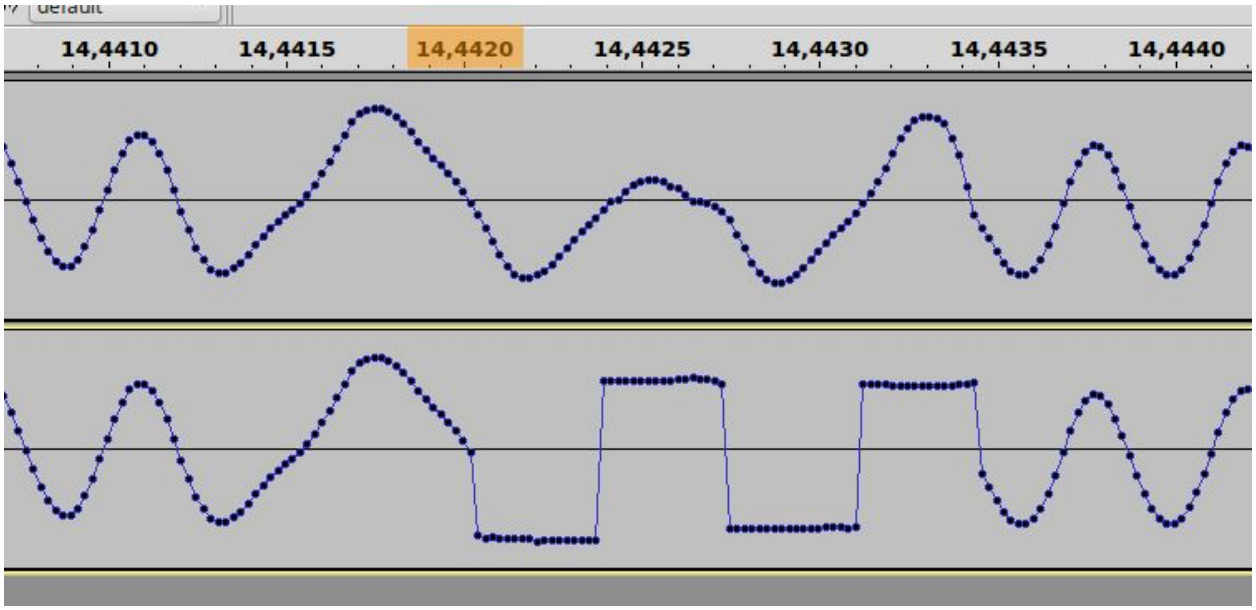
Si todos coinciden será un gran indicativo de que estemos ante una conversión correcta al 99%.

Si en algún bloque no coinciden los CRC habrá que mirar en qué byte/bytes se encuentran las diferencias, para ello exportamos esos bloques con el propio ***ZXBlockEditor*** y los comparamos con algún comparador de archivos binarios (***vbindiff*** es una buena opción de línea de comando por ser multiplataforma).



Supongamos que el byte en cuestión se encuentra en la posición *0x0217* del segundo bloque. Una vez localizado nos vamos al ***makeTSX*** y con la opción **-v** miramos el milisegundo donde se encuentra ese byte en el *WAV*.

Como podemos observar en la imagen se ha detectado un pulso extraño en su bit 7, por lo que tendremos que abrir el *WAV* con ***Audacity*** y hacer zoom en el segundo **14,4420**.



Aquí podemos ver la señal original arriba y la corregida manualmente abajo en la que hemos corregido los bits 7 y 8.

Lo importante es recordar que ante cualquier discrepancia entre el *CAS* y el *TSX* lo mejor es ir a la fuente original, el *WAV*, y así poder determinar el valor correcto de esos bits.

3.5.2.- Verificación de cintas con bloques Spectrum

Todo lo indicado en el anterior punto es válido para los bloques *MSX* de estas cintas, salvo en que en al menos uno de estos bloques habrá un grupo de bytes que serán distintos entre el *TSX* y el *CAS*. Esto es debido a que la **rutina de carga** de bloques *MSX* (incorporada en los *CAS*) y de bloques Spectrum (la original) son distintas. En este caso, si la carga se comprueba correcta en el *TSX*, dejaremos la rutina de carga *Spectrum* como correcta.

En los bloques *Spectrum* (con ID *#10* y *#11*) se puede comprobar fácilmente si el checksum es correcto o no, aún así, se pueden comparar también sus *CRC* con sus bloques correspondientes del *CAS* aun estando en formatos distintos.

Para compararlos abriremos nuestro *TSX* y el *TSX* generado desde el *CAS* como vimos anteriormente, pero en el fichero procedente del *CAS* abriremos los bloques que se convirtieron desde Spectrum y le eliminaremos el primer byte (byte *Flag*). También los ceros del final y el último byte que es el *Checksum*.

Esto lo haremos así debido a que el ***ZXBlockEditor*** no utiliza los bytes de *Flag* y de *Checksum* para calcular el *CRC* en los bloques *#10* y *#11* por lo que hay que eliminar esos bytes de los bloques correspondientes del *CAS*.

En cuanto a la eliminación de ceros del final es debido a que por limitación del formato *CAS* los bloques deben de tener un número de bytes múltiplo de 8 por lo que los bloques se rellenan con ceros.

El objetivo es por tanto obtener un bloque del mismo tamaño que el original de *Spectrum* y que empiece y termine por los mismos bytes, solo así podremos comparar sus *CRC*.

Es importante tener siempre en mente que si dudamos de la verificación de los bloques *Spectrum* siempre podemos usar ***MAKETZX*** como se indica en el capítulo **3.3** y añadir esos bloques problemáticos editando tu *TSX*.

---

(1) https://yotambientuveunmsx.wordpress.com/2016/12/11/el-casete-en-msx-info-tecnica/

(2) https://en.wikipedia.org/wiki/Kansas_City_standard

(3) *TSX MSX Repository:* http://tsx.eslamejor.com

(4) *https://github.com/nataliapc/MSX_devs/tree/master/TSXphpclass*