Desarrollo de una interfaz USB 2.0 para los analizadores lógicos LA-4540

Ignacio Martín Vázquez, Francisco Javier Martín Gutiérrez

5 de septiembre de 2008

Índice general

Índice de figuras

Índice de cuadros

1 Introducción

1.1. Justificación

A finales de los años 90 el Departamento de Electricidad y Electrónica adquirió un conjunto de analizadores lógicos LA- 4540. Dichos analizadores se comunican con el PC mediante una tarjeta de comunicaciones que utiliza el conector ISA. Dado que los últimos PCs incorporados a los Laboratorios de Electricidad y Electrónica ya no disponen de este conector, surge la necesidad de implementar o bien una interfaz Paralelo o bien una interfaz USB 2.0, que permita usar los LA-4540 con los nuevos PCs.

1.2. Objetivos

El objetivo principal del proyecto será el *desarrollo del hardware y el software* necesario para implementar tanto la interfaz Paralelo como la interfaz USB 2.0.

Dentro de este objetivo se incluye el proceso de *ingeniería inversa* para poder conocer el funcionamiento del LA-4540, el *desarrollo y test de dos PCB's* como interfaces entre el PC y el LA-4540 y por último la realización de un *software* para poder manejar el analizador.

1.3. Estrategias

1.3.1. Estrategia Inicial

Para empezar con el proyecto lo primero es conocer con detalle el analizador lógico, para ello se utiliza la documentación proporcionada por el fabricante norteamericano *Link Instruments* al comprar los analizadores lógicos *LA-4540*. Esta documentación consta de una guia denominada *Guide to LA4000 Series Lógica Analyzer Cards, Software Revision I (1)* y un diskette con lo que en un principio se piensa que es el código en ensamblador del modelo que se va a manejar.

Al consultar la guía se obtienen entre otros aspectos las características técnicas, los modos de funcionamiento,la utilización del software y por último cómo conectar el analizador al

PC mediante una tarjeta de comunicaciones al bus ISA del PC. Al no encontrar ninguna documentación sobre dicha tarjeta es paso obligado conocer el esquema eléctrico de la tarjeta, como también el protocolo del bus al que esta conectada, en este caso el protocolo ISA. Con esto se conocerán las señales que interactúan desde el bus ISA con la tarjeta de comunicaciones, y cuáles de esas señales son las que realmente intervienen en el protocolo de comunicación entre el PC y el LA-4540.

Por otro lado, se estudia el código en ensamblador proporcionado por el fabricante en el diskette. En dicho archivo se reconocen partes de código que corresponden a acciones tales como configuración del analizador, transmisión de parámetros, petición de datos de los pods... Tras esta primera impresión se piensa en una linea de trabajo basada en este código. Es por ello que se pasa a compilarlo y ejecutarlo. Como en primera instancia esto no funciona se pasa a depurar el código. Después de un largo periodo de pruebas, test y depuración del código (explícitamente suministrado por el fabricante), se llega a un punto de inflexión donde se cree que dicho código no permitirá el manejo del analizador.

En consecuencia, se pasó a buscar un archivo válido en la página web del fabricante *www.linkinstruments.com*, llegándose a la conclusión de que el código suministrado no corresponde con el modelo LA-4540 sino con otra serie anterior denominada LA-32xxx, y por tanto dicho código no hace funcionar nuestro analizador.

Llegado este momento se pasa a busca otra linea de trabajo. Sin embargo, hay que mencionar que todo este trabajo no será en balde, ya que parte del código en ensamblador será útil para describir acciones de funcionamiento del analizador.

1.3.2. Estrategia Segunda

Siguiendo en la misma linea de trabajo, es decir, basarse en aplicar ingeniería inversa sobre un código que haga funcionar al analizador y conseguir nuestro propio código, se pasa a analizar los archivos ejecutables basados en dos sistemas operativos diferentes.

En primer lugar se desensambla el archivo ejecutable para MS-DOS utilizando el software IDA Pro. El resultado es un código en ensamblador en el que ya no existen etiquetas o comentarios con las que poder saber que esta haciendo el programa.

En segundo lugar al desensamblar el ejecutable para Windows se obtuvo un código nuevamente en ensamblador. Al ser un programa desarrollado para Windows el programa de desensamblado era capaz de reconocer funciones propias de este lenguaje, sin embargo, las instrucciones que se buscaban tales como transmisión de parámetros, inicio de funcionamiento... no se encontraban.

Ambos códigos desensamblados podían ser depurados utilizando el programa IDA Pro, pero

no se obtuvieron resultados satisfactorios.

Nuevamente había que cambiar de línea de investigación.

1.3.3. Estrategia Final

Dado que las anteriores opciones de estudio no llegaron a buen puerto se opto por el camino más seguro para obtener el funcionamiento del analizador aunque más largo y costoso. Esta línea de trabajo consiste en "espiar" el comportamiento del analizador para después reproducirlo a nuestro antojo.

Para poder conocer el funcionamiento de los analizadores lógicos LA-4540 se necesitarán:

- Un primer PC con un LA-4540 muestreando señales provenientes del entrenador Function Generator IDL-800 Digital Lab.
- Un segundo PC con otro LA-4540 que muestreé las señales de interés del bus ISA del primero.

Y la estrategia a seguir consistirá en:

- El primer PC mandará comandos a su analizador y desde el segundo PC mediante el otro analizador se obtiene la secuencia de datos transmitidos.
- Entonces, cada vez que se mande un comando desde el primer PC a su LA-4540, esta información será recogida desde el otro PC en los archivos de salida `log.la` generados por el programa de usuario del segundo analizador.
- Por último,esos archivos `log.la` se filtran y se obtiene la información necesaria para hacer funcionar a los LA-4540.

Una vez que se sabe lo qué hay que hacer, de dónde hay que sacar la información y qué dispositivos hay que usar sólo queda iniciar el proceso de investigación.

2 Estudio de la tarjeta de expansión del LA-4540

2.1. Introducción al estándar ISA

El analizador lógico se conecta al PC a través de una tarjeta de comunicaciones que se inserta en un slot donde este presente el conector del bus ISA. Por tanto, a continuación se da una pequeña noción sobre ello.

Historia

Cuando en 1980 IBM fabricó su primer PC, este contaba con un bus de expansión conocido como XT que funcionaba a la misma velocidad que los procesadores Intel 8086 y 8088 (4.77 MHz). El ancho de banda de este bus (8 bits) con el procesador 8088 formaba un tándem perfecto, pero la ampliación del bus de datos en el 8086 a 16 bits dejo en entredicho este tipo de bus (aparecieron los famosos cuellos de botella).

Dada la evolución de los microprocesadores el bus del PC no era ni mucho menos la solución para una comunicación fluida con el exterior del micro. En definitiva no podía hablarse de una autopista de datos en un PC cuando esta sólo tenía un ancho de 8 bits. Por lo tanto con la introducción del AT apareció un nuevo bus en el mundo del PC, que en relación con el bus de datos tenía finalmente 16 bits (ISA), pero que era compatible con su antecesor. La única diferencia fue que el bus XT era síncrono y el nuevo AT era asíncrono. Las viejas tarjetas de 8 bits de la época del PC pueden por tanto manejarse con las nuevas tarjetas de 16 bits en un mismo dispositivo.

Así cuando en 1984 IBM presenta el PC AT (con el procesador Intel 80286) se rompió la aparentemente inquebrantable relación entre bus y microprocesador. Aunque en la practica el reloj del procesador de un AT funciona a la misma velocidad que su reloj de bus, IBM había abierto la puerta a la posibilidad de que este último fuese más rápido que el reloj del bus. Así pues el bus que incorporó el AT fue de un ancho de banda de 16 bits funcionando a 8.33 MHz. Este enfoque de diseño no oficial se denominó oficialmente ISA (Industry Standard Arquitecture) en 1988.

Puesto que el bus ISA ofrecía algunas limitaciones en IBM se desarrolló otro tipo de bus que funcionaba a 10 MHz y que soportaba un ancho de banda de 32 bits. Este bus se monto en la gama PS/2. El gran problema de este bus es que no era compatible con los anteriores y necesitaba de tarjetas de expansión especialmente diseñadas para su estructura. El ISA es el bus usado en las computadoras PC de IBM y también en las computadoras compatibles (clones). Su función principal es realizar comunicación entre el procesador, ubicado en la placa madre (motherboard) y las tarjetas de expansión de periféricos (add-in board).

Inicialmente había problemas de compatibilidad, puesto que IBM no publicó las especificaciones de timing del bus. Esto fue resuelto en 1987, cuando el IEEE produjo una especificación completa del bus (incluyendo timing).

El PC original también tenía un bus de datos de 8-bits (llamado bus PC/XT), debido a que el procesador Intel 8088 CPU (usados en los primeras PCs de 4.77-MHz de 1982) tenía un bus de datos de 8-bit. Este bus, de 62 pines, tenia las siguientes características:

- Ocho líneas de datos (que permite la transferencia simultánea de datos de 8 bits).
- Veinte líneas de direcciones (Permite direccionar 1 Mbyte, aunque para el adaptador vídeo se asignó 128 KBytes, comenzando en la dirección 640 KB, esto creó la famosa limitación de memoria del DOS).
- Seis líneas de interrupciones, identificadas como IRQ2 a IRQ7.
- Los canales de DMA 1, 2, y 3 (con dos señales para cada canal, demanda y reconocimiento).
- El motherboard original de los PC usaba de canal 0 de DMA para refresco de memoria y debido a que el controlador de DMA y la memoria estaban en el motherboard, estas señales no se pusieron en el bus.
- Los otros pines se usaban para voltajes y señales de control.

El procesador Intel 80286, que fue usado en la computadora IBM PC/AT (Advanced Technology –Tecnología Avanzada, 1984), tenía un bus de datos de 16 bits, tal que IBM, que en ese momento era el único fabricante "normalizador" de los PCs, agregó un conector de 36 pines que proporcionaba las siguientes características:

- Ocho líneas más de datos (permitiendo transferencias de datos de 16 bits).
- Cuatro líneas más de direcciones (permitiendo direccionar 16 Mbytes de memoria)

- Las interrupciones 10, 11, 12, 14, y 15 (interrupción 13 es reservada para el coprocesador matemático que estaría en el motherboard de manera que la interrupción 13 no está disponible en el bus).
- Los canales de DMA 0, 5, 6, y 7 (el canal 4 de DMA se usa para enlazar el nuevo controlador de DMA con el original). Al contrario de la PC original, la PC/AT usa un circuito dedicado para el refresco de memoria, tal que el canal 0 está ahora disponible para su uso en el bus.

Este es el bus PC/AT de 16 bits, o más comúnmente (y simplemente), el denominado "bus ISA", el cual está presente en la mayoría de los PCs (los PCs más antiguos sólo tenían bus ISA y los PCs más nuevas típicamente tienen los buses ISA y PCI). En años recientes, se ha creado una norma más formal llamada bus ISA (Arquitectura Industrial Normalizada - Industry Standard Architecture),

El bus ISA tiene una velocidad de transferencia máxima teórica de 16 Mbytes/s (aunque en realidad es solo de 8 Mbytes/s, debido a que normalmente se requiere de un ciclo para el direccionamiento y otro ciclo para los datos).

La velocidad típica máximo es 1 a 2.5 Mbytes/s (8 a 20 Mbits/s). Esta velocidad es variable debido a la contención del bus con otros dispositivos (principalmente la memoria) y el retardo de los buffers debido a la naturaleza asíncrona del bus (la velocidad del procesador es diferente de la velocidad del bus).

Con el paso del tiempo los procesadores de las computadoras AT (y sucesivas) se volvieron más rápidos y eventualmente sus buses se hicieron más anchos, pero el deseo de mantener la compatibilidad con los dispositivos existentes llevó a los fabricantes para resistirse a un cambio de norma y por lo tanto el bus ISA ha permanecido semejante desde ese tiempo.

Descripción de bus

En la figura 2.1 se puede ver el pinout del bus ISA.

El bus está compuesto por un conector hembra para placa de circuito impreso de 62 pines, correspondiente al original XT y un conector hembra para placa de circuito impreso de 36 pines, que corresponde a la ampliación AT. El conector está dividido en dos caras.

- En la primera los pines se denominan desde A1 hasta A31 y es la cara de componentes. Contiene el bus de direcciones y de datos.
- Los pines de la segunda cara se denominan desde B1 hasta B31 y es la cara de soldadura. Esta cara contiene los pines de alimentación así como las señales relacionadas con las interrupciones y las transferencias de datos vía DMA.

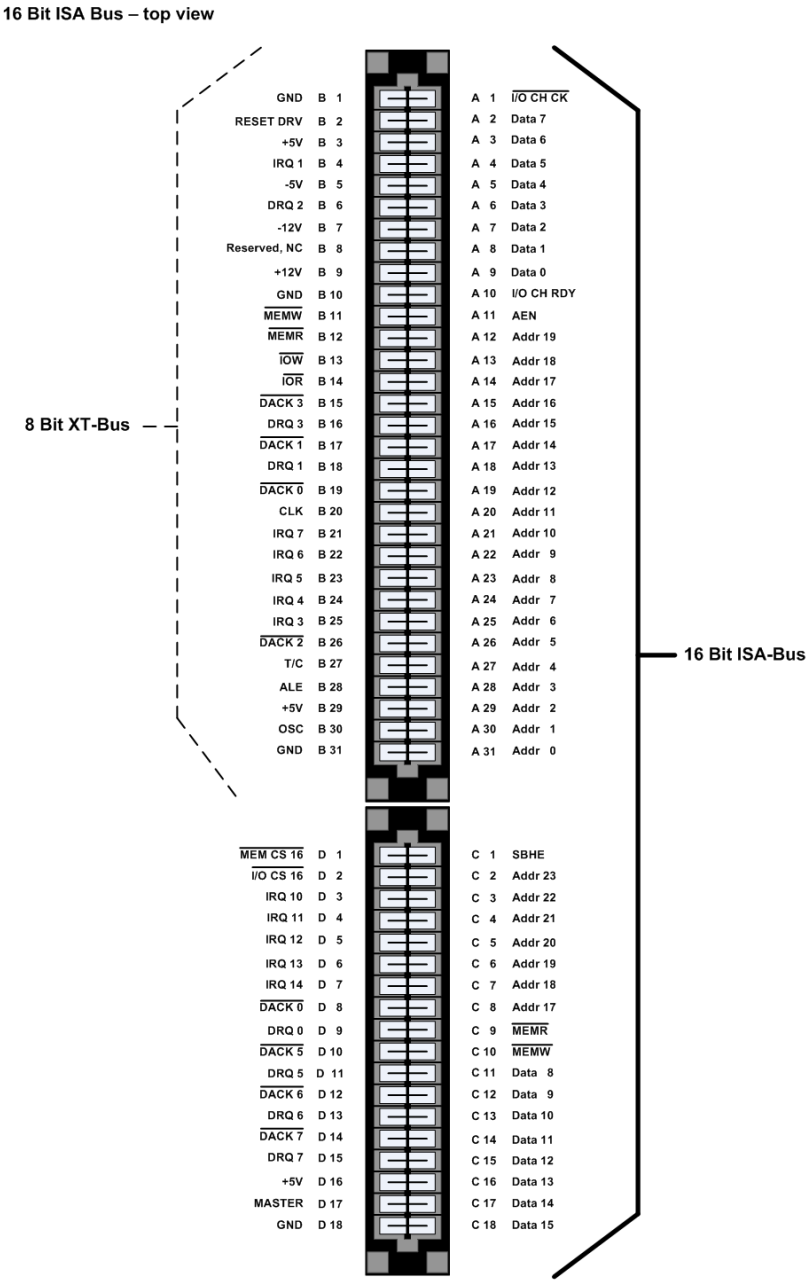


Figura 2.1: Pinout ISA

En la tabla 2.1 esta las descripción del las señales de la figura 2.1[1]:

[1] más documentación en (10, 11)

Cuadro 2.1: Descripción de las señales del bus

Señal	Pines	Descripción
Addr0 - Addr19	A31-A12	Bits de dirección 0-19, permiten direccionar 1Mb de memoria y 64K de puertos de e/s.
Addr17 - Addr23	C8-C2	Bits de dirección 17-23, permiten direccionar desde 256Kb de memoria a 16Mb. Son válidas cuando ALE está activa.
AEN	A11	Address Enable; Cuando está activa el controlador DMA posee el control de las lineas de dirección y del BUS de datos, conforme se indique en MEMR/MEMW. Cuando está inactiva la CPU tiene el control de estas lineas.
ALE	B28	Address Latch Enable (salida); se emplea para que la CPU esté aislada de las lineas de dirección (triestado). Es forzado activado durante los ciclos DMA.
CARD SLCTD	B8	Card Selected; indica que una tarjeta ha sido activada en el slot XT de 8 bits.
CLK	B20	Señal de reloj del sistema (actual velocidad del BUS).
Data0- Data7	A9-A2	Bits de datos 0-7 para e/s a memoria o puertos de e/s.
Data8- Data15	C11-C18	Bits de datos 8-15 para e/s a memoria o puertos de e/s.
$\overline{DACKx}$		Reconocimiento DMA para los canales 0 al 3; empleada por el controlador para reconocer una petición DMA (validación de acceso DMA). DACK0 es empleada para el refresco de memoria (MREF).
DRQx		Petición DMA 0-3 y 5-7; empleada por periféricos que desean los servicios del controlador DMA; Se mantiene activa hasta que la correspondiente señal DACKx se hace activa. Las señales de "requerimiento de DMA" son Solicitudes asincrónicas emitidas por los dispositivos del bus para obtener servicios de DMA.
$\overline{I/OCHCK}$	A1	I/O Channel Check; Genera una interrupción no enmascarable.

Cuadro 2.1: Descripción de las señales del bus

Señal	Pines	Descripción
$I/O\,CH\,RDY$	A10	I/O Channel Ready; es puesta inactiva por memoria o dispositivos de e/s para retardar el acceso a memoria o los ciclos de e/s. Normalmente es empleada por dispositivos lentos para añadir estados de espera. No debe ser inactiva durante más de 17 ciclos.
$\overline{I/O\,CS16}$	D1	I/O Chip Select 16 Bit; indica ciclo de e/s de 16 bits
$\overline{IOR}$	B14	I/O Read; indica a un dispositivo de e/s que coloque su dato en el BUS del sistema.
$\overline{IOW}$	B13	I/O Write; indica a un dispositivo de e/s a leer un dato del BUS del sistema.
IRQx		Petición de interrupción; indica a la CPU que un dispositivo de e/s necesita servicio. IRQ2 tiene la prioridad más alta. IRQ 10-15 están sólo disponibles en máquinas AT y son de prioridad más alta que IRQ 3-7.
MASTER	D17	Empleado por DRQ para ganar el control del sistema.
$\overline{MEM\,CS16}$	D1	Memory Chip Select 16 bit; indica ciclo de memoria de 16 bits.
$\overline{MEMR}$	B12 y C9	Memory Read; esta señal es producida por la CPU o el controlador DMA e indica a la memoria que debe introducir el dato direccionado en el BUS del sistema. Presente tanto en el BUS PC como en la extensión AT.
$\overline{MEMW}$	B11 y C10	Memory Write; esta señal es producida por la CPU o el controlador DMA e indica a la memoria que debe leer y almacenar el dato presente en el BUS. Presente tanto en el BUS PC como en la extensión AT.
OSC	B30	Oscilador; Señal de reloj de 14.31818 MHZ (periodo de 70ns); 50 % del ciclo de servicio.
RESET DRV	B2	Reset Drive; empleada para resetear la lógica del sistema.
SBHE	C1	System BUS High Enable; activa los bits de datos 8-15 de la extensión AT del BUS.

Cuadro 2.1: Descripción de las señales del bus

Señal	Pines	Descripción
TC	B27	Terminal Count; produce un impulso cuando la cuenta final de un canal DMA es alcanzado.

Funcionamiento

A continuación se describe el funcionamiento del bus ISA con un ciclo de lectura desde un puerto de entrada/salida. Lo primero que hace el microprocesador es poner la señal ALE hasta un nivel alto, entonces envía la dirección del puerto a través de las señales A0-A19. Después, la señal ALE vuelve a nivel bajo. En adelante la dirección del puerto a ser leído quedará retenida en un latch. Entonces el bus pone $\overline{IOR}$ a nivel bajo. El dispositivo direccionado enviara un byte de datos a través de las líneas D0-D7 del bus de datos. El microprocesador leerá el bus de datos y pondrá la señal $\overline{IOR}$ a nivel alto de nuevo.

Un ciclo de escritura desde un puerto funciona de la siguiente manera: El microprocesador pone la señal ALE a "1", entonces envía la dirección del puerto a través de A0-A19. ALE es puesta a nivel bajo. El microprocesador envía el byte de datos que será escrito. Luego pone un "0" en $\overline{IOW}$. Después de que el dispositivo ha tenido tiempo de leer el byte, el uP pone la señal$\overline{IOW}$ a nivel alto de nuevo.

La única diferencia entre un ciclo de lectura/escritura a memoria y un ciclo de lectura/escritura a un puerto consiste en que en un ciclo de memoria se utilizarán las señales $\overline{MEMR}$ y $\overline{MEMW}$ de la misma manera que se hace con $\overline{IOR}$ y $\overline{IOW}$.

2.2. Esquema eléctrico

Para poder comprender que hace la tarjeta de comunicaciones lo primero es obtener su esquema eléctrico.

Como entradas están las señales provenientes de los pines ISA, es decir, las señales de la cara de componentes - cara A - A1, A2... A31 y también las de la cara de soldadura - cara B - B1, B2... B31. Y como salida tenemos los pines del conector SCSI de la tarjeta de comunicaciones.

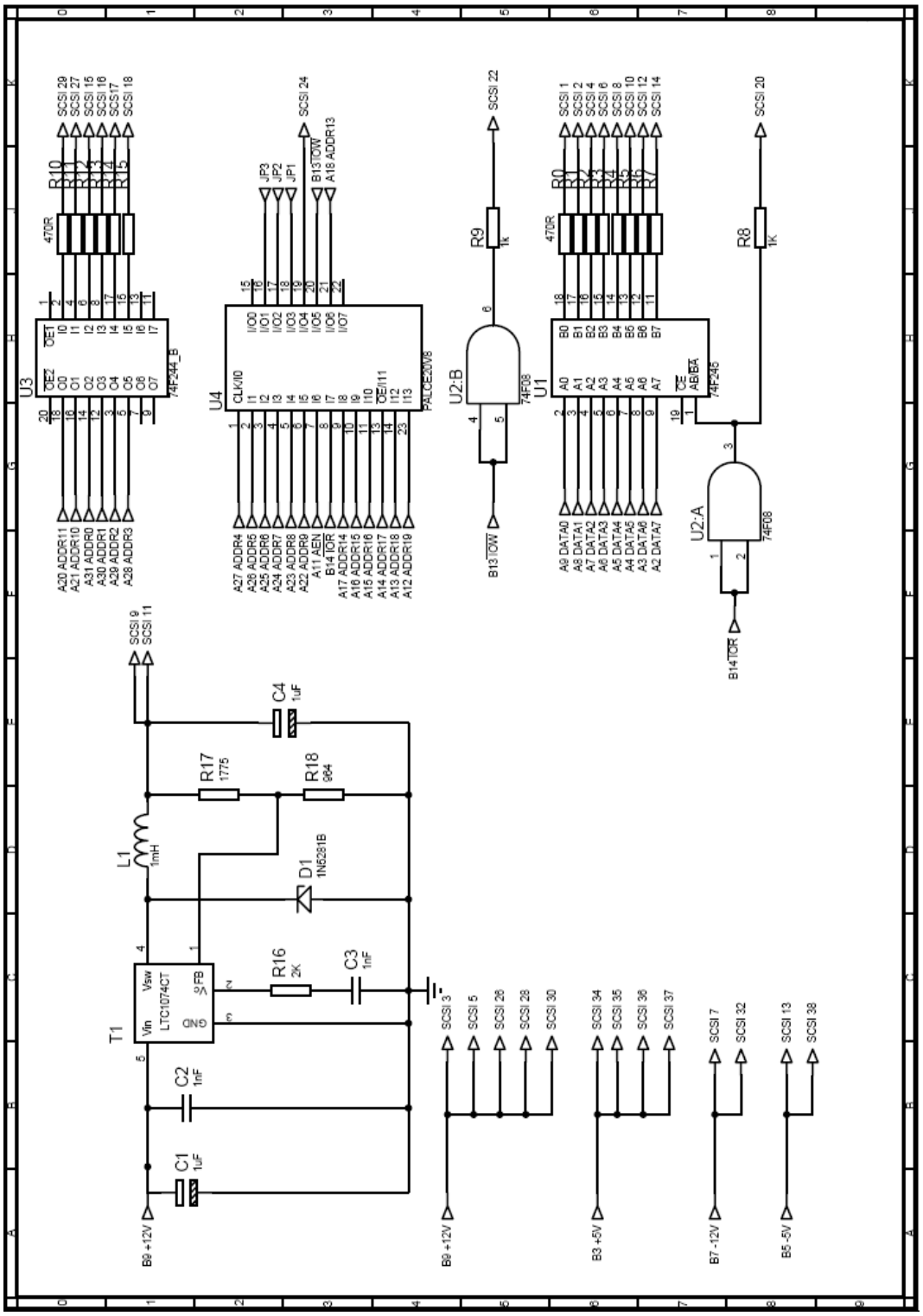


Figura 2.2: Esquema Eléctrico Tarjeta de Comunicaciones de LA-4540

A partir de ahora habrá una equivalencia entre los nombres de las señales que desde el conector ISA consiguen llegar hasta el LA-4540. Por ejemplo, cuando se hable de la señal $\overline{IOR}$ se entenderá que es la misma señal que la denominada SCSI 20 y B14, ya que si uno se fija en el esquema eléctrico ambas son la misma señal.

pin ISA	pin SCSI	nombre de la señal
A 8	SCSI 2	Data1
A 31	SCSI 15	Addr0
A 9	SCSI 1	Data0
A 7	SCSI 4	Data2
A 6	SCSI 6	Data3
A 5	SCSI 8	Data4
A 4	SCSI 10	Data5
A 3	SCSI 12	Data6
A 2	SCSI 14	Data7
A 1	SCSI 17	Data8
A 30	SCSI 16	Addr1
A 29	SCSI 17	Addr2
A 28	SCSI 18	Addr3
A 21	SCSI 27	Addr10
A 20	SCSI 29	Addr11
B 13	SCSI 22	$\overline{IOW}$
B 14	SCSI 20	$\overline{IOR}$
PALCE pin 19	SCSI 24	$\overline{SEN}$

Cuadro 2.3: Equivalencia de nomenclatura conector ISA - conector SCSI

2.3. CI presentes en la PCB y descripción de su funcionamiento en la tarjeta

Los componentes que presenta el anterior esquema eléctrico son:

- PALCE20V8
- 74F245
- 74F244
- LCT1074CT
- 74F08

2.3.1. PALCE20V8 Reprogramable CMOS OAL Device

Descripción General

El CI Cyperss PALCE20V8 es un array lógico programable basado en memoria Flash CMOS. Esta implementada mediante estructuras lógicas AND-OR, suma de productos, y macro células programables. El dispositivo es provisto de 20 entradas y 8 salidas. La PALCE20V8 es borrable eléctricamente y reprogramable.

Hay un total de 18 bits de arquitectura en el modelo macrocélula de PALCE20V8; 2 de ellos son bits globales que se aplican a todas las macrocélulas, y los 16 restantes se aplican localmente, siendo 2 bits por célula. La arquitectura de bits determina si las macrocélulas funcionan como un registro o como circuito combinacional con salida inversora o no inversora.

Todas las entradas y entradas/salidas tienen incorporado un pull-up. El fabricante recomienda que las entradas no usadas y las salidas triestado no usadas deben ser conectadas a otras entradas activas, Vcc o Gnd para mejorar la inmunidad al ruido y reducir Icc.

Diagrama de bloques lógico

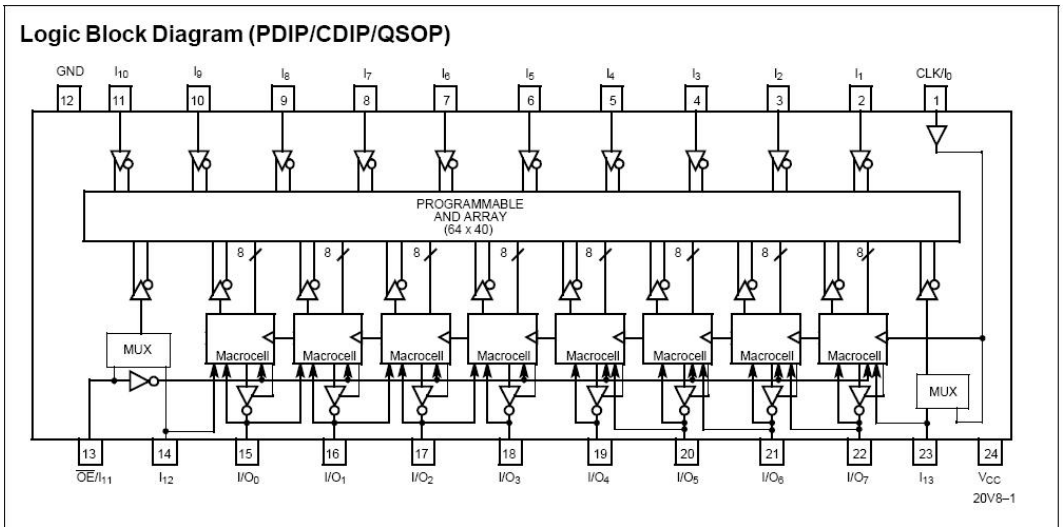


Figura 2.3: PALCE20V8

Función que ejerce dentro de la tarjeta

Para empezar hay que fijarse en qué señales son las que entran a la PAL y cuáles son sus salidas. Después hay que investigar para averiguar como han programado la PAL, para ello se

extrajo el CI de la PAL de la tarjeta de comunicaciones y se montó en una placa de pruebas en el entrenador.

Como ayuda se tuvo el datasheet de la PALCE, aunque en realidad consultarlo hizo surgir más interrogantes que soluciones; por otro lado, estaba la guía de manejo del analizador. Al consultarla se obtuvo una pequeña aproximación de la función de la PAL; dependiendo de 3 jumpers que están en la tarjeta de comunicaciones y que son entradas de la PAL, la tarjeta de comunicaciones obtiene un mapa de direcciones distinto.

Para comprobar la anterior premisa se colocó la PAL en el entrenador, para después realizar una pequeña batería de prueba con sus entradas. Tras esta prueba se llego a las siguientes conclusiones:

- No todas las señales que llegan a la PAL ejercen una función. Es el caso de AEN, $\overline{IOR}$,$\overline{IOW}$ y las direcciones desde Addr13 hasta Addr19.

- Las señales de entradas relevantes para la PAL son las direcciones desde Addr4 hasta Addr9.

- Solo hay una salida - pin 19 de la PAL- y es activa a nivel bajo.

- El conexionado de los jumpers denominados JP1, JP2 y JP3 influyen en el comportamiento de la PAL. Se dejarán a un valor por defecto, todos a '0' lógico.

Por tanto la función de la PAL es la de saber cuando el PC se está dirigiendo a la tarjeta de comunicaciones. Para el caso por defecto JP1=JP2=JP3='0' el valor de las señales de entrada se expone en la tabla 2.5. Y cuando se direcciona correctamente la única salida de la PAL se activa a nivel bajo,2.3.1.

Numero de Pin ISA	Descripción ISA	Valor lógico
A 27	Addr 4	1
A 26	Addr 5	0
A 25	Addr 6	0
A 24	Addr 7	0
A 23	Addr 8	0
A 22	Addr 9	1
A 18	Addr 13	X
A 17	Addr 14	X
A 16	Addr 15	X
A 15	Addr 16	X
A 14	Addr 17	X
A 13	Addr 18	X
A 12	Addr 19	X
A 11	AEN	X
B 13	$\overline{IOW}$	X
B14	$\overline{IOR}$	X

Cuadro 2.5: Señales ISA que entran a la PAL

Numero de Pin PAL	Descripción funcional	Valor
19	$\overline{SEN}$	Cuando se direcciona la PAL $\overline{SEN}$= 0

Cuadro 2.6: Señales que salen de la PALCE

Con la información anterior y teniendo en cuenta como se ha decidido llamar a las señales en la tabla 2.3se puede aclarar el mapa de direcciones del LA[2]:

Addr11	Addr10	Addr9	Addr8	Addr7	Addr6	Addr5	Addr4	Addr3	Addr2	Addr1	Addr0
-	-	1	0	0	0	0	1	-	-	-	-

Cuadro 2.7: Mapa de direcciones

Por tanto la dirección base seleccionada con esta conexión de jumpers es la 0x210.

[2]A la vista de la tabla 2.7 hay unas direcciones fijas y otras variables para así poder dirigirse a los distintos CI que el dispositivo LA-4540 lleva como hardware.

2.3.2. 74F245 Octal Bidirectional Transceiver with 3-STATE Outputs

Descripción General

El 74F245 contiene 8 buffers no inversores con salida triestado y fue pensado para aplicaciones tipo bus. La corriente por el puerto A es de 24 mA y por el puerto B es de 64 mA. La entrada Transmit/Receive ($T/\bar{R}$) determina la dirección del flujo de datos a través del transceptor. Si en la entrada ($T/\bar{R}$) está a nivel alto, los datos se transmiten desde el puerto A hasta el B. Si por el contrario ($T/\bar{R}$) está a nivel bajo, los datos pasan del puerto B al A. Por último, si la entrada Output Enable está a nivel alto se deshabilitan ambos puertos y sus salidas pasan a alta impedancia.

Diagrama de conexión

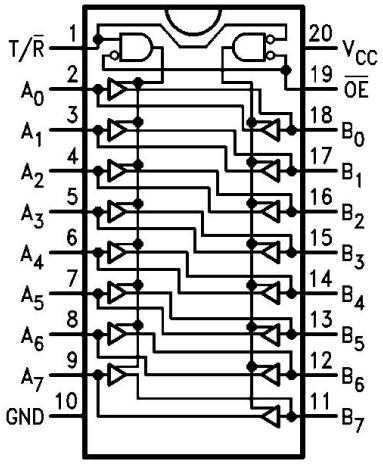


Figura 2.4: 74F245

Función que ejerce dentro de la tarjeta

Este integrado se encarga de la transmisión bidireccional del bus de datos. En el puerto A se encuentra el PC (pines A2 -A9) y en el puerto B está el analizador.

El sentido de la transmisión viene dado por la señal $\overline{IOR}$ (pin B14). Cuando $\overline{IOR}$ = '0' el sentido es desde el analizador hacia el PC. Y cuando $\overline{IOR}$ = '1', el sentido es desde PC al analizador.

Este CI estará activo cuando la única salida de la PAL, es decir, la llamada $\overline{SEN}$ este a nivel bajo; esto se consigue poniendo en el patilla 19($\overline{ChipEnable}$) del 74F245 la señal $\overline{SEN}$, como se aprecia en la figura 2.2.

2.3.3. 74F244 Octal buffers/Line Drivers with 3 STATE Outputs

Descripción General

El CI 74F244 está compuesto por 8 buffer de salida triestado con una corriente de 64 mA por cada uno de ellos. Está diseñado para ser empleado como driver de memoria y direccionamiento, mejorando la transmisión/recepción entre PC y dispositivos.

Diagrama de conexión

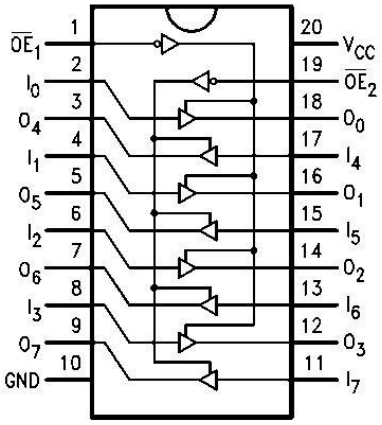


Figura 2.5: 74F244

Función que ejerce dentro de la tarjeta

Por este circuito pasan las lineas de dirección Addr11, Addr10, Addr3, Addr 2, Addr1 y Addr0 desde el PC a la tarjeta. Por tanto, este CI sirve para adaptar y mejorar las lineas de direcciones que van a llegar al analizador.

2.3.4. LT1074CT Step Down Switching Regulator

Descripción General

El LT1074 es un regulador de conmutación bipolar capaz de suministrar hasta 5A que requiere de unos pocos elementos externos para un funcionamiento normal. El conmutador, el oscilador, circuito de control y los componentes limitadores de corriente se encuentran dentro del integrado. La topología básica es la de un reductor pero varias inovaciones en el diseño permiten a este dispositivo ser usado como convertidor positivo o negativo, como un convertidor elevador negativo, y como un convertidor flyback.

El LT1074 usa un verdadero multiplicador analógico en el lazo de realimentación lo que le permite responder casi inmediatamente a las fluctuaciones de tensión de entrada y hace que la ganancia del sistema sea independiente de la tensión de entrada.

La entrada de tensión del CI va desde 8v hasta 60v, pero la característica de autoarranque permite tensiones tan pequeñas como 5V para configuración inversora y configuración elevador.

Diagrama Bloques

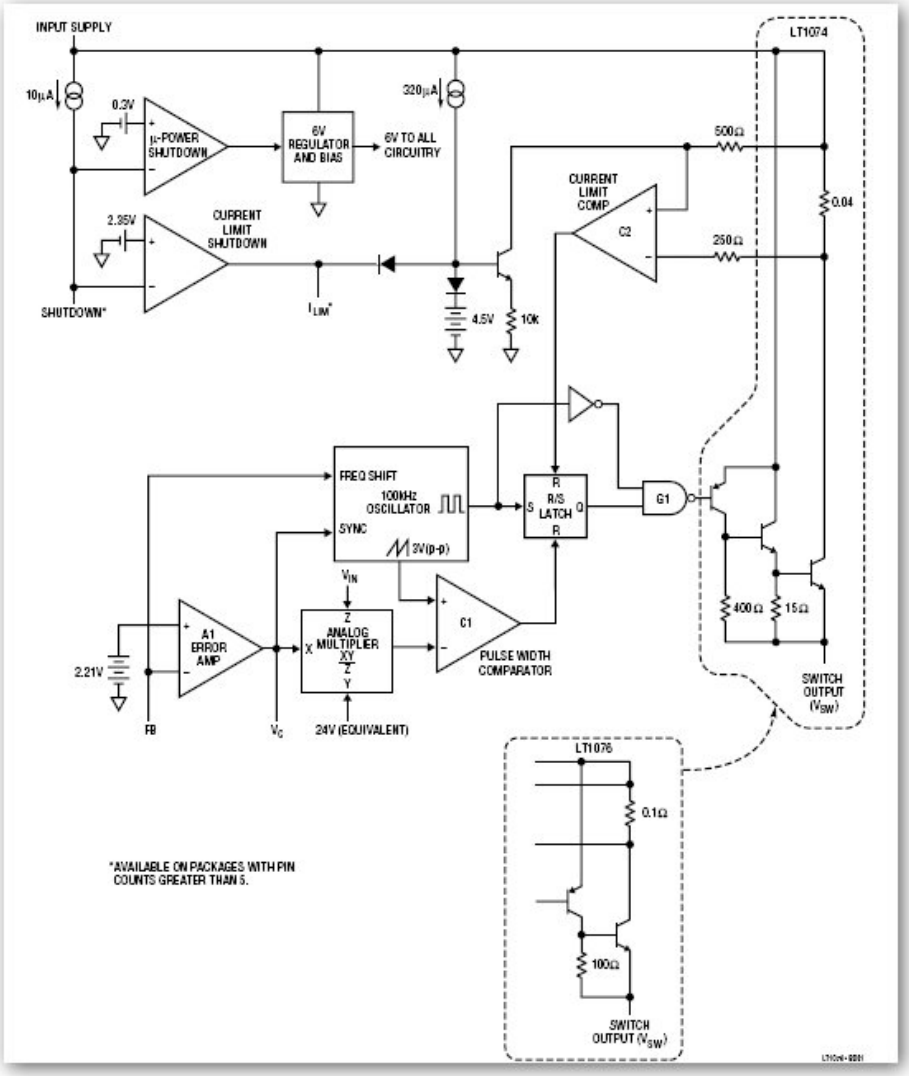


Figura 2.6: Diagrama de bloques del LT1074

Esquema de aplicación típica: Convertidor reductor positivo

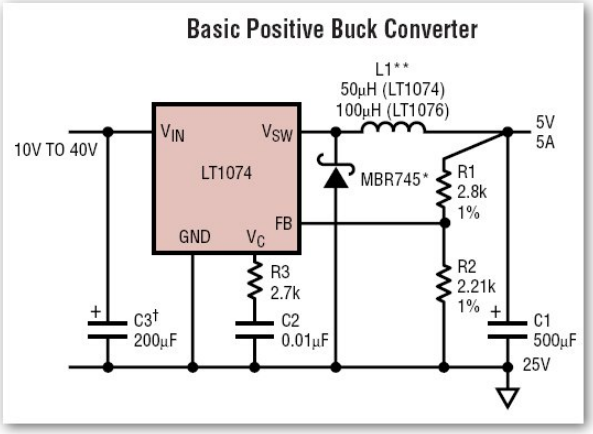


Figura 2.7: Convertidor reductor positivo usando el LT1074

Función que ejerce dentro de la tarjeta

Link Instruments ha dispuesto en la tarjeta de comunicaciones el esquema propuesto por el fabricante en la opción de reductor. Como tensión de entrada toma el pin B9, que es la tensión de 12V; y como salida presenta 5.54 V. Aunque aún no se ha hablado del consumo de corrientes se puede suponer que este reductor proporcionará una alta corriente de salida para el analizador.

En la figura 2.2está el esquema eléctrico para este reductor.

2.3.5. 74F08 Quad 2-Input and Gate

Descripción General

Se trata de 4 puertas lógicas tipo AND con una corriente de salida máxima de 20 mA.

Diagrama de conexión

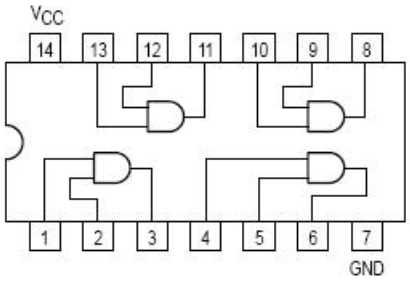


Figura 2.8: Puertas AND

Función que ejerce dentro de la tarjeta

Observando el esquema eléctrico figura 2.2 se puede ver que tanto $\overline{IOR}$ y $\overline{IOW}$ antes de pasar al analizador pasan por estas puertas. $\overline{IOR}$ es la entrada pin 1 y 2 de la puerta lógica cuya salida es el pin 3, entonces a la salida de esta puerta volvemos a tener la señal $\overline{IOR}$ pero con un pequeño desfase que según el datasheet del fabricante es de alrededor de 6ns. Por tanto, este integrado parece que introduce un pequeño retardo en las señales $\overline{IOR}$ y $\overline{IOW}$.

Observación: Al hacer el esquema eléctrico de la tarjeta se puede ver como el fabricante en un primer diseño llevaba las señales MEMR y MEMW hasta este integrado pero en una posterior corrección secciono las pistas que llevaban tales señales. Lo que da a pensar que el incluir en el 74F08 en la PCB final es un fallo de diseño y que su inclusión no influye en el resultado final de la aplicación hardware.

2.4. Señales de interés

Tras los apartados anteriores se deduce que de las 64 señales del bus ISA XT solo son útiles 23 de ellas, que se dividen en cuatro grupos como se indica en la tabla 2.8

Señales de Control
$\overline{IOW}$
$\overline{IOR}$
$\overline{SEN}$

Señales de Datos	Señales de Direcciones	Señales de Alimentación
Data 0	Addr0	+ 12 Vdc
Data 1	Addr 1	+ 5,4 Vdc
Data 2	Addr 2	+ 5 Vdc
Data 3	Addr 3	Gnd
Data 4	Addr 10	- 5 Vdc
Data 5	Addr 11	- 12 Vdc
Data 6		
Data 7		

Cuadro 2.8: Señales que llegan hasta el LA-4540

2.5. Cronograma de las señales de interés

Una vez conocidas las señales que intervienen en la comunicación entre PC y LA se puede llegar a explicar el protocolo de comunicaciones entre ambos. En el apartado 2.1 se hace especial interés en la señal ALE sin embargo al analizar el comportamiento de la tarjeta de expansión esta señal entra en la PALCE pero su valor no implica ningún cambio en la única señal de salida del la PALCE.

La salida de la PALCE es la correspondiente al pin out 19 que como se indica en la tabla 2.8 se denomina $\overline{SEN}$, se activa a nivel bajo cuando se cumplen los valores descritos en la tabla 2.7. La señal $\overline{SEN}$ activa el buffer triestado de transmisión de datos y también es una señal necesaria para controlar al LA-4540.

Otras señales necesarias para controlar al LA-4540, ya que son señales que llegan directamente hasta él, son $\overline{IOR}$ e $\overline{IOW}$. Por último recordar que $\overline{IOR}$ controla la dirección del buffer de transmisión de datos.

A continuación se muestran el cronograma de tiempos para transmitir un dato al LA, *cronograma de escritura*, y también el cronograma de tiempos para recibir un dato del LA, *cronograma de lectura*. En ambos se muestran las señales de control ($\overline{SEN}$, $\overline{IOR}$ e $\overline{IOW}$), el buffer de datos y el buffer de direcciones.

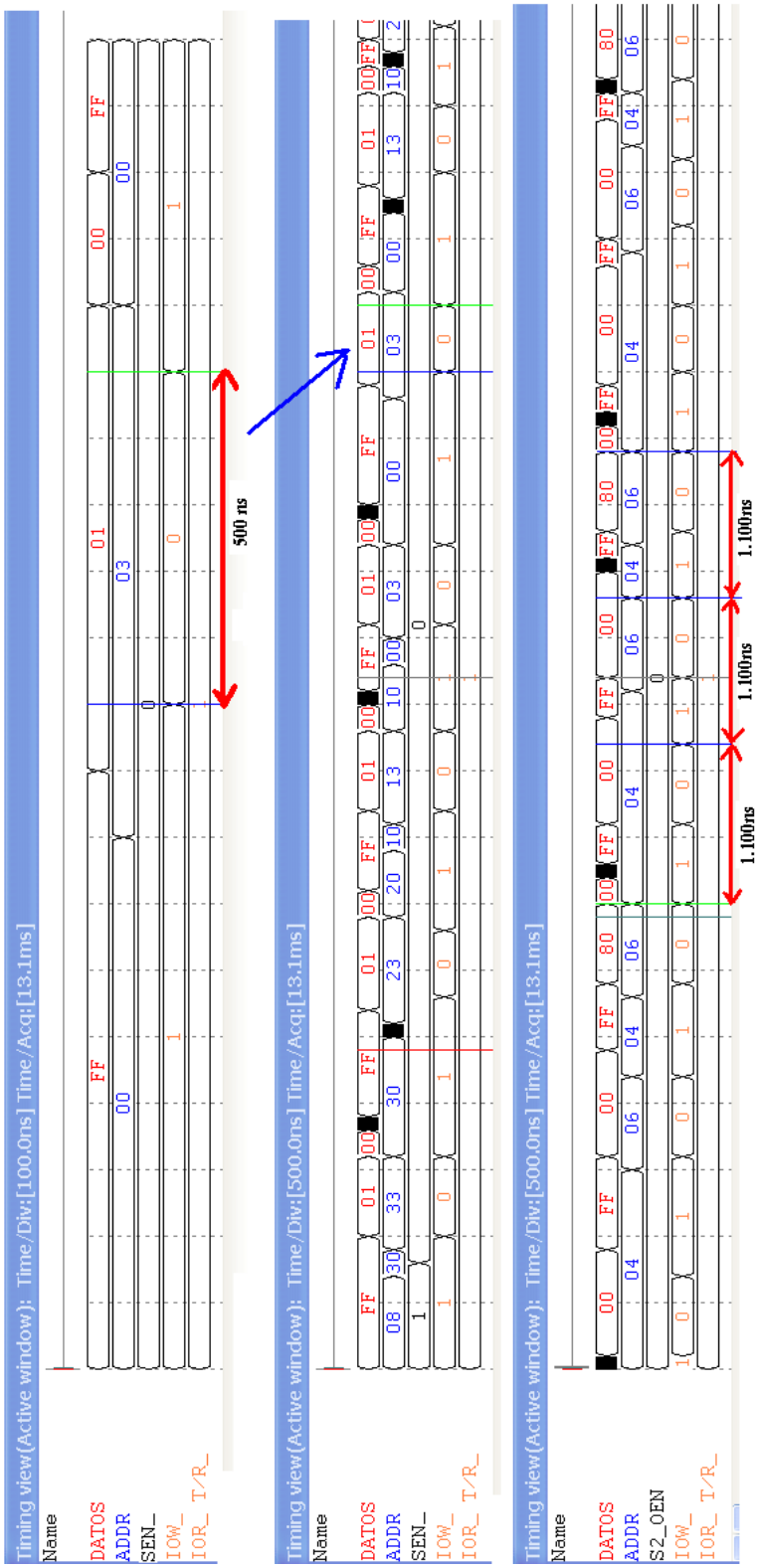


Figura 2.9: Cronograma de escritura ISA

Cronograma de escritura La figura 2.9 son varias capturas de la misma secuencia de tiempos pero con distinto zoom.

En la vista de tiempos superior el time/div es de 100ns, y se puede apreciar la siguiente secuencia de sucesos que dan lugar a una escritura en el periférico LA-4540:

- Lo primero es que la señal $\overline{SEN}$ sea estable a nivel bajo.
- Después se mantienen estables el bus de direcciones primero y a continuación el bus de datos.
- Por último, se activa a nivel bajo la señal de escritura $\overline{IOW}$ un tiempo mínimo de 500ns. Por tanto, se podría decir que el **tiempo hold o de mantenimiento para conseguir una escritura en el periférico LA-4540 es de 500ns.**

En la vista de tiempos del medio el time/div es de 500ns, y se puede apreciar la secuencia de *distintas escrituras consecutivas*; empezando a analizar la figura de izquierda a derecha se puede decir:

- En primer lugar se activa $\overline{SEN}$ a nivel bajo, se esta direccionando al LA-4540. Esta señal se mantiene activa durante todo el tiempo que se vaya a escribir o leer en el analizador.
- 100ns después se estabiliza el bus de direcciones y 200 ns después se estabiliza el bus de datos.
- A la vez que se estabiliza el bus de datos también se activa la señal $\overline{IOW}$ y se mantiene durante un tiempo mínimo de 500ns. Hay que advertir que no siempre la activación de $\overline{IOW}$ coincide con la estabilización del bus de datos sino que suele ocurrir que tarde 100ns más desde que esta última se estabilizo. A la vista de este primera escritura se podría decir que una escritura aislada tardaría un tiempo de 700ns, de los cuales 200ns son necesarios para cambiar los buses y los otros 500ns de tiempo de hold. Pero ¿cuánto tiempo se tarda en volver a hacer otra escritura?
- Tras analizar varios cronogramas de tiempo se concluye que el mejor caso de escrituras sucesivas lleva un tiempo de aproximadamente 1.100ns de los cuales 600ns son el tiempo necesario para cambios de señales de control y bus de datos y direcciones, y los otros 500ns se dedican a tiempo hold para que el periférico sea capaz de leer esos datos. Esto se puede ver en la ultima vista de tiempos en la que se muestran varias escrituras seguidas.

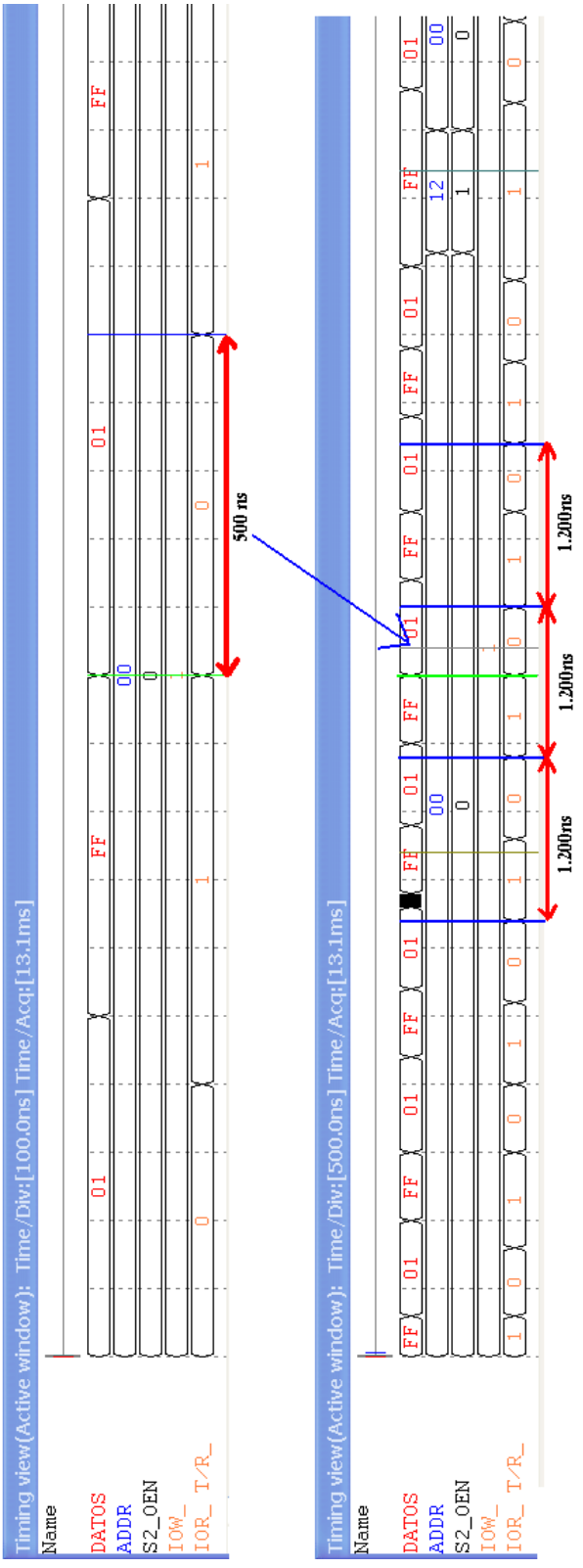


Figura 2.10: Cronograma de lectura ISA

Cronograma de lectura En la figura 2.10 se muestra el cronograma de tiempos para realizar una lectura; como en el anterior apartado se analiza el mejor caso para realizar una lectura, entendiendo como mejor caso el que lleva menos tiempo.

La captura superior con time/div de 100ns muestra el tiempo de hold mínimo que deben permanecer las señales para que el PC pueda leerlas.

- Se aprecia que en el mejor caso la señal $\overline{IOR}$ se activa justo cuando el bus de datos está estabilizado. En algunas capturas se ha podido observar como $\overline{IOR}$ se activa antes de que lo haga el bus de datos, sin embargo esto no es problema ya que el analizador se queda con el último valor estabilizado del bus de datos.
- Por tanto, se puede decir que **el tiempo hold para realizar una lectura del periférico es de 500ns.**

La siguiente captura con time/div de 500ns muestra una secuencia de lecturas consecutivas - otro caso es el de escritura y a continuación una lectura o viceversa -. El caso de lecturas consecutivas es el de mas interés, ya que es el caso que se da cuando se están recogiendo los datos de los pods.

- En esta captura se puede ver que el tiempo entre lecturas es de aproximadamente 1.200ns de los cuales 700ns son necesarios para cambiar los buses, en este caso sólo el de datos, y los otros 500ns son de tiempo de mantenimiento o hold.
- Viendo que para lecturas consecutivas se necesitan 1.200ns para cada dato y que existen 5 pods con una memoria de 8K o 128K se puede conocer el tiempo usado para extraer los datos del analizador.
  - Siendo $tiempo_{128K} = 5 \cdot 128 \cdot 1024 \cdot 1{,}200ns = 0'786432$ segundos para extraer la información de los 5 pods con una memoria de 128K.
  - Y para el caso de los 5 pods con una memoria de 8K cada uno de ellos el tiempo sería de $tiempo_{8K} = 5 \cdot 8 \cdot 1024 \cdot 1{,}200ns = 0'049152$ segundos.

3 Software

Como se ha expuesto en capítulos anteriores el fabricante ha puesto en manos del usuario tres fuentes distintas de software. Cada una de ellas enfoca el manejo del analizador lógico desde una plataforma diferente. Las fuentes de software originales disponibles son:

- Código ensamblador: ATE Macro Language
- Ejecutable para MS-DOS: ->`MAIN.EXE`
- Paquete instalador para Windows: ->`LA.EXE`

En la fase preliminar del proyecto se consideran todas las posibilidades de software que puedan aportar una posible línea de trabajo para conocer el funcionamiento del analizador lógico a bajo nivel. Es necesario realizar un estudio cuyo resultado permita a un usuario configurar el analizador lógico y poder capturar datos usando sus diferentes modos de funcionamiento y opciones. Por este motivo se procede a una fase de análisis de las herramientas software disponibles para el manejo del LA.

3.1. Análisis del software existente

El cauce deseable de trabajo sería poseer el código fuente que corresponda al manejo del analizador lógico. Partiendo de éste código fuente se podría compilar de tal manera que permitiese manejar el analizador sin desperdiciar ninguna de sus capacidades de funcionamiento. Además, este código sería fácilmente integrable en una aplicación desarrollada en lenguaje C y basada en la librería SDL, la cual se asemejaría en aspecto a cualquier otra aplicación de Windows o Linux y permitiría un control total del instrumento.

Por todos estos motivos se abre una fase de análisis del software disponible, el cual se encuentra en el disco de instalación facilitado por el fabricante del instrumento en el momento de su compra.

3.1.1. Código ATE

El fichero principal es el `ATE.ASM` cuyo contenido es código en ensamblador compilable. Se parte también del fichero `USER. ASM`, el cual contiene los parámetros de configuración del programa. Para el uso de esta técnica se requiere de conocimientos de programación en lenguajes C y ensamblador.

Para utilizar el fichero `ATE.ASM` se compila el fichero con Borland Assembler TASM y se crea el `ATE.EXE`. Se realiza lo mismo con el fichero `USER.ASM.` Después ha de usarse `EXE2BIN` para convertir el fichero `exe` en el fichero `USER.BIN`. Esta operación crea un fichero binario que ha de encontrarse en el mismo directorio que el archivo `exe`. De esta manera, al ejecutar el archivo `ATE.EXE`, el programa cargará los parámetros de configuración almacenados en el fichero binario. Todas estas operaciones pueden automatizarse mediante la creación de un fichero *makefile*.

Analizando en mayor profundidad el contenido del fichero `ATE.ASM` pueden distinguirse diferentes partes de código:

- Área de control: contiene parámetros relativos a si el fichero de configuración se cargará y el nombre del fichero (se deben realizar las oportunas modificaciones sobre el fichero `USER.ASM` para que se pueda cargar correctamente).
- Área de parámetros: especifica la configuración de opciones de funcionamiento del LA por defecto si no se cargan otras opciones mediante el fichero de configuración. Son opciones tales como la velocidad de adquisición, la configuración del trigger, tensiones umbrales...
- Área de datos de visualización: información relativa a la forma de presentación de los datos. Se reconocen opciones como: zoom, posición de cursores, posición de pantalla...
- Área de código de programa: contiene el código que realiza la comunicación con el analizador lógico. Se reconocen partes de código dedicadas a diferentes tareas: carga del fichero de configuración, configuración inicial del LA, configuración de trigger, petición de datos de los pods, configuración del generador de patrones...

Conocidos todos estos datos se pasa al estudio en profundidad del código, compilando los ficheros de la forma descrita y pasando a su depuración. Para ello se utiliza la herramienta de depuración Turbo Debugger, la cual permite monitorizar y controlar un programa durante su ejecución. Mediante esta herramienta se puede ejecutar un fichero `exe` hasta una

posición de memoria concreta, mostrando en ese momento el contenido de los registros del microprocesador y de la memoria, permitiendo por tanto el análisis de cualquier variable que se emplee en dicho instante. Es de especial utilidad la monitorización simultánea del código ensamblador que se ejecuta, información de depuración contenida en el ejecutable, la cual sólo se encuentra disponible si se ha incluido durante su compilación. La herramienta TD se encuentra disponible en los PCs del laboratorio.

Durante el test de depuración se logran cargar los parámetros almacenados en un fichero, se transmiten los parámetros de configuración al LA, y se realiza lo que parece ser la configuración inicial del LA. En esta parte se comprueba si existe algún LA conectado bien en el bus ISA o en el puerto paralelo LPT. A continuación se transmite una primera ristra de comandos que corresponde a la inicialización del analizador.

Se comienzan a reconocer grandes subrutinas que se encargan de la configuración de opciones de funcionamiento del analizador lógico tales como la configuración de palabras de trigger, nivel de trigger, configuración de frecuencia de muestreo... Conforme se va depurando se encuentra en el código la configuración de opciones de funcionamiento descritas en el manual del analizador.

En este momento se advierte que el código ATE parece incompleto ya que no se encuentra la forma de realizar capturas mediante el mismo. Se han depurado la mayoría de las líneas del fichero pero falta una parte de control en el código que realice la acción GO (comienzo de captura de datos) disponible en los programas `MAIN.EXE` para MS-DOS o su versión para Windows. Se reconocen otros grandes bloques de procedimientos dentro del fichero ATE dedicados a otras tareas de configuración del LA pero se desconoce la forma de lograr capturar datos.

Este problema hace que se busque información en la página web de Link Instruments para contrastar si existe otro código ATE diferente al suministrado en el disco original. La respuesta es que el código del que se dispone en el disco corresponde al suministrado para otra serie de analizadores lógicos anterior denominada LA-32xxx. De hecho, no existe código ATE disponible para el analizador LA-4540 por lo que no será posible utilizar este software.

3.1.2. Versiones para Windows

Conocida esta línea de trabajo se plantea repetir el procedimiento con el software existente para MS-DOS. Como no se tiene información a cerca del código fuente se pretende decompilar el fichero ejecutable `MAIN.EXE` para obtener el código original y poder depurarlo. De esta forma se podría obtener un código depurable que configurase al LA para hacer capturas.

Se busca un software capaz de realizar esta tarea y se prueban varios desensambladores, adoptándose como la solución más completa el programa IDA Pro 4.9 cuya versión además es gratuita.

El desensamblador multiprocesador IDA Pro es una herramienta capaz de desensamblar y depurar código. La estrategia que utiliza es ejecutar bajo su control el fichero `exe` objeto de estudio. Mientras crea un mapa de saltos de ejecución, transcribiendo el código ejecutable a instrucciones. Es capaz de reconocer variables, etiquetas de salto, llamadas a funciones, bucles... Incluso puede reconocer el lenguaje con el que fue ensamblado originalmente el fichero `exe`, lo que le permite aumentar su capacidad de reconocimiento del código original ensamblado. Todas estas facilidades quedan agrupadas en las siguientes ventanas principales: depuración, gráficos, llamadas a funciones, vista hexadecimal, variables de pila, referencias a strings, estructuras, y tabla de saltos.

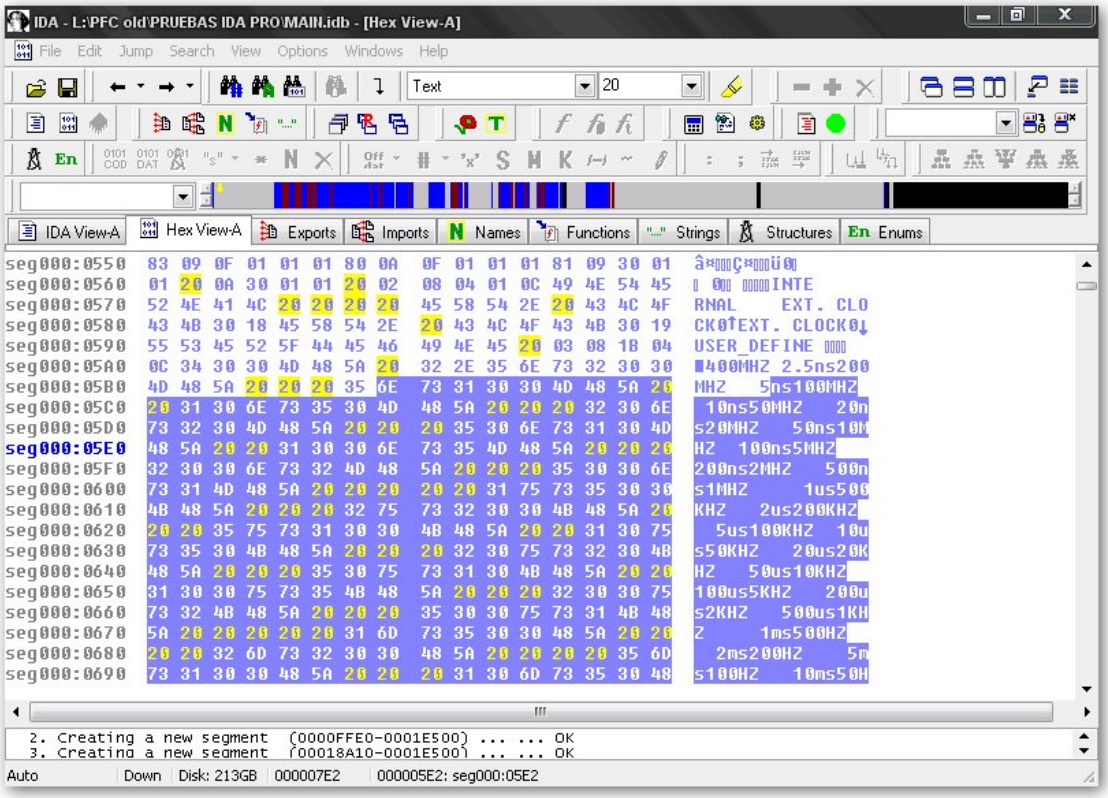


Figura 3.1: IDA Pro: Ventana principal

El resultado obtenido con este proceso es el código desensamblado, el cual es fácilmente

depurable con la misma herramienta. Sin embargo, la falta de información de depuración o su encriptado en el proceso de compilación original hace que se obtenga un código poco legible. Ahora no se reconocen de forma sencilla las diferentes áreas de código en el fichero decompilado. No existe información alguna a cerca de las subrutinas de configuración del LA.

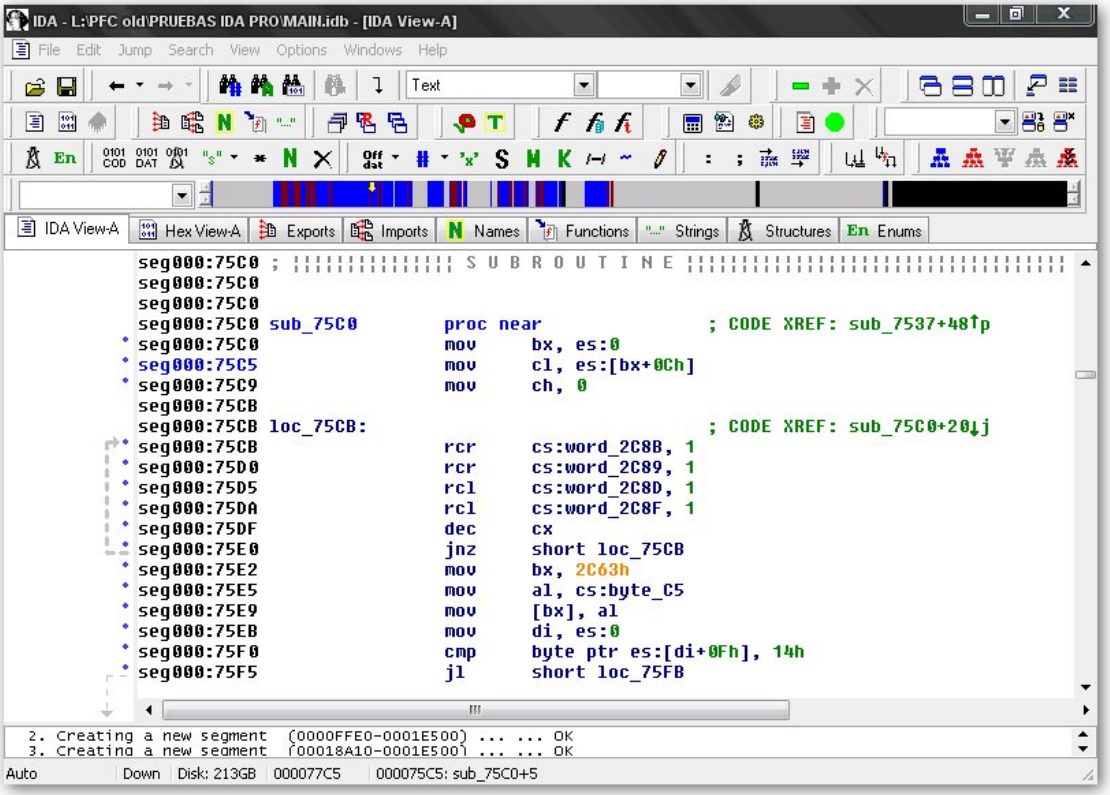


Figura 3.2: IDA Pro: Desensamblado de `MAIN.EXE`

Tras un proceso de comparación del código desensamblado con el código ATE disponible se procede a intentar comentar el código de forma que sea más legible. Se utilizan en este momento las facilidades de la herramienta IDA Pro que permite reemplazar elementos del código e introducir información de depuración. Se permite cambiar el nombre a las variables, poner nombre a las subrutinas, cambiar el nombre a las etiquetas de salto... Los ficheros no tienen el mismo orden en cuanto a la estructura de las subrutinas pero se comienza a depurar y a comentar el código con ayuda del código ATE existente.

Esta tarea se hace ardua y penosa. Las subrutinas de los dos ficheros no son comparables ya que el proceso de desensamblado no produce el mismo código que el fichero ATE del que se dispone. Por este motivo se hace prácticamente imposible reconocer las rutinas de con-

figuración del LA. Seguir en esta línea de trabajo sería un proceso demasiado costoso y no aseguraría el éxito por lo que se deshecha esta opción.

No obstante, conocida esta línea de trabajo, se decide realizar el mismo proceso de análisis con la versión del programa instalable para los sistemas operativos Windows. Se utiliza de nuevo el programa IDA Pro para decompilar el fichero disponible `LA.EXE`.

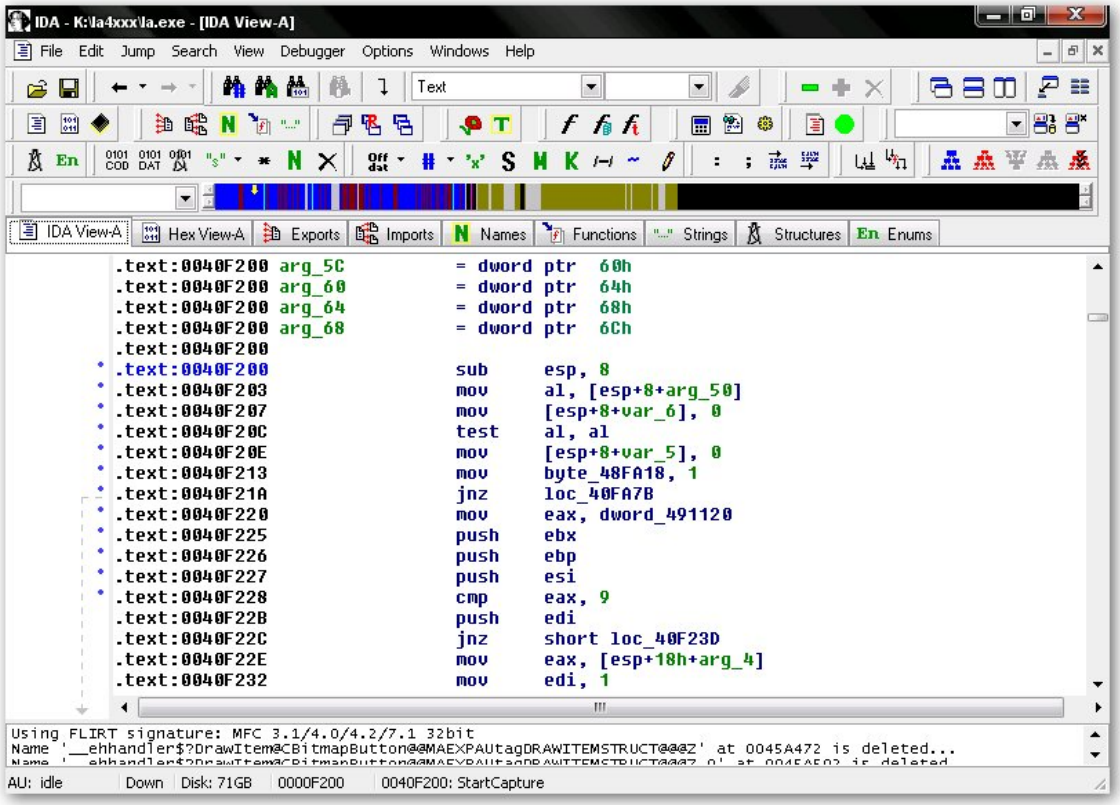
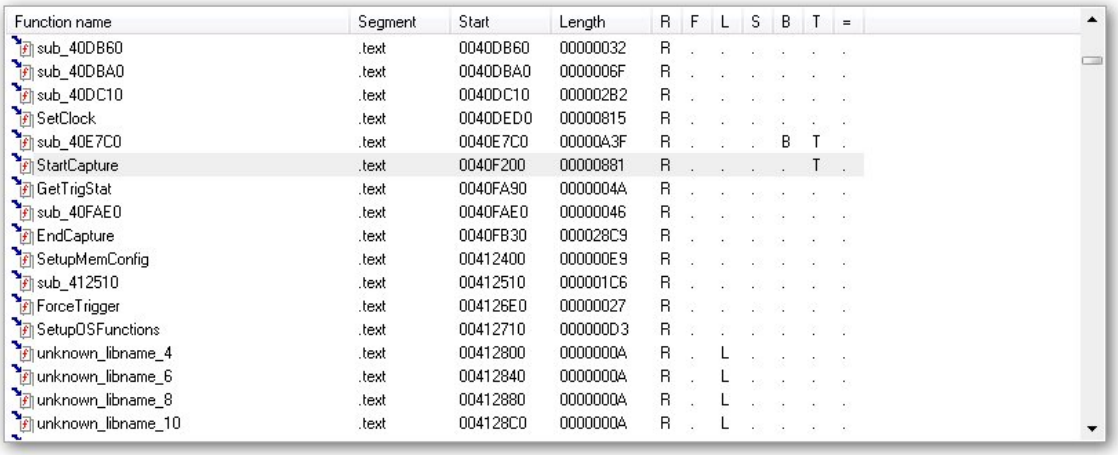


Figura 3.3: IDA Pro: Desensamblado de *LA.EXE*

El resultado utilizando las opciones que IDA Pro escoge por defecto nos muestra que la aplicación fue desarrollada en el entorno Microsoft Visual C++. Se observa que algunas de las librerías utilizadas para su desarrollo fueron *KERNEL32*, *GDI32*, *ADVAPI32*, *MSVCRT*, *MFC42*, y *USER32*. Se obtiene una información más completa que en el caso del fichero desensamblado anteriormente. En en lo relativo al manejo del LA aparecen más nombres de funciones, llamadas a funciones, mejor reconocimiento de etiquetas de salto de programa, y un mejor reconocimiento de las variables de programa. En lo relativo a la ejecución del programa y su entorno se reconocen las llamadas explicitas a las librerías mencionadas anteriormente y

su interacción con el SO Windows.



Function name	Segment	Start	Length	R	F	L	S	B	T	=
sub_40DB60	.text	0040DB60	00000032	R	.	.	.	.	.	.
sub_40DBA0	.text	0040DBA0	0000006F	R	.	.	.	.	.	.
sub_40DC10	.text	0040DC10	000002B2	R	.	.	.	.	.	.
SetClock	.text	0040DED0	00000815	R	.	.	.	.	.	.
sub_40E7C0	.text	0040E7C0	00000A3F	R	.	.	.	B	T	.
StartCapture	.text	0040F200	00000881	R	.	.	.	.	T	.
GetTrigStat	.text	0040FA90	0000004A	R	.	.	.	.	.	.
sub_40FAE0	.text	0040FAE0	00000046	R	.	.	.	.	.	.
EndCapture	.text	0040FB30	000028C9	R	.	.	.	.	.	.
SetupMemConfig	.text	00412400	000000E9	R	.	.	.	.	.	.
sub_412510	.text	00412510	000001C6	R	.	.	.	.	.	.
ForceTrigger	.text	004126E0	00000027	R	.	.	.	.	.	.
SetupOSFunctions	.text	00412710	000000D3	R	.	.	.	.	.	.
unknown_libname_4	.text	00412800	0000000A	R	.	L	.	.	.	.
unknown_libname_6	.text	00412840	0000000A	R	.	L	.	.	.	.
unknown_libname_8	.text	00412880	0000000A	R	.	L	.	.	.	.
unknown_libname_10	.text	004128C0	0000000A	R	.	L	.	.	.	.

Figura 3.4: IDA Pro: Detalle de funciones

El código desensamblado aparece más completo pero sigue estando vacío de información de depuración. Además ahora son reconocidas llamadas a funciones del sistema operativo por lo que la depuración del código se hace más compleja.

Para entender el motivo de su mayor complejidad ha de tenerse en cuenta que la aplicación corre bajo un SO Windows que no es de tiempo real. La aplicación realiza las pertinentes llamadas al SO para comunicarse con el analizador generando tareas que son atendidas por el planificador de tareas. Éste las atiende según la carga de procesos existente en el sistema y su prioridad. En este contexto puede entenderse que su depuración se hace compleja y larga ya que cada petición del software del analizador realiza una llamada al sistema operativo, lo que desencadena la llamada de otras muchas subrutinas que interactúan directamente con los servicios del SO.

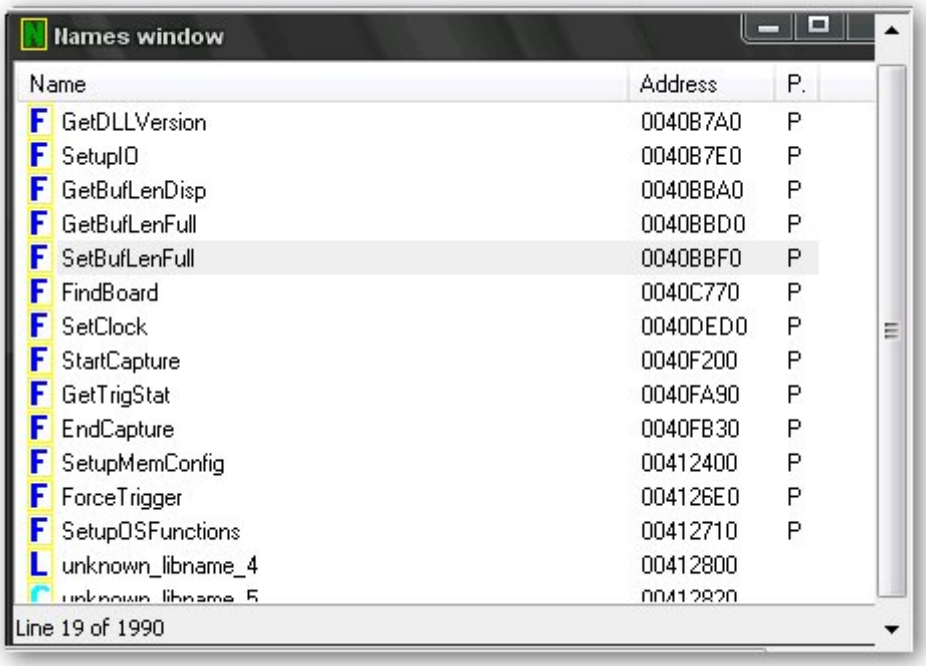


Figura 3.5: IDA Pro: Detalle de nombre de funciones reconocidas

Tras el estudio preliminar de este código se pasa a su depuración. Entonces puede verse de nuevo que este camino es demasiado complejo como para distinguir a simple vista qué código corresponde a la configuración de las diferentes funciones del analizador. No es posible por tanto conocer de forma precisa el código fuente de la aplicación. Además, el hecho de no disponer de la información de depuración complementaria dificulta aún más la tarea de depuración por lo que, como en el caso anterior, se decide abandonar esta línea de trabajo.

Llegados a este punto se hace necesario buscar otro camino que permita averiguar el funcionamiento a bajo nivel del instrumento. Se concluye que la ingeniería inversa aplicada al software anterior ha tenido poca aplicación práctica, por lo que se cambia de estrategia y se decide aplicar la ingeniería inversa directamente sobre el analizador lógico.

3.2. Ingeniería inversa

La ingeniería inversa consiste en obtener información a partir de un producto con el fin de determinar de que esta hecho, que lo hace funcionar, y como fue fabricado. Éste método implica avanzar en sentido contrario al desarrollo del producto de forma que analizando su comportamiento y sus componentes pueda llegar a concluirse como se hace funcionar.

El producto a estudiar se centra en este caso en el hardware compuesto por un LA-4540 y su tarjeta de comunicaciones ISA, y en el software que se encarga de manejar dicho instrumental.

La nueva estrategia a seguir consiste en monitorizar las señales que intervienen en el protocolo de comunicación entre el PC y el analizador lógico. Dichas señales se pueden agrupar en señales de direcciones, control, y datos. Cada comando que se transmite al analizador estará compuesto por tanto de dirección y dato. Las líneas de control cambiarán de valor según el protocolo ISA produciendo la escritura o lectura de un dato en una dirección concreta.

3.2.1. Montaje del hardware

Para la realización del estudio se disponen dos PCs provistos de su correspondiente analizador lógico y tarjeta de comunicaciones. El primer PC se emplea para realizar capturas de señales conocidas mientras que el segundo se emplea para monitorizar el protocolo entre el primer PC y su analizador lógico.

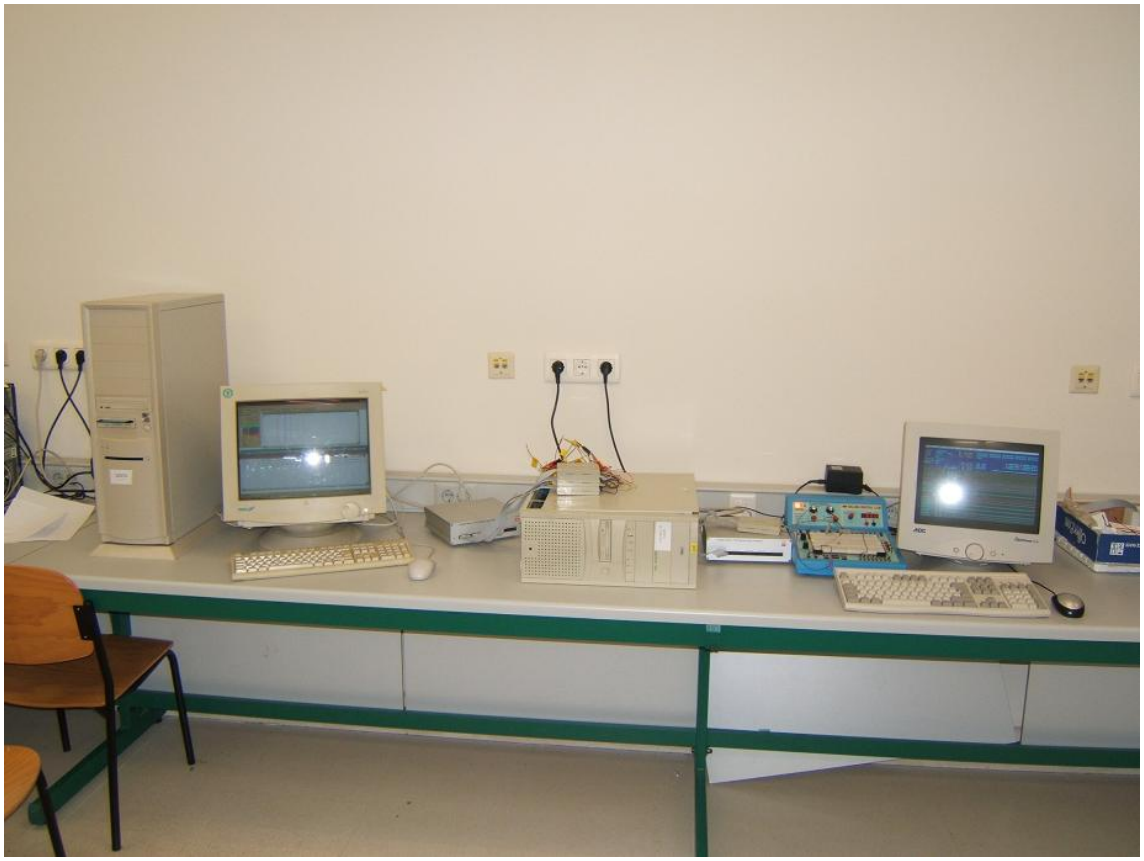


Figura 3.6: PCs en el laboratorio

El montaje del primer PC consiste en un analizador lógico con uno o varios pods conectados a señales generadas por el entrenador Función Generator IDL-800 de Digital Lab. El analizador lógico se conecta al PC mediante su correspondiente tarjeta de comunicaciones, la cual ha sido convenientemente modificada para poder monitorizar el protocolo de comunicación. Las señales se pinchan en un punto concreto de la tarjeta: entre las salidas de los integrados y el conector SCSI. Se escoge esta solución por sencillez en el momento de cablearlas y porque de este modo las señales se observan exactamente como el analizador lógico las recibe, con los mismos retardos y el mismo nivel de señal. Las señales se cablean soldando trozos de cable de unos 10 cm en los puntos adecuados de la tarjeta y posteriormente se numeran para facilitar su identificación una vez que la tarjeta se monte en el PC.

Figura 3.7: PC número 1

Las señales que se monitorizan son las siguientes: ADDR11, ADDR10, ADDR3-ADDR0, DATA7-DATA0, $\overline{IOR}$, $\overline{IOW}$, $\overline{SEN}$, y GND.

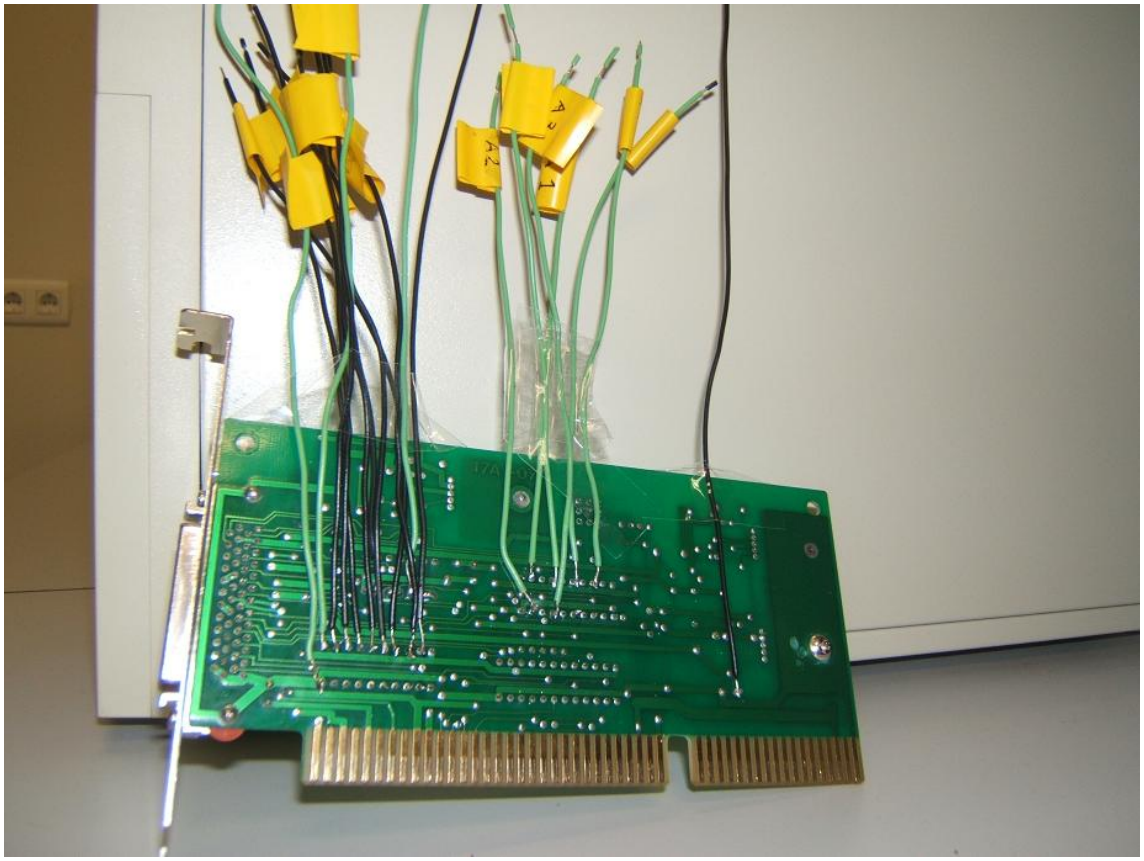


Figura 3.8: Tarjeta de comunicaciones cableada

El montaje del segundo PC es un montaje típico para LA-4540 con tres pods del tipo *Logic Pod* conectados. La diferencia estriba en que los pods del analizador lógico se utilizan para muestrear las señales anteriormente cableadas.

Figura 3.9: PC número 2

La forma en que las señales se conectan a los pods es un caso particular y debe tenerse en cuenta ya que todos los estudios posteriores se referirán a este esquema de conexión. Los canales de los pods se conectan de la siguiente forma:

CANAL	POD 1	POD 2	POD 3
CH0	Data0	$\overline{SEN}$	Addr10
CH1	Data1	$\overline{IOW}$	Addr11
CH2	Data2	$\overline{IOR}$	
CH3	Data3		
CH4	Data4	Addr0	
CH5	Data5	Addr1	
CH6	Data6	Addr2	
CH7	Data7	Addr3	
GND	Gnd	Gnd	Gnd

Cuadro 3.1: Esquema de conexión de los pods

Además de las señales lógicas de interés los pods lógicos deben estar convenientemente puestos a masa cableando en todos ellos su entrada de masa a la señal GND, también cableada

en la tarjeta.

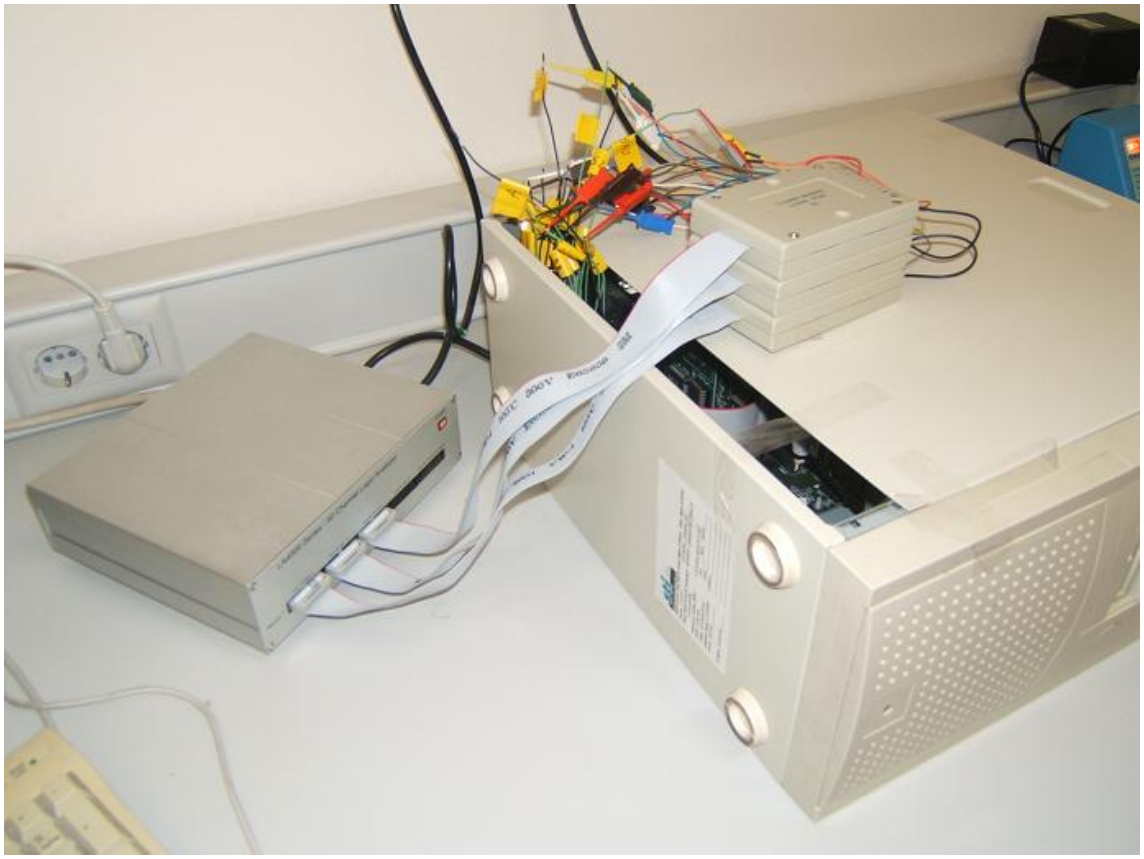


Figura 3.10: Detalle de conexión de los pods

3.2.2. Requisitos de configuración software

Una vez adoptado este montaje se procede a monitorizar el protocolo de comunicación. En primera instancia se hace correr el software `LA.EXE` en los dos PCs y se realizan pruebas cambiando opciones en el programa para determinar la estrategia de trabajo óptima. Esta estrategia debe perseguir un único fin: **no puede haber pérdida de datos en la captura del protocolo**. Este requisito es indispensable si lo que se pretende conseguir es que las ristras de comandos hagan funcionar un analizador. Una ristra de comandos defectuosa sería aquella en la que se hubiera perdido información de configuración del LA, lo cual no permitiría el correcto manejo del aparato. Por este motivo, los datos obtenidos a partir de las capturas realizadas sobre el protocolo deben ser consistentes. Además, se hace necesario guardar las capturas del protocolo para analizarlas con posterioridad y extraer la información

útil de configuración del analizador.

Las opciones disponibles en el programa que están relacionadas con estos requisitos son las siguientes:

Data LOG Permite guardar ficheros con el resultado de las capturas realizadas en los formatos LA, CSV, I2C, y PRN.

Memory Mode Tamaño del buffer almacenado de 8K o 128K.

Rate Frecuencia de adquisición de datos, desde 1Hz hasta 500MHz.

De estas opciones se desprende que los ficheros almacenados por el programa deben ser fácilmente tratables para extraer la información pertinente relativa a los comandos dirigidos al analizador. Debe notarse que al capturar todos los sucesos que ocurren en el bus ISA los ficheros contendrán mucha información no válida ya que existen tiempos muertos en los que el PC no se comunica con el analizador, pudiéndose comunicar con cualquier otro dispositivo conectado en el bus. Por este motivo los ficheros deberán tener el mayor tamaño posible para asegurar que los ficheros contienen todos los comandos transmitidos. De la misma manera, el ritmo de adquisición de los datos del protocolo debe ser un compromiso entre la velocidad mínima posible para que el fichero contenga el menor número de datos repetidos y la velocidad suficiente como para que no exista pérdida de información. Realizadas las pruebas pertinentes se conviene el uso de los siguientes parámetros:

- Los archivos LOG se guardan con el formato LA ya que es un formato fácilmente tratable mediante lenguaje C. Los ficheros pueden abrirse y leerse de forma sencilla en modo binario para extraer la información capturada contenida en el buffer de adquisición.

- La memoria seleccionada es 128K. Es la máxima posible, para asegurar que no existe pérdida de información.

- La frecuencia de captura se fija en 10MHz. Es la mínima posible ya que como se conoce la tasa de muestreo debe ser al menos la de Nyquist, el doble de la velocidad de la señal a muestrear. Ya que las especificaciones del bus ISA determinan que su frecuencia de funcionamiento es de 4,77 MHz, la tasa de Nyquist son 9,54 MHz. La velocidad de captura inmediatamente superior disponible es de 10MHz, razón por la que se escoge.

A partir de este momento la tarea se centra en ejecutar una determinada acción en el software del primer PC que produzca una comunicación con su analizador. Mientras, el segundo PC escucha la información transmitida en el bus ISA y la almacena en un fichero de extensión `la`. Con los parámetros seleccionados se obtiene un buffer de datos que corresponde a un tiempo de adquisición de 13,1 ms. Por tanto los ficheros la contendrán información del protocolo de comunicación perteneciente a 13,1 ms. La dificultad (como se verá más adelante) radica en que el buffer de adquisición es muy pequeño como para poder almacenar todo el protocolo de comunicación generado.

La única variable por especificar es qué versión de software será la encargada de manejar el analizador lógico del primer PC. En un principio se opta por comprobar el funcionamiento del analizador tanto con la versión para MS-DOS `MAIN.EXE` como con la disponible para Windows `LA.EXE` para poder elegir una de las dos opciones.

Las pruebas preliminares realizadas para ambas versiones muestran que cada programa tiene una forma propia de manejar el analizador lógico. El código de comandos que emplea cada versión es completamente diferente, incluso cada programa puede o no enviar diferentes ristras de comandos cuando se configura alguna opción de funcionamiento. Se debe escoger una de las dos versiones para realizar el estudio de funcionamiento ya que parece incompatible la mezcla de código de una versión con otra.

La versión de software escogida para el primer PC es `MAIN.EXE` para MS-DOS ya que se consideran importantes los siguientes criterios:

- El código desensamblado mediante IDA Pro es el de mayor parecido con el código ATE disponible, el cual podría ser útil posteriormente en el descubrimiento del funcionamiento del LA.
- El programa puede ejecutarse bajo MS-DOS. La ejecución del software en un SO con poca carga de procesos puede ser una ventaja, contribuyéndose a que la información del protocolo aparezca de forma más continuada en el bus ISA. Los tiempos de espera producidos por las llamadas a servicios del SO serán menores que en Windows por lo que se podrá ver más información en un único fichero de captura de datos, que tan solo podrá contener información de 13,1 ms de datos en el bus ISA.

En primera instancia se utilizará una configuración específica de opciones de funcionamiento

del primer analizador para obtener un caso base de funcionamiento. El primer analizador se configura con las opciones de captura de frecuencia 20KHz y tamaño de memoria de 8k. El modo de captura será *ONCE*, cuyo funcionamiento parece más sencillo. Posteriormente se realizarán pequeñas modificaciones graduales sobre estas opciones base para descubrir los cambios de configuración de cada opción.

En el segundo PC se opta por utilizar el software suministrado para Windows debido a las mejoras del entorno gráfico, permitiendo un mejor análisis de los datos capturados.

3.2.3. Estrategia final adoptada

Finalmente, tras el proceso de investigación expuesto con anterioridad se decide utilizar la siguiente configuración:

- Primer PC: tarjeta de comunicaciones cableada para muestrear el bus ISA del PC, analizador LA-4540 muestreando señales provenientes del entrenador Función Generator IDL-800 Digital Lab, programa *MAIN.EXE* encuitándose bajo sistema operativo MS-DOS, configuración base de frecuencia de captura 20KHz y tamaño de buffer de memoria 8k.
- Segundo PC: tarjeta de comunicaciones, analizador LA-4540 muestreando la tarjeta de comunicaciones cableada del primer PC, programa `LA.EXE` ejecutándose bajo SO Windows y capturando con buffer de 128K y frecuencia de muestreo de 10MSa.

3.2.4. Formato de los ficheros LOG.LA

Los ficheros de formato `la` pueden visualizarse mediante cualquier herramienta de procesado de texto. En su contenido se distinguen agrupaciones de datos precedidas de una cabecera que tiene el formato: [CABECERA]. Los datos que se almacenan después de esa cabecera pueden ser de dos tipos: variables de programa o datos capturados. Atendiendo a esa clasificación el fichero puede dividirse en dos partes:

En una primera parte del fichero se encuentran variables de programa, de fichero, de visualización, y de opciones de la aplicación `LA.EXE`. Las cabeceras que contienen dichas variables son: *File*, *TimingWin*, *StateWin*, *SerialWin*, *Cursor DLG*, *Cursor*, *Trigger*, *Clock*, *Search*, *Pattern*, *Group0*, *Group1*, *Group2*, *Group3*, *Group4*, *ChannelData*, *Screen*, *Glitch*, *Pulse*, *Hardware*, *Print*, y *Capture*. El contenido de estos datos puede editarse directamente en el fichero.

Por ejemplo, si se desea cambiar la palabra de trigger 0 basta con modificar la línea correspondiente con los valores posibles para una palabra de trigger: '0', '1', o 'X'. En la figura 3.11 se escriben dos ceros a modo de ejemplo.

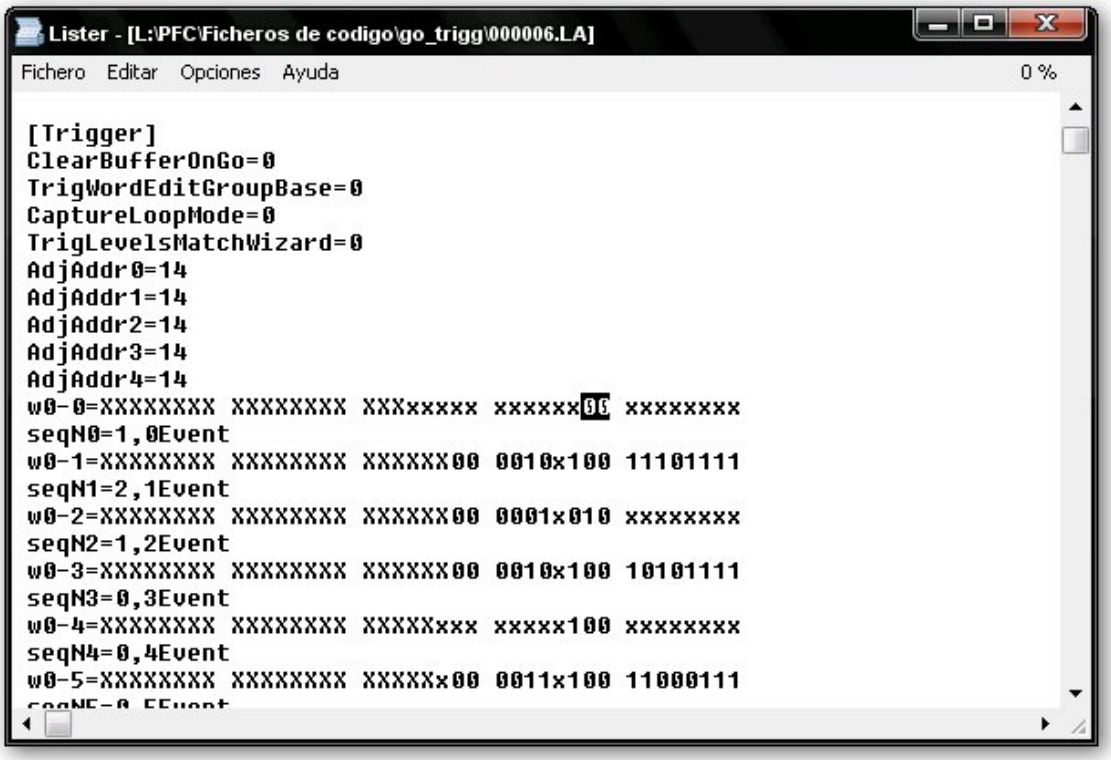


Figura 3.11: Fichero `LA`: Detalle de opciones de fichero

Una vez realizado y guardado el cambio el fichero puede ser cargado por la aplicación y mostrar en pantalla el cambio de la opción modificada.

En una segunda parte del fichero que comienza con la cabecera tipo *[Model LA-4540-128K 1 0]* (que indica el modelo de analizador y el tamaño de memoria usada) se almacena el contenido de la última captura realizada en el momento de salvar el fichero. Los datos se agrupan por pod y son almacenados en formato ANSI. Cada byte corresponde a una muestra capturada, y su valor decimal es el dado por el juego de caracteres ANSI estándar. Por ejemplo, acudiendo a la tabla de caracteres el símbolo ***ÿ*** corresponde al valor decimal *255* (11111111 en binario) e indica que en la muestra capturada el nivel de las señales de entrada en todos los canales del pod fue un *"1"* lógico.

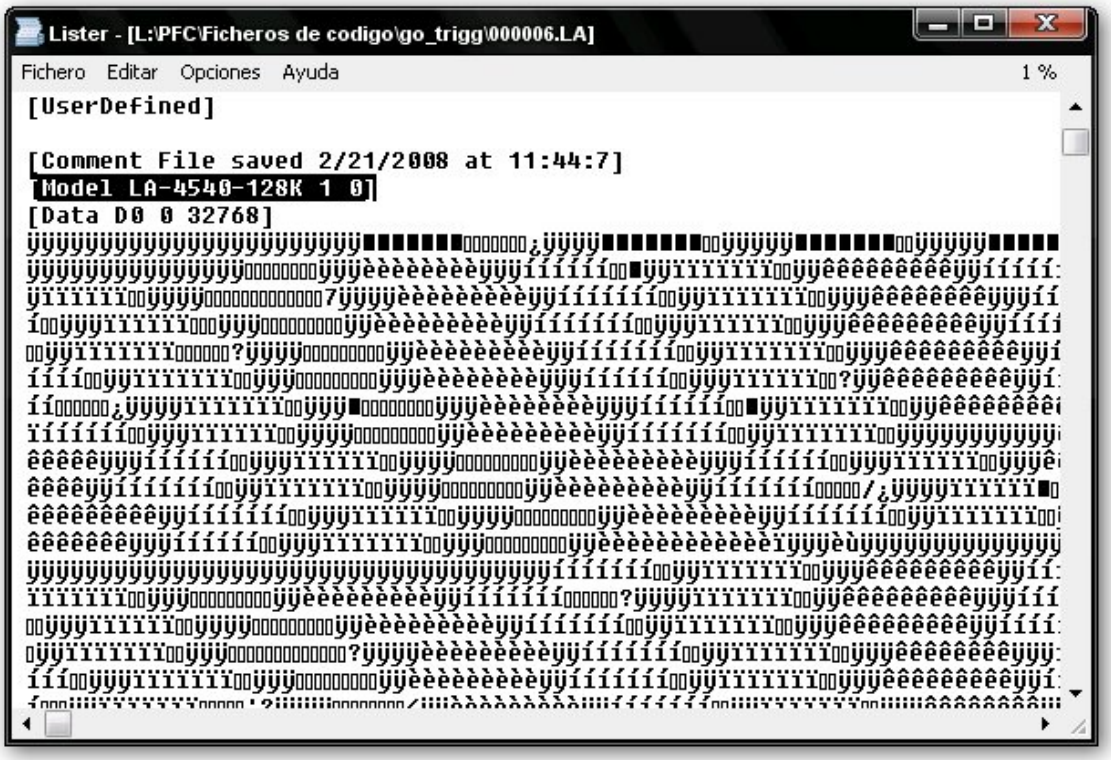


Figura 3.12: Fichero `LA`: Detalle de datos capturados

La información de interés es la almacenada en los pods 1, pod 2, y pod 3. Con las opciones de captura seleccionadas se puede observar que los datos aparecen agrupados en bloques de 32k bytes. En cada pod existen por tanto cuatro cabeceras que delimitan bloques de 32k bytes de datos. Dichas cabeceras son las siguientes:

Pod 1: [Data D0 0 32768], [Data D0 32768 32768], [Data D0 65536 32768], [Data D0 98304 32768]

Pod 2: [Data D1 0 32768], [Data D1 32768 32768], [Data D1 65536 32768], [Data D1 98304 32768]

Pod 3: [Data D3 0 32768], [Data D3 32768 32768], [Data D3 65536 32768], [Data D3 98304 32768]

Entre todos estos bloques se contendrán los comandos del protocolo de comunicación del analizador lógico capturados en un buffer de 13,1 ms.

3.2.5. Metodología de obtención de los ficheros LOG.LA

Los ficheros objeto de estudio se obtienen mediante el analizador conectado en el segundo PC. Para su correcto análisis posterior los pods de adquisición de datos se conectan de la forma descrita en el esquema de conexiones de la tabla 3.1. Una vez dispuesto el hardware de forma correcta se procede a dar las claves necesarias para poder obtener todos los ficheros de configuración. Se recuerda que el software tanto del PC número 1 como del PC número 2 debe estar configurado con las opciones descritas en apartados anteriores.

Para mejorar la visualización del protocolo las señales se agrupan de una forma específica. Se crean grupos de señales cuyo valor se visualiza en hexadecimal. Los grupos de señales que se muestran son:

- DATOS: Data7 - Data0.
- ADDR: Addr11, Addr10, Addr3 , Addr2 , Addr1 , Addr0.
- SEN_: $\overline{SEN}$.
- IOW_: $\overline{IOW}$.
- IOR_: $\overline{IOR}$.

La estrategia de trabajo es la siguiente. El software del segundo PC se configura como se muestra en la figura 3.13, con la opción *Data Log* activa y se pulsa el botón *Reset Counter* con el campo de nombre de fichero vacío. A continuación el segundo PC se pone en espera de capturar datos configurado con una secuencia de disparo concreta (trigger) y en modo de captura *NORMAL*. El primer PC se configura con el modo de captura *ONCE*. Acto seguido se realiza una acción en el software del primer PC (por ejemplo cambiar la velocidad de captura). El segundo PC se dispara cuando se produce el evento de trigger y realiza la captura de un buffer de datos. La consecuencia de esta acción es que se habrá guardado un fichero de nombre *000000.LA* en el disco del segundo PC en la ubicación predeterminada. El fichero será apto para su posterior análisis.

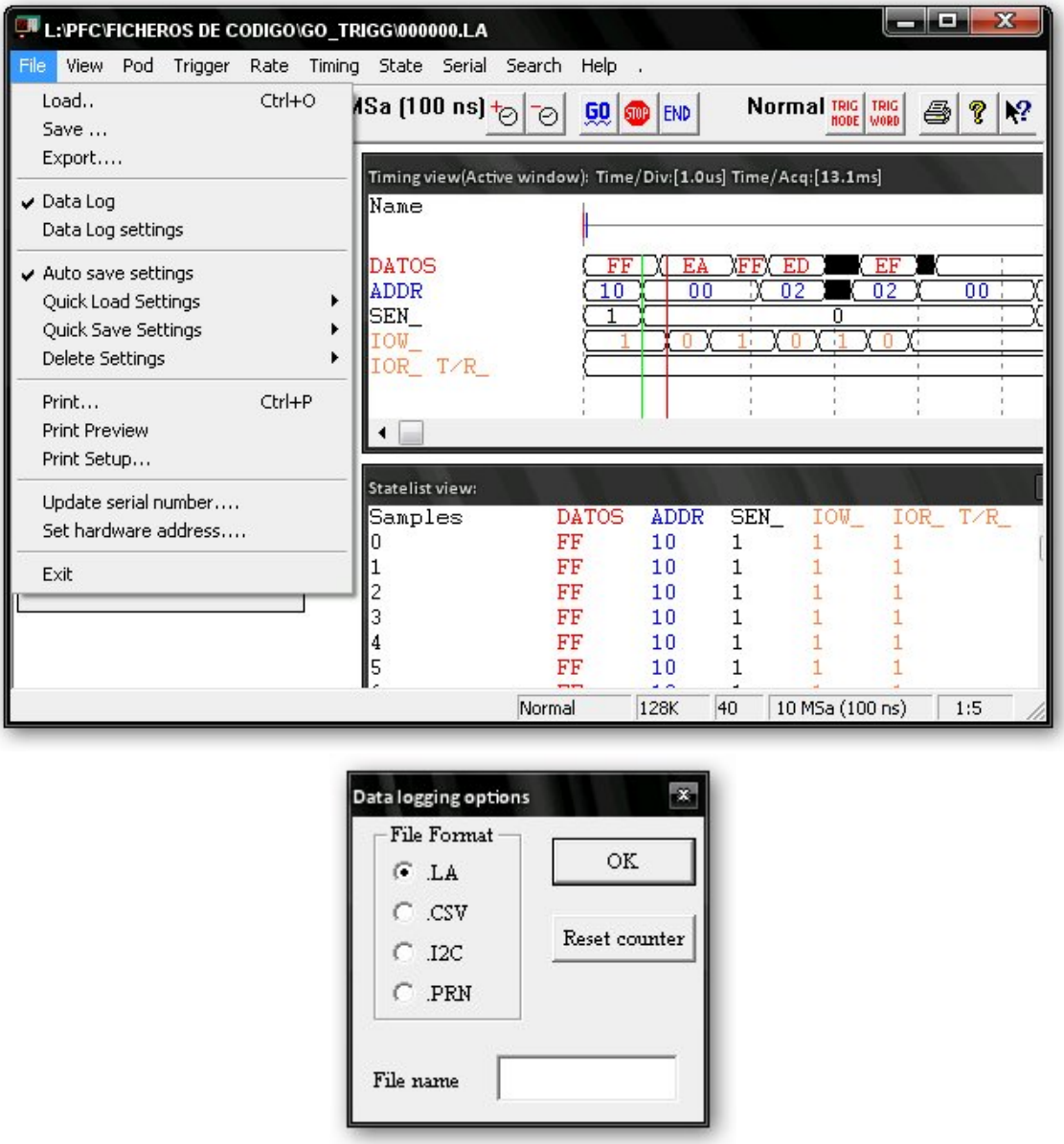


Figura 3.13: Detalles de configuración de *LA.EXE*

Para dar un ejemplo completo de cómo se realiza una la captura de protocolo compuesta por múltiples ficheros se supondrá que se desea obtener el protocolo de comunicación correspondiente a una única captura de un buffer de datos en el primer PC. En este caso de captura es necesario utilizar las capacidades disponibles de trigger del software del analizador. Se precisa que el analizador se dispare y comience a capturar datos en el mismo instante en que comienza

el protocolo de comunicación entre el software del primer PC y su analizador. Se da una breve explicación para entender como funciona la opción de disparo.

El patrón de trigger es utilizado para sincronizar la captura de datos con una porción de flujo de datos particular provenientes de un circuito de test. Se usa en los modos de funcionamiento de captura *ONCE* y *NORMAL*, y la posición de la palabra de trigger en el buffer de datos puede ser fijada mediante la función *Move Trigger*. En la ventana de trigger aparecen las 16 palabras de trigger susceptibles de ser usadas como eventos de disparo. Cada palabra puede ser un evento de disparo por si misma, siendo posible escoger la combinación de trigger words que forman una secuencia de eventos conjuntos.

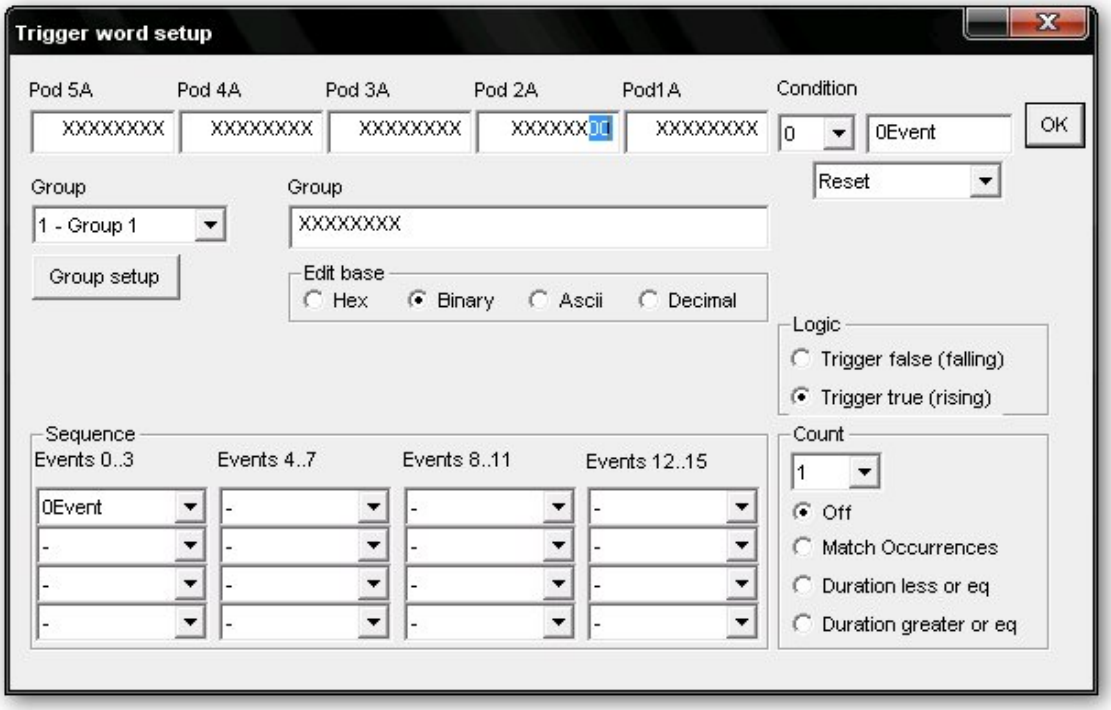


Figura 3.14: Detalle de ventana de trigger

La secuencia de sucesos que ocurre durante la captura de un buffer de datos desde el punto de vista del analizador del segundo PC utilizando esta característica es:

- El analizador se dispara al ocurrir la secuencia de trigger configurada.
- Se captura un buffer de datos de tamaño 128k muestras correspondiente a un tiempo de 13,1 ms.

- Los datos son transmitidos del LA al PC.
- El software almacena un fichero `log` con el buffer de datos capturados. Esta operación requiere un tiempo relativamente grande en comparación con el tiempo de captura de un buffer ya que implica un acceso a disco para almacenar el fichero.
- El analizador lógico se rearma con la secuencia de configuración de trigger y queda a la espera de que ocurra la siguiente secuencia de eventos de disparo.

Se ha comprobado que el protocolo de comunicación siempre comienza con una instrucción de escritura del PC por lo que el primer evento para obtener el comienzo de cualquier fichero es el que se muestra en la figura 3.14(véase un ciclo de escritura de bus ISA). Las señales $\overline{IOW}$ y $\overline{SEN}$ valen "0" lógico mientras que el resto de señales de interés tendrán un valor desconocido por lo que se rellenan con el valor poco importa "X". Utilizando la configuración del trigger con el *Evento 0* descrito se consigue alcanzar el primer comando que el PC envía al LA en cualquier caso.

Dispuesta esta configuración se ordena al primer PC la captura de un único un buffer de datos (modo de captura *ONCE*) pulsando sobre la opción GO el valor "YES". Mientras en el segundo PC se guardan varios ficheros `log` en disco. Analizando el contenido del primer fichero se comprueba que tan solo contiene 3 comandos de escritura. En el resto de ficheros el sincronismo se habrá perdido respecto a los 3 comandos de escritura debido al tiempo necesario para guardar cada fichero en disco y que el analizador se rearme. Por lo tanto, sólo se considera el primer fichero y el resto de ficheros se desechan. Se guarda a parte únicamente el primer fichero `000000.LA` para su posterior análisis y se configura el trigger para obtener el siguiente fichero en sincronismo. Como se desea que el LA se sincronice con el protocolo cuando ya hayan pasado tres comandos de escritura, la secuencia de captura se configura para que capture a partir del cuarto comando de escritura. Esto es configurar la secuencia de trigger con el *Evento 0* repetido cuatro veces.

A continuación se obtienen varios ficheros. Se guarda el primero de ellos como `000001.LA` y se configura el nuevo trigger. Utilizando la utilidad de búsqueda se observa que el último comando capturado en el buffer es una lectura sobre la dirección 0x01. En realidad, la dirección sobre la que se ha leído no es la que se muestra en el grupo ADDR sino la 0x211. Esto se explica porque el nibble de mayor peso del grupo está formado por las direcciones Addr11 y Addr10, mientras que el nibble de menor peso se compone de las direcciones Addr3-Addr0 (véase que la dirección real se compone según la tabla 2.7 de la página 28). El *Evento 1* se configura por

tanto como un comando de lectura con el grupo de señales ADDR a valor 0x01 y las señales de control $\overline{IOR}$ y $\overline{SEN}$ a nivel bajo. El siguiente fichero de captura comenzará también con un comando de escritura por lo que la secuencia de disparo queda como se muestra en la figura 3.15.

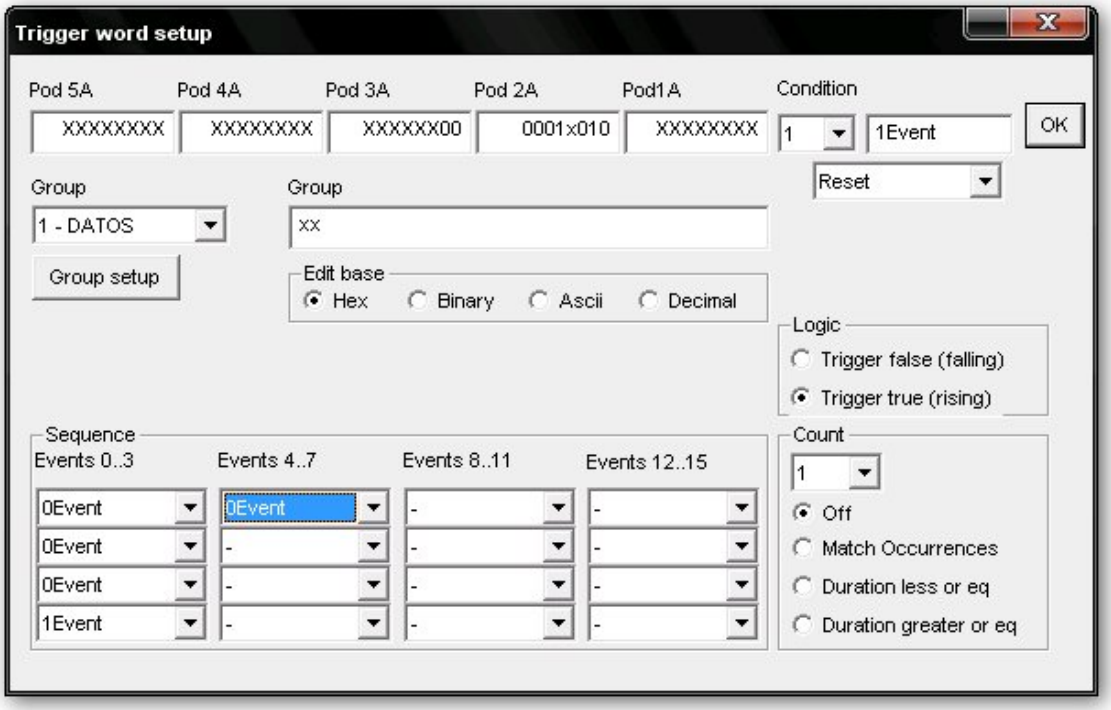


Figura 3.15: Detalle de configuración de secuencia de trigger

Con esta configuración se vuelven a obtener varios ficheros. El primero de ellos se guarda con el nombre `000002.LA` y se vuelven a desechar el resto. Observando el contenido del fichero el usuario puede darse cuenta de que los datos de captura del primer pod conectado al analizador del primer PC están contenidos en él. En primer lugar aparece una ristra de comandos de escritura y posteriormente los datos, que son leídos de la dirección real 0x210 (grupo de señales ADDR en valor 0x00). A continuación aparece una nueva ristra de comandos de escritura, y posteriormente los datos capturados por el segundo pod. El tamaño del buffer de captura no es suficiente para mostrar todos los datos leídos a través del segundo pod así que aparecen sesgados. Es necesario ampliar el número de eventos y configurar una nueva secuencia de disparo para obtener completamente los datos capturados a través del pod 2.

Se decide que la mejor estrategia es separar los datos de cada pod en un fichero log diferente

por lo que a la secuencia actual se añaden una serie de eventos según el pod que desee capturarse. Estos eventos deben ser únicos de tal manera que en el fichero log se logre la captura de la ristra de comandos de escritura que preceden a los datos. Estos comandos serán de especial interés para la conseguir el objetivo de poder manejar el analizador y capturar datos, y se consideran como las instrucciones de petición de pods. Los datos de los pods no tendrán interés por el momento. Bastará con saber que tienen un tamaño fijo de 8K ya que el software del analizador del primer PC se encuentra configurado con esa opción.

Llevando a cabo esta estrategia se repiten los pasos anteriores y se obtienen los ficheros de los datos capturados a través de los 5 pods conectados en el analizador lógico del primer PC. Los ficheros se guardan siguiendo la nomenclatura adoptada, donde cada pod queda almacenado en un fichero diferente. Los pasos se repiten hasta que el proceso sólo produce la captura de un único fichero log. Éste será el primer indicio de que se ha terminado de capturar un protocolo de transmisión completo.

El último de los ficheros obtenidos referente al pod 5 contendrá los comandos de petición del pod y los 8K de datos capturados a través del pod. Al final del mismo aparece una ristra de comandos de escritura que más tarde se identificará como los comandos de fin de configuración del LA, los cuales rellenan completamente el fichero log y llegan hasta el final, por lo que parecen sesgados. Para asegurarse de que la ristra se captura en su totalidad se configura una nueva secuencia mediante la técnica conocida para la obtención del último fichero. El proceso concluye con la obtención del último fichero del protocolo, el cual corresponde al fin de la configuración del analizador.

A modo de resumen se exponen los pasos necesarios para la captura de los ficheros log correspondientes a un protocolo de comunicación completo:

1. Configuración del software de los PCs. Primer PC preparado para realizar una acción. Segundo PC configurado con la secuencia de trigger compuesta por el *Evento 0.*

2. Ejecución de la acción en el primer PC. Uno o varios ficheros log son almacenados en el segundo PC.

3. Análisis del contenido de los ficheros log del segundo PC. El primero de los ficheros se guarda para su posterior análisis.

4. Configuración de la siguiente captura de ficheros log modificando la secuencia de eventos según el contenido del fichero log anterior.

5. Repetición de los pasos 2, 3 y 4 hasta que la acción en el primer PC sólo produzca un único fichero log.

De todo lo anteriormente expuesto puede extraerse que la metodología de trabajo para capturar el protocolo de comunicación completo entre el PC y un analizador lógico exige la captura de toda una secuencia de ficheros. En el ejemplo expuesto ha sido necesaria la obtención de 8 ficheros para lograr contener todo el protocolo de comunicación, y debe recordarse que tan solo se ha capturado un caso concreto de funcionamiento del analizador lógico. Posteriormente se explicará como influye el cambio de las diferentes opciones del programa `MAIN.EXE` en la configuración de analizador, pero puede adelantarse que será necesario capturar los mismos ficheros cada vez que se cambie una opción de funcionamiento en el programa principal. Si se tiene en cuenta que existen 26 frecuencias posibles para la adquisición de datos y que, para cada una de ellas es necesaria la captura de al menos 4 de los 8 ficheros mencionados anteriormente, el número total de ficheros que se deben capturar se dispara por encima de la centena. Es por tanto un proceso meticuloso que requiere de mucho tiempo y gran paciencia.

A través del ejemplo expuesto se han dado las claves de trabajo para obtener los diferentes ficheros de manejo del LA. Es una tarea ardua que requiere de gran meticulosidad, pero no tanta como para analizar el contenido de los ficheros. La siguiente fase del proyecto exige la extraer la secuencia de comandos que contienen los ficheros capturados. El gran volumen de ficheros generados hace necesaria la automatización del proceso de análisis por lo que se recurre a la programación como solución al problema.

3.2.6. Análisis de los ficheros LOG.LA

Para extraer la información de los ficheros se necesita una aplicación que se encargue de analizar los ficheros de formato .la de forma automática. La aplicación deberá ser capaz de abrir un fichero, leer la información de los pods de interés, y almacenar la información de forma legible en un nuevo fichero. Para cumplir con estos objetivos se crea una aplicación en lenguaje C a la que llamaremos *An*. Para lograr este objetivo se utiliza el entorno de desarrollo Dev-C++.

La aplicación *An* consiste en un conjunto de funciones y procedimientos que se encargan de realizar las operaciones de análisis y tratamiento de la información de ficheros de extensión `la`. Se ha desarrollado y adaptado progresivamente para contemplar las diferentes necesidades de análisis según precisaba el proyecto en cada una de sus etapas. Por tanto, *An* es flexible y puede proporcionar diferentes ficheros de salida en función del análisis deseado. Para cambiar

el formato de análisis se debe recompilar la aplicación incluyendo en su función principal las funciones o procedimientos de análisis deseados. Para explicar el funcionamiento de esta aplicación se incluye un diagrama de flujo que muestra las operaciones principales.

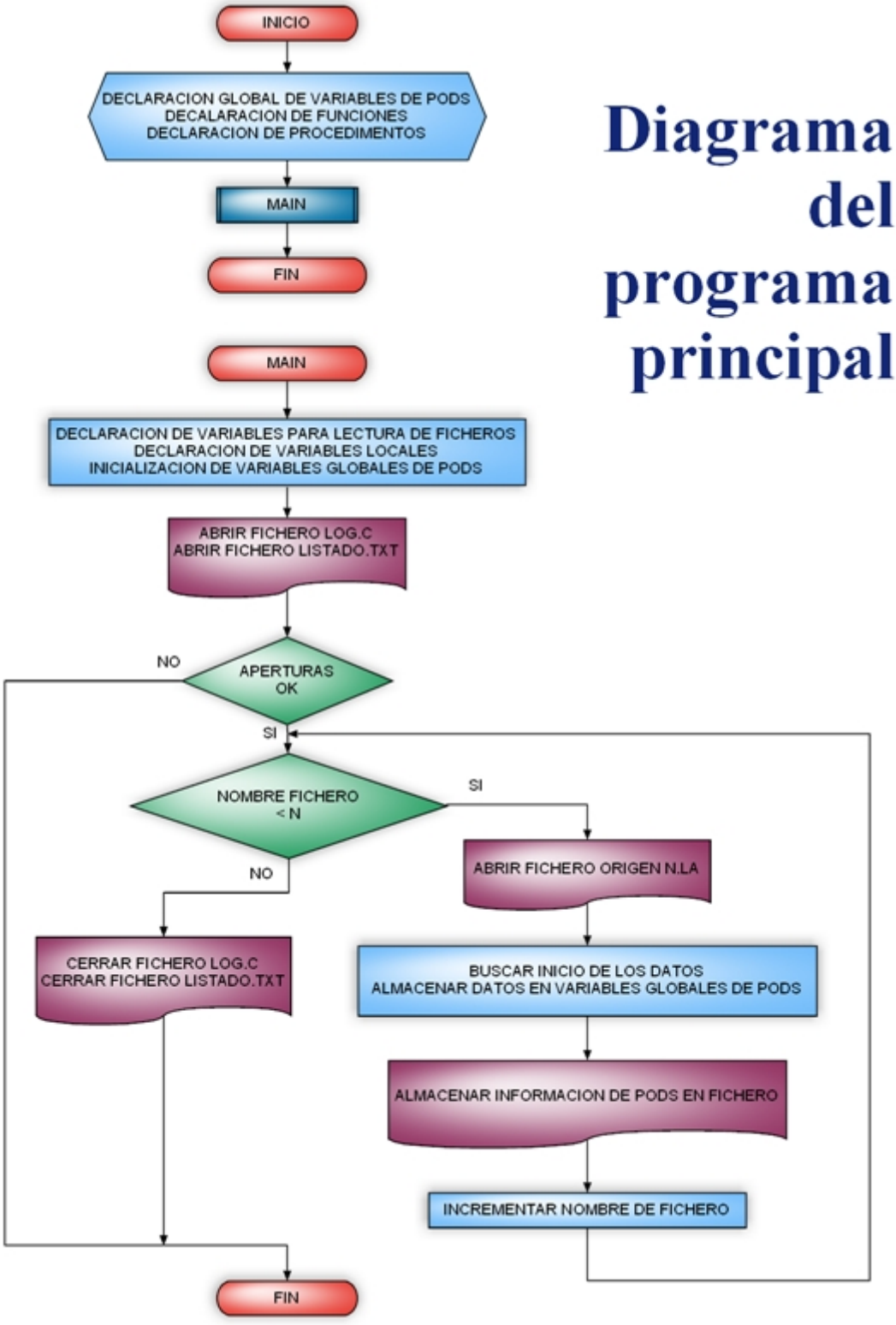


Figura 3.16: Diagrama de flujo: Flujo principal de *An*

El programa principal es el encargado de abrir los ficheros de salida. A continuación lee un fichero de nombre `000xxx.LA` y almacena su contenido en unas variables globales auxiliares que hacen referencia al pod1, pod2, y pod3. Posteriormente la información es tratada y escrita en ficheros. La manera en que la información aparece en los ficheros esta relacionada con la función que se encarga de tratarla y escribirla en los mismos. En términos generales, los ficheros de salida pueden darse en dos formatos: `listado.txt` *y* `log.c`.

Los ficheros `.txt` están orientados a almacenar de forma visual las muestras recogidas en el bus ISA para poder realizar comparaciones entre ficheros. En el fichero se escribe cada muestra en una línea. Y cada línea se desglosa en los campos de dirección, dato, y valores de las señales de control $\overline{IOR}$, $\overline{IOW}$, $\overline{SEN}$. Opcionalmente también pueden aparecer en la misma línea el número de muestra y un comentario que indica si la muestra es una escritura o lectura del analizador. Si es una escritura al final de la línea aparece la palabra "OUT" mientras que si es una lectura aparece "IN". Si por algún motivo se ha producido un error en la lectura del fichero origen y las señales de control aparecen todas activas a cero la palabra que aparece al final de la línea es "AMBAS". Si se diera este caso se debe pensar que el proceso de análisis del fichero falló, probablemente porque al guardar el fichero `.la` se produjo un error y el fichero tiene un pequeño defecto respecto a los ficheros estándar. Este caso se ha comprobado y puede distinguirse fácilmente ya que el fichero defectuoso tiene un tamaño diferente al del fichero estándar.

Los ficheros `.c` están enfocados a la creación de código ejecutable. En ellos se incluyen las líneas necesarias para poder ejecutar los ficheros utilizando el compilador Borland Compiler. La información del protocolo se escribe en el fichero en forma de comandos ejecutables en lenguaje C. Por este motivo, cada muestra de lectura y escritura del analizador contenida en el fichero `.la` se traduce en una operación de lectura y escritura en el fichero de extensión `.c`. De este modo, una línea OUT del fichero `.txt` se traduce en la instrucción *outportb(addr, data)*, mientras que una línea IN se traduce en la instrucción *inportb(addr)*. Este enfoque facilita en gran medida la obtención de las instrucciones que formarán los ficheros de manejo del analizador. Posteriormente estos ficheros podrán ser ejecutados mediante Borland Compiler en el primer PC para intentar manejar el analizador lógico sin ninguna interfaz gráfica intermediaria. Si se consigue un buen funcionamiento del LA se podrá dar por hecho que los ficheros son correctos, lográndose una interesante realimentación en la investigación del funcionamiento del instrumento.

Hasta ahora se conocen los formatos genéricos de presentación de la información. En cualquier caso, el origen de esta información es el mismo. Los datos de interés del proto-

colo a estudiar se encuentran almacenados en los ficheros `.la` como se describe en la sección anterior. Los caracteres ANSI almacenan el estado de los canales de captura de los pods en un instante concreto. Cada instante es una muestra y, en cada muestra se almacena información del protocolo de comunicación de la forma descrita en la tabla 3.1 del esquema de conexión de los pods. En la lectura ha de tenerse en cuenta cómo se agrupan las señales y qué peso ocupan dentro de los bytes para poder extraer las señales por separado y presentar la información de forma que sea inteligible a una persona. Como paso previo se realiza la extracción de los bytes de datos del fichero `.la` objeto de estudio y se copian a variables globales. De este modo los datos están disponibles para cualquiera de las posibles funciones que realizan la tarea de escribir los datos de forma amigable en los nuevos ficheros.

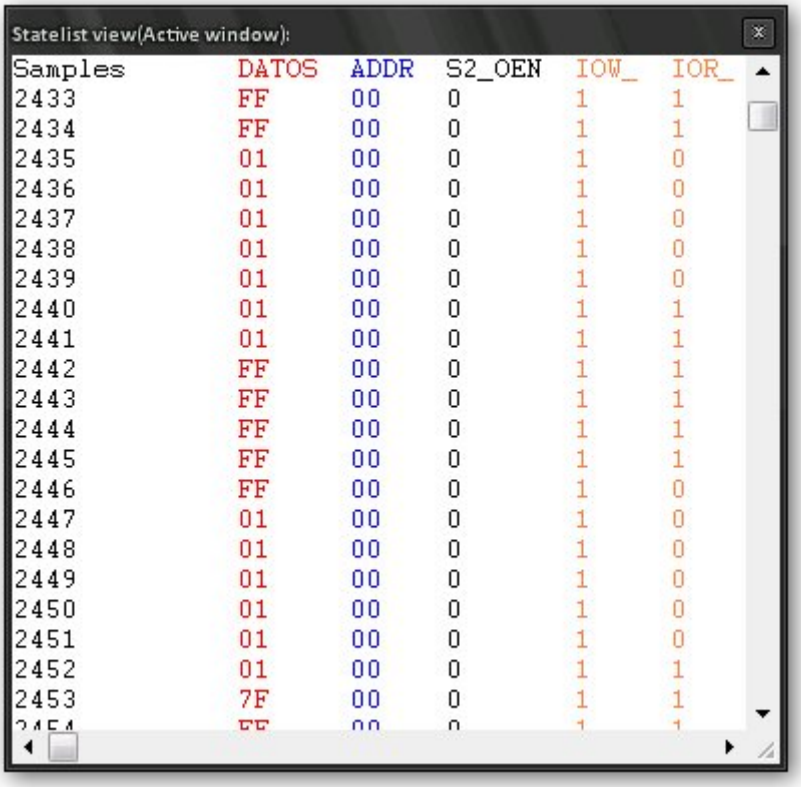
Hacer la información más legible es el paso previo fundamental que permite comparar posteriormente distintos ficheros de captura. Se compararán ficheros que contengan los mismos comandos y el mismo número de líneas, pero que tengan una única diferencia de configuración del analizador. El cambio de una única opción de funcionamiento hará que cambien múltiples líneas de comandos dentro de los ficheros. Dichos comandos se parametrizarán por medio de variables o matrices que hagan referencia a los cambios realizados en el manejo del LA. La comparación de ficheros será una parte importantísima dentro del proyecto, siendo una de las fases que más tiempo ha consumido.

Para realizar un correcto análisis de la información contenida en los ficheros `.la` debe tenerse en cuenta la forma en que se capturaron los datos en el PC2. El bus del PC1 se muestrea al menos al doble de la frecuencia de funcionamiento dada por la especificaciones del bus ISA, por lo que es de esperar que aparezcan más de dos muestras para cada ciclo del bus ISA del primer PC.

Experimentalmente se comprueba que el protocolo de comunicación está siendo sobremuestreado de tal forma que por cada ciclo de lectura o escritura en el bus ISA se capturan seis muestras consecutivas en el PC2. Por ejemplo, durante un ciclo de escritura del PC1 en el LA las señales de control $\overline{IOW}$ y $\overline{SEN}$ cambian su valor a nivel bajo y permanecen un tiempo de 500 ns en cada ciclo. Mientras el PC2 captura a una frecuencia de 10 MHz, es decir, realiza una nueva adquisición cada 100ns. Por tanto se capturan cinco o seis muestras consecutivas correspondientes a un único ciclo de escritura. Si el ciclo es de lectura se capturan también cinco o seis muestras y la diferencia se encuentra en que las señales que cambian de valor son $\overline{IOR}$ y $\overline{SEN}$.

Además, debido a retardos en las señales se comprueba que en la mayoría de las ocasiones los datos de las muestras capturadas no son estables desde la primera muestra. Primero bajan

las señales de control y no puede garantizarse hasta la tercera muestra del ciclo que el dato capturado es estable. Es el caso de un ciclo de lectura en el se lee un dato de la memoria del analizador. En la figura 3.17 se muestra un ejemplo de ciclo de lectura en el que se aprecia el retardo del dato capturado respecto al instante en el que bajan las líneas de control.



Samples	DATOS	ADDR	S2_OEN	IOW_	IOR_
2433	FF	00	0	1	1
2434	FF	00	0	1	1
2435	01	00	0	1	0
2436	01	00	0	1	0
2437	01	00	0	1	0
2438	01	00	0	1	0
2439	01	00	0	1	0
2440	01	00	0	1	1
2441	01	00	0	1	1
2442	FF	00	0	1	1
2443	FF	00	0	1	1
2444	FF	00	0	1	1
2445	FF	00	0	1	1
2446	FF	00	0	1	0
2447	01	00	0	1	0
2448	01	00	0	1	0
2449	01	00	0	1	0
2450	01	00	0	1	0
2451	01	00	0	1	0
2452	01	00	0	1	1
2453	7F	00	0	1	1

Figura 3.17: Retardo en los datos capturados

La figura 3.17 muestra dos ciclos de bus ISA de lectura. La señal $\overline{S2\_OEN}$ permanece a bajo nivel mientras que la señal $\overline{IOR}$ baja en cada ciclo de lectura. El primer ciclo de lectura se capturó con 5 muestras que van desde la muestra 2435 hasta la 2439. Se puede observar que es una captura correcta ya que en las muestras aparece estable la dirección 0x210 y el dato 0x01 es estable para todas las muestras. El segundo ciclo de lectura se capturó con 6 muestras que van desde la muestra 2446 hasta la muestra 2451. Se puede observar que es existe un retardo en las líneas de datos para la muestra 2446. Mientras que la dirección permanece estable el bus de datos permanece en alta impedancia (0xFF) y su valor no es correcto hasta la siguiente muestra número 2447. Por tanto, existe un retardo de los datos respecto a las líneas de control que experimentalmente se comprueba que puede oscilar entre una y dos muestras.

Puesto que el tiempo que las señales de control es cinco veces superior al que se muestrea podría plantearse la reducción de la velocidad de muestreo a la mitad (5 MHz). Sin embargo, se ha comprobado que no se puede garantizar que los datos capturados sean estables ya que sólo se capturarían entre dos y tres muestras por cada ciclo de bus ISA, y puede no ser estable la segunda muestra. Por este motivo y aunque el protocolo está siendo sobremuestreado, se decide mantener la frecuencia de captura actual del PC2 en 10 MHz para asegurar que en cada ciclo escritura capturado hay muestras de datos que son estables.

Como conclusión final puede extraerse que:

- Por cada ciclo de bus ISA se obtienen entre cinco y seis muestras de datos.
- La primera muestra estable es la tercera muestra de cada ciclo.

La aplicación de análisis de ficheros puede ser modificada en función de los ficheros de salida que deseen obtenerse y de cómo se desea que la información se organice en ellos.

En primera instancia la aplicación *An* se orientó a escribir de forma organizada toda la información capturada. De este modo, cada listado `txt` contenía la totalidad de datos del buffer de memoria, es decir, 131072 líneas (una por cada muestra). Esta tarea es realizada por medio de la función *escribir_todo*.

Sin embargo, el tamaño de los ficheros de salida no era adecuado para poder compararlos posteriormente entre sí. Por tanto, para poder comparar diferentes listados la aplicación *An* debe orientarse al tratamiento de la información de los pods extrayendo la tercera muestra de cada ciclo de bus ISA capturado y almacenándola en el listado txt. De esta forma, cada línea del fichero corresponde a un ciclo de bus ISA La organización de la información siguiendo esta pauta es realizada por la función *escribir_listado*.

Conocido el cometido de la aplicación *An* se procede ahora a la descripción de las principales funciones y procedimientos que son usadas para realizar el análisis de los ficheros.

void buscar_inicio(FILE *puntero) Busca el inicio de los datos dentro del fichero. El procedimiento recibe un puntero al fichero a analizar. Su funcionamiento se basa en que las cabeceras de datos siempre comienzan con la cabecera *[Data D0 0 32768]*. Se leen caracteres hasta que se encuentre una marca de inicio de cabecera *'['*. A continuación es suficiente con comprobar que la cabecera continua con la cadena de caracteres *"Dat"*. Se continúan leyendo caracteres hasta que se encuentra la marca de final de cabecera *']'*. Finalmente se leen dos caracteres correspondientes al salto de línea existente entre la cabecera y los datos. El puntero a fichero apunta al primer dato de los pods.

void rellenar_POD(FILE *pfich) Copia desde el fichero el contenido de los datos capturados a través de los pods en tres arrays globales, uno para cada pod. Se parte de un puntero que apunta el primer dato del primer pod. La forma de trabajar es copiar un bloque de 32K de datos en el array de un pod y ejecutar la función *buscar_inicio* para posicionarse en el siguiente bloque de datos. Esta operación se realiza un total de cuatro veces para llenar el array de datos correspondiente a un pod. A continuación se repite el proceso para los otros dos pods restantes.

void escribir_listado(FILE *pflist,char idfich[11],int conta0,int conta1) Extrae el listado de los comandos contenidos en la información almacenada en en los arrays de pods globales y lo almacena en el fichero de salida *listado.txt*. Recibe el puntero al fichero donde escribe y el nombre del fichero del que se ha obtenido la información de los pods. Sólo escribe la líneas que suponen un comando de lectura o escritura sobre el analizador. Si las señales $\overline{IOR}$ y $\overline{SEN}$ son activas a nivel bajo se escribe un IN, mientras que si las señales $\overline{IOW}$ y $\overline{SEN}$ son activas a nivel bajo se escribe un OUT. En este proceso debe tenerse en cuenta que el protocolo esta siendo sobremuestreado. Por cada vez que el PC envía un comando OUT o IN al analizador lógico la información capturada contiene aproximadamente seis muestras referentes a la misma instrucción. Además, debido a retardos en las señales de datos introducidos por la tarjeta de comunicaciones ISA, en ocasiones las señales de datos cambian su valor una o dos muestras más tarde que las señales de control. Por este motivo, en el fichero de salida sólo se escribe una línea por cada comando capturado, y siempre se escribe el contenido de la tercera muestra. Cada línea del fichero contiene la dirección real del comando, el dato leído o transmitido, el valor por separado de cada señal de control, y el texto "IN" o "OUT" según corresponda.

void escribir_todo(FILE *pflist,char idfich[11]) Escribe en el fichero `listado.txt` todo el contenido del protocolo de comunicación del bus ISA sin realizar ninguna modificación respecto a como fue capturado. Recibe el puntero al fichero donde escribe y el nombre del fichero del que se ha obtenido la información de los pods. En el fichero se escribe toda la información capturada en el bus ISA tal cual fue muestreada por lo que se observa que cada comando de lectura y escritura se escribe en el fichero seis veces aproximadamente debido al sobremuestreo. Cada línea que se escribe en el fichero contiene el numero de muestra, dirección, dato, y el valor por separado de cada señal de control. Si la línea escrita en el fichero es un comando de lectura o escritura sobre el LA se incluye el texto "IN" o "OUT" según corresponda para diferenciar que dicha línea es un comando.

void escribir_c(char idfich[11], FILE *fich9, int index1) Escribe en el fichero `log.c` la estructura de programa necesaria para que sea compilable y copia en su función principal los comandos "OUT" e "IN" contenidos en la información almacenada en las variables globales de los pods. Los prototipos de macros que implementan la escritura y lectura en un puerto del PC son los siguientes:

OUT int outportb(int addr_puerto, int valor);

IN unsigned char inportb(int id_puerto);

Por tanto, por cada muestra que contiene una instrucción "OUT" o "IN" se escribe en el fichero una instrucción outportb o inportb según proceda.

unsigned int ADDR(int i) Recibe el número de muestra (posición dentro del array de información de los pods) que se analiza y devuelve el valor de la dirección contenida en esa muestra.

unsigned int DATA(int i) Recibe el número de muestra que se analiza y devuelve el valor del dato contenido en esa muestra.

unsigned int IOR(int i) Recibe el número de muestra que se analiza y devuelve el valor de la señal $\overline{IOR}$ para esa muestra.

unsigned int IOW(int i) Recibe el número de muestra que se analiza y devuelve el valor de la señal $\overline{IOW}$ para esa muestra.

unsigned int S2_OEN(int i) Recibe el número de muestra que se analiza y devuelve el valor de la señal $\overline{SEN}$ para esa muestra.

NOTA* La extracción de la información de las señales desde las variables globales que contienen los datos capturados por los pods debe de realizarse de la misma forma que la información se almacena, siguiendo el esquema de conexión de la tabla 3.1.

NOTA** Los ficheros .la se abren en formato binario ya que es el formato que presenta menos problemas al tratar los datos. Debe tenerse en cuenta la particularidad de que un salto de línea escrito en ASCII se convierte en los códigos de control CR+LF cuando es leído de forma binaria en sistemas Windows y MS-DOS. El código CRLF no es más que la sucesión de dos caracteres uno detrás de otro, por tanto 0D 0A en hexadecimal.

3.3. Aplicación de la ingeniería inversa

3.3.1. Introducción

Ahora que ya se ha descrito en los apartados anteriores la metodología de trabajo, se pasa a aplicar la ingeniería inversa.

Como introducción se recuerda:

1. El PC1 trabaja con una versión del software en MS-DOS en modo captura 8K y el PC2 con una versión en Windows en modo captura 128K y frecuencia de muestreo 10MHz.

2. Los fragmentos de código de manejo del LA se obtendrán en bloques de 128K, cabe recordar que en una sola captura no se obtendrá el código completo para un comando debido a:

   - El valor de una misma muestra se repite varias veces, se esta sobremuestreando, en otras palabras un mismo out se repite mas de una vez.
   - Los tiempos perdidos para cambiar de dirección o datos.
   - Los tiempos que introduce el software en MS-DOS entre comunicaciones con el LA.

   Entonces aunque hay 131072 muestras por captura, el número de ellas es significativamente menor.

3. Por otro lado, cada vez que se llene la memoria de 128K el tándem hardware LA y software LA necesitará un tiempo extra debido a:

   - El tiempo para pedir esos datos desde el hardware LA hasta las variables software.
   - El tiempo para guardar los datos en un fichero, en el caso de estar activa la opción log.
   - El tiempo usado para refrescar en pantalla.
   - El tiempo necesario para volver a mandar los OUT's e IN's necesarios para a pedir que el LA vuelva a muestrear otros 128K.

   Esto implica que cuando en el PC1 se pulse una comando, por ejemplo el comando GO, en el PC2 se obtendrán varios archivos.log, sin embargo, sólo será válido el primero de ellos. Esto es debido a que hay un tiempo entre el 1^{er} `archivo.log` y el 2^{o} `archivo.log` donde se han perdido datos de interés.

   La solución a este problema pasa por utilizar la opción de trigger en el PC2.

4. En consecuencia con lo dicho en el punto 3 cuando se vaya a analizar un comando en PC2 realizado en el PC1 se procederá como se indica a continuación:

   a) **En PC2**: Se activa la opción de guardar cada captura en un archivo.log[1], se pulsa GO y como es el primer archivo que se va a obtener será un GO con una condición de disparo OUT ($\overline{SEN}$=$\overline{IOW}$=0 y $\overline{IOR}$=1)[2].

   b) **En PC1**: Se pulsa el comando del que queremos obtener el código. Hay que fijarse en que *estado de inicio* se encuentra el analizador en el momento de realizar ese GO.

   c) **En PC2**: Una vez realizada la captura se pulsa STOP y se analiza con el programa *AN.exe* el primer archivo.log de todos los obtenidos, observando cuales han sido las últimas instrucciones ejecutadas para fijarse en los aspectos de:

   - instrucción out o instrucción in
   - dirección
   - dato

   NOTA: Dado que cada comando outportb o inportb necesita un tiempo de mantenimiento o hold de 500ns como mínimo- esto se puede ver en la figura 2.9o 2.10 - cuando se muestree el bus ISA del PC1 a la frecuencia de 10Mhz - 100ns - desde el PC2 se obtienen como mínimo 5 muestras por comando. Es necesario configurar el ejecutable AN.exe para que si una muestra se repite mas de una vez solo aparezca escrita una vez en el archivo listado - esto se explica mas detalladamente en el apartado 3.2.6 - y de esta forma cada comando outport o inport corresponde con una única linea del archivo listado.

   Esas últimas instrucciones se ponen como condición de trigger en el PC2.

   Ahora se va a comprobar si después de ese archivo existe otro o por el contrario el comando se ha obtenido de forma completa.

   d) **En PC1**: Se vuelve a buscar el *estado de inicio* cuando se pulso el GO.

   e) **En PC2**: se pulsa Go con la condición de disparo establecida en 4*c*.

   f) Volvemos al paso 4*b*,repitiendo esta secuencia hasta comprobar que ya no hay ficheros nuevos para ese mismo comando.

[1] La opción de archivos LOG debería activarse desde un principio y resetear cuando se pase a analizar otro comando.

[2] Se entiende que quien inicia la comunicación es el PC, por eso se busca el primer out hacia el LA

Al hablar del término *estado inicial* se hace referencia al estado en el que se encontraba el analizador ese GO. Se puede hablar de tres estados:

- Estado 0: Es el estado al que se llega nada más iniciar el software de usuario.
- Estado 1: Se llega a él cuando o bien se va a muestrear a una frecuencia 'x' por primera vez o bien se ha cambiado algún parámetro del trigger o bien se ha cambiado la opción de memoria.
- Estado 2: Se permanece en el mientras se ejecuten capturas en las que no se ha cambiado ninguna opción de frecuencia, parámetro trigger o opción de memoria.

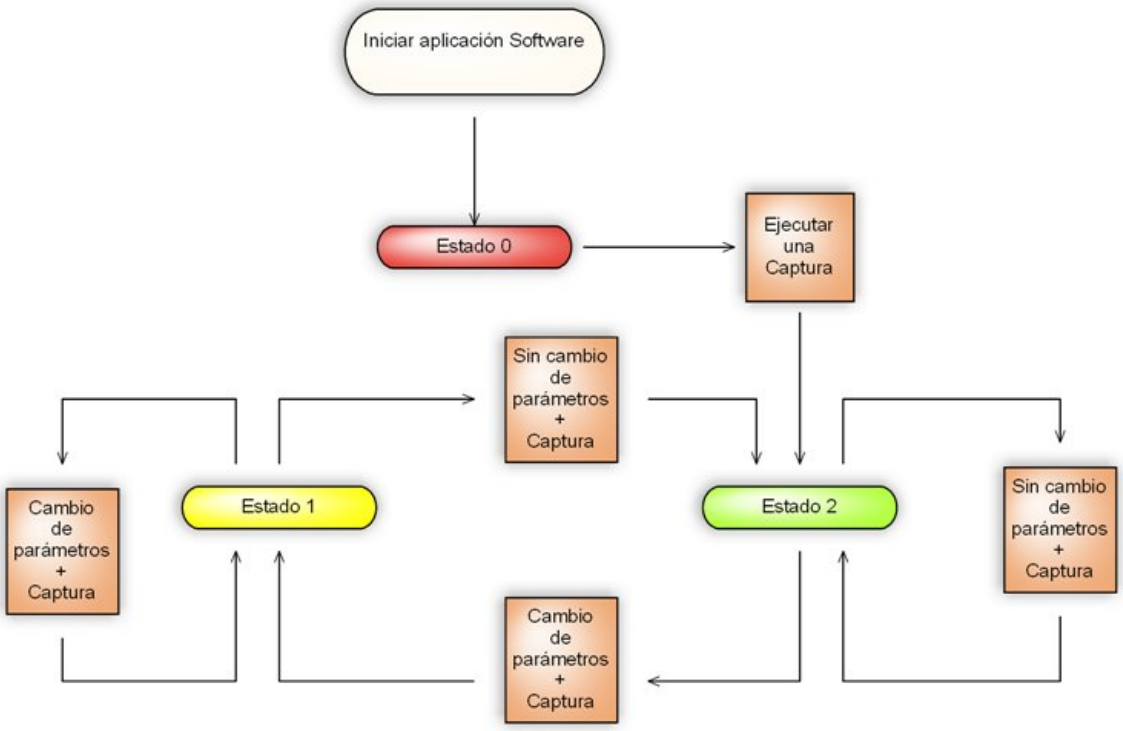


Figura 3.18: Estados del software MS-DOS para el LA-4540

Se habla de estos 3 estados porque dependiendo de cual se parta el código que manda el software es distinto. Esto se explicará mas detalladamente según se avance en la adquisición de nuevos comandos.

Una vez recordados los paso generales de obtención de ficheros y comentado el término estado inicial se pasa a explicar la obtención de los ficheros de funcionamiento del LA-4540,

su secuencia de funcionamiento y por último la secuencia de funcionamiento desarrollada por nosotros.

3.3.2. Obtención del primer archivo: " inicio de aplicación software "

En esta subsección se describe como se halla el primer archivo de comunicación entre PC y LA. Está detallado en pasos desde el punto de vista de cada ordenador. La obtención de otros comandos de comunicación entre PC y LA siguen la misma estructura.

Cuando se inicia la aplicación software del analizador suministrada por el fabricante se realizan dos operaciones:

1. Comprobar si el analizador lógico está conectado.
2. Hacer un reset del LA para dejarlo en un estado conocido, en este caso al estado 0.

Para conseguir este inicio de aplicación se debe actuar de la forma siguiente:

1. En PC1: sistema operativo MS-DOS.
2. En PC2: sistema operativo Windows, y con la aplicación software ejecutándose con condición de disparo en OUT.
3. En PC1: iniciamos la aplicación software, se escribe: 'main' y se pulsa 'intro'.

   En esto momentos se inicia la aplicación software del analizador del PC1.
4. En PC2: se obtienen varios archivos.log. (`000000.log`, `000001.log`, ...). Una vez obtenidos podemos pulsar STOP.
   - Se borran todos los .log excepto el primero de ellos, en este ejemplo el `000000.log`[3].
   - Se ejecuta la aplicación AN.exe obteniendo el fichero `listado.txt`, que se muestra a continuación.

MUESTRA	ADDR	DATA	IOR	IOW	S2_OEN	000000.la
10	[0x0E13],	0x01,	1,	0,	0	OUT
28	[0x0A13],	0x01,	1,	0,	0	OUT
42	[0x0613],	0x01,	1,	0,	0	OUT
56	[0x0213],	0x01,	1,	0,	0	OUT

[3]La aplicación AN.exe lee en orden todos los archivos .log que se encuentre en el mismo directorio que él. Y crea un fichero llamado listado.txt donde se escriben todos los out e in de los ficheros .log encontrados. Si sólo nos interesa el 000000.log no hay razón de procesar los demás y se opta por borrarlos.

```
75    [0x0213],    0x01,    1,    0,    0    OUT
.....
.....
829   [0x0212],    0x06,    1,    0,    0    OUT
840   [0x0211],    0x00,    0,    1,    0    IN
872   [0x0611],    0x00,    1,    0,    0    OUT
889   [0x0611],    0xFF,    0,    1,    0    IN
```

- A la vista de las últimas lineas se podría poner como condiciones de disparo:

  secuencia 0: $\overline{SEN}$=$\overline{IOW}$=0 y $\overline{IOR}$=1 esto se denominara Out genérico, porque no se determinan ni los datos ni la dirección.

  secuencia 1: $\overline{SEN}$=$\overline{IOW}$=0 y $\overline{IOR}$=1 + dir=0x611 + dato=0x00, es decir, se busca un out(0x611,0x00).

  secuencia 2: $\overline{SEN}$=$\overline{IOR}$=0 y $\overline{IOW}$=1 + dir=0x611 + dato=0xFF, se busca un in(0x611,0xFF).

5. En PC1: se vuelve al estado inicial. En este caso se sale de la aplicación pulsando Alt+F4 volviendo al símbolo del sistema

6. En PC2: se pulsa la opción GO.

   En PC1: se escribe main iniciándose la aplicación software.

7. Ahora el analizador se disparará cuando se cumplan las tres secuencias descritas en el apartado 4 consiguiéndose otra serie de archivos .log. Como ocurría antes sólo el primero de ellos es válido. Se pulsa STOP.

   - Lo siguiente es borrar todos los `.log` menos la primera captura y ejecutar la aplicación *AN.exe.*
   - Se obtiene el fichero `listado.txt`.
   - Las últimas lineas de este fichero serán las nuevas secuencias de disparo.
   - Se pasa al paso 5

Esta secuencia de pasos termina cuando ya no se consigan ficheros .log distintos.

Ahora que ya se han obtenido la secuencia de fichero `.log` que contiene la información de los out´s e in´s que hay que mandar al analizador sería deseable tenerla en forma de lenguaje de programación C. Ésto también lo hace An.exe; en la misma carpeta en la que estén esos

archivos `.log` válidos y ordenados secuencialmente por nombre (por ejemplo: 000001.log y 000004.log) se ejecuta *AN.exe*. De todos los archivos de salida que genera habrá que fijarse en el llamado "`log.c`". Este archivo está compuesto por una cabecera con las líneas necesarias para ser un archivo compilable, y por los dos tipos de instrucciones out e in. Si este archivo se lleva al PC1 y se compila bajo BC se inicializara el LA-4540.

Listing 3.1: inicioLA()

```
#include <stdio.h>
#include <dos.h>
void main()
{
        unsigned int addr;
        unsigned int data;
        unsigned char rdata;
        float i;

//      000000.la
        outportb(0x0E13,0x01);
        outportb(0x0A13,0x01);
        outportb(0x0613,0x01);
        outportb(0x0213,0x01);
        outportb(0x0213,0x01);
        outportb(0x0613,0x01);
        ...
}
```

3.3.3. Obtención de una captura completa

En esta subsección se va a explicar como hallar una secuencia de captura para los siguientes parámetros de entrada en PC1:

- frecuencia de muestreo: 20Khz
- memoria de captura: 8k
- trigger: sin ninguna condición
- modo: true.

- método: once.
- tensiones umbrales: 1.45v.

Otro punto muy importante es el *estado inicial* en el que nos encontramos:

- Como *estado inicial* estamos en el caso de que una vez iniciada la aplicación software del PC2 se han configurado los parámetros antes especificados y se ha hecho una captura. Por tanto, si no se cambia ninguno de los parámetros permaneceremos siempre en el estado 2, y después de cada captura completa no hará falta hacer nada.

Por último, como se ha recordado anteriormente el PC1 funciona bajo Windows y su LA muestrea a 10Mhz con una memoria de 128K.

Si se mantienen todos los parámetros especificados y el estado inicial del cual se parte es siempre el mismo se obtendrá el fichero que permite indicar al modelo LA-4540 que inicie una captura de muestreo y devuelva los datos que obtenga en los pods.

El procedimiento es similar al descrito en la subsección 3.3.2, los pasos a seguir son idénticos y lo que cambia es el trigger para cada archivo. Los pasos generales son:

- Captura de un archivo.
- Análisis de ese fichero con *AN.exe*.
- Obtención de listado.txt.
- Nuevas condiciones de trigger.
- Repetir esos pasos hasta que no haya ficheros nuevos.
- Generar el código en C de los .log válido.
- Compilarlo en el PC1 y probar que funciona.

La secuencia de pasos para obtener los ficheros de captura sería:

1. En PC1: debe de estar en el estado inicial antes descrito.
2. En PC2: activada la opción reset de archivos `.log`, con método *once* y condición de trigger un OUT genérico se pulsa GO.

3. En PC1: se activa la opción GO ON, es decir, se pide al PC1 que inicie una captura[4].Cuando o bien la pantalla del PC1 actualice los datos de entrada o bien el PC1 produzca un pitido con su altavoz interno tendremos una captura completa.

4. En PC2: Ahora tendremos varios archivos .log resultado del muestreo por parte del LA del PC2.

   - De todos los .log sólo será válido el primero de ellos. Se pueden borrar todos los demás. Se pulsa STOP.
   - A ese primer archivo de todos lo obtenidos se le aplica el programa *AN.exe* y se obtiene un secuencia de out's e in's.
   - Las últimas instrucciones serán condiciones de trigger en el PC2 para obtener el siguiente fichero.
   - Una vez introducidas como eventos de trigger del PC2 volvemos a pulsar GO.

5. Volvemos al paso 3. Cuando ya no tengamos mas archivos nuevos se dará por terminado el *comando captura*.

Con lo dicho hasta aquí se ha conseguido la secuencia de archivos `.log` para pedir una captura a 20Khz, a continuación se obtiene el código en C - ejecutando *AN.exe* sobre esos archivos -, por último se lleva ese archivo al PC1 y tras compilarlo se consigue hacer funcionar al analizador bajo un sistema operativo MS-DOS.

Los pasos aquí explicados serían también válidos para obtener el comando de captura para el resto de las frecuencias de muestreo, distinta capacidad de memoria, distinto trigger, distinto umbral...

Si se cambia de frecuencia y se realiza una captura, se mandaran los mismos out´s e in's que en el caso de 20khz, sin embargo, en alguna parte del código, la que indica la frecuencia de muestreo, habrá un sutil cambio[5].

Con lo explicado hasta ahora aún no se sabe la zona de código donde se indica la información de frecuencia de muestreo, y la única forma de averiguarlo sería cambiar la frecuencia de 20KHz a otra frecuencia y volver a obtener una captura completa.

A priori parece que uno se ve obligado repetir los 5 pasos anteriores con la pérdida de tiempo que eso implica. Para disminuir ese tiempo se puede usar la información obtenida en

[4]Como recordatorio hay que decir que el LA del PC1 esta muestreando señales conocidas provenientes del entrenador.

[5]La misma premisa se puede aplicar si manteniendo la frecuencia de 20Khz se cambia la capacidad de memoria, el LA recibirá la misma cantidad de out´s e in´s que en 20khz pero al comparan los archivos listado.txt para cada caso se encontrarán las lineas que indican la capacidad de memoria.

la captura de 20 KHz y así fijar las condiciones necesarias que permitan obtener la secuencia correcta de archivos `.log`.

Los eventos - condiciones de disparo - se describen en la tabla 3.2.

Numero de POD	POD 3				POD 2								POD 1							
Nº de evento	–	–	–	A11	A10	A3	A2	A1	A0	-	$\overline{IOR}$	$\overline{I0W}$	$\overline{SEN}$	D7	D6	D5	D4	D3	D2	D1	D0
0 (out(x,x))				x	x	x	x	x	x		1	0	0	x	x	x	x	x	x	x	x
1 (out(212,EF))				0	0	0	0	1	0		1	0	0	1	1	1	0	1	1	1	1
2 (in(211,x))				0	0	0	0	0	1		0	1	0	x	x	x	x	x	x	x	x
3 (out(212,0F))				0	0	0	0	1	0		1	0	0	0	0	0	0	1	1	1	1
4 (out(x,x))				0	0	x	x	x	x		1	0	0	x	x	x	x	x	x	x	x
5 (out(213,C7))				0	0	0	0	1	1		1	0	0	1	1	0	0	0	1	1	1
6 (out(210,B8))				0	0	0	0	0	0		1	0	0	1	0	1	1	1	0	0	0
7 (out(210,D8))				0	0	0	0	0	0		1	0	0	1	1	0	1	1	0	0	0
8 (out(213,C7))				0	0	0	0	1	1		1	0	0	1	1	0	0	0	1	1	1
9 (out(210,E8))				0	0	0	0	0	0		1	0	0	1	1	1	0	1	0	0	0
10 (out(210,F0))				0	0	0	0	0	0		1	0	0	1	1	1	1	0	0	0	0
11 (out(210,F8))				0	0	0	0	0	0		1	0	0	1	1	1	1	1	0	0	0

Cuadro 3.2: Eventos para conseguir una captura completa

Una vez definidos los eventos hay que especificar la secuencia correcta para ir obteniendo los ficheros .log válidos, para ello se indica una posible *secuencias de eventos* para ir obteniendo esos archivos .log:

1 archivo: Es el inicio de petición de captura. Se denominará ***go***.

Secuencia trigger:	evento 0

2 archivo: Es el primer archivo después del denominado go, se denominara ***continua1.***

Secuencia trigger:	evento 0
	evento 1
	evento 0

3 archivo: Es el segundo archivo después del denominado go, se denominará ***continua2***. Aquí va incluido la petición de ***datos del pod 1***.

Secuencia trigger:	evento 0
	evento 1
	evento 2
	evento 4

4 archivo: Es la petición de ***datos del pod 2.***

Secuencia trigger:	evento 0
	evento 1
	evento 0
	evento 2
	evento 4
	evento 5
	evento 6

5 archivo: Es la petición de ***datos del pod 3.***

Secuencia trigger:
evento 0
evento 1
evento 0
evento 2
evento 4
evento 5
evento 7

6 archivo: Es la petición de ***datos del pod 4.***

Secuencia trigger:
evento 0
evento 1
evento 0
evento 2
evento 4
evento 5
evento 7
evento 8
evento 9

7 archivo: Es la petición de ***datos del pod 5.***

Secuencia trigger:
evento 0
evento 1
evento 0
evento 2
evento 4
evento 5
evento 7
evento 8
evento 10

8 archivo: Es el ***último archivo*** de petición del comando captura. Se trata de una ultima configuración.

Secuencia trigger:	evento 0
	evento 1
	evento 0
	evento 2
	evento 4
	evento 5
	evento 7
	evento 8
	evento 10

Observación:

- sólo es válido el primer archivo de cada captura.

En el diagrama de bloques de la figura se puede ver los archivos obtenidos a excepción del denominado espera, que se explicará mas adelante.

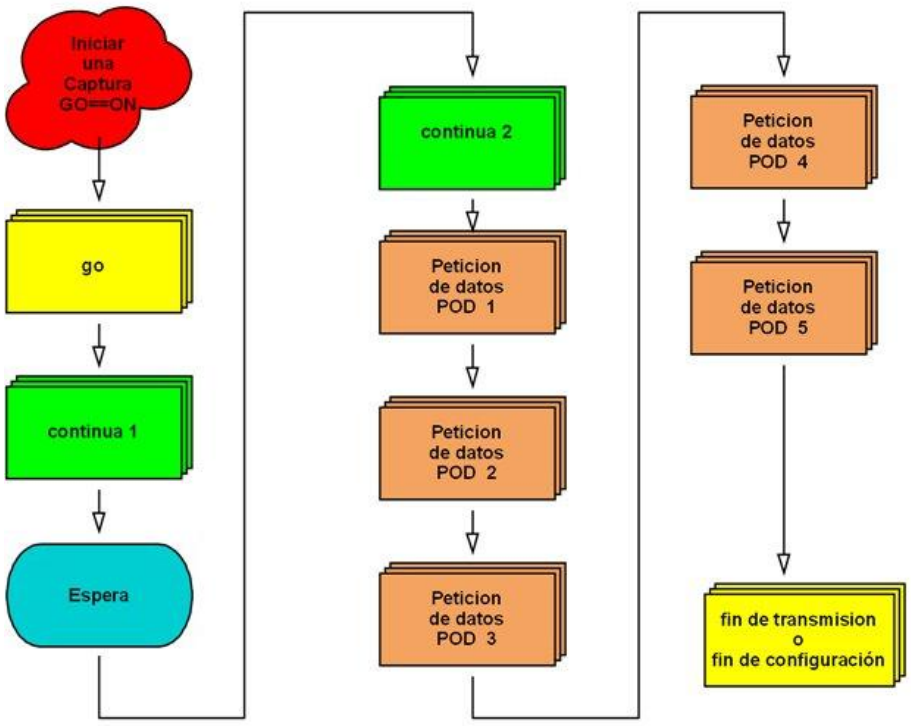


Figura 3.19: Archivos para el comando *captura*

3.3.4. Muestreo a distintas frecuencias con el LA-4540

Una vez que se ha conseguido hacer funcionar al LA a la frecuencia de captura 20KHz se pasó a buscar los cambios necesarios para que funcionase a cualquier frecuencia. Para ello era necesario repetir el proceso de captura de archivos `.log` a cada frecuencia, creando un archivo completo de captura para cada frecuencia; para después ir comparándolos dos a dos y saber que líneas son las que llevan la información de frecuencia.

A priori esta tarea sería lenta - hay que repetirla 20 veces, una para cada frecuencia - pero nos daría la clave para saber como cambiar de frecuencia.

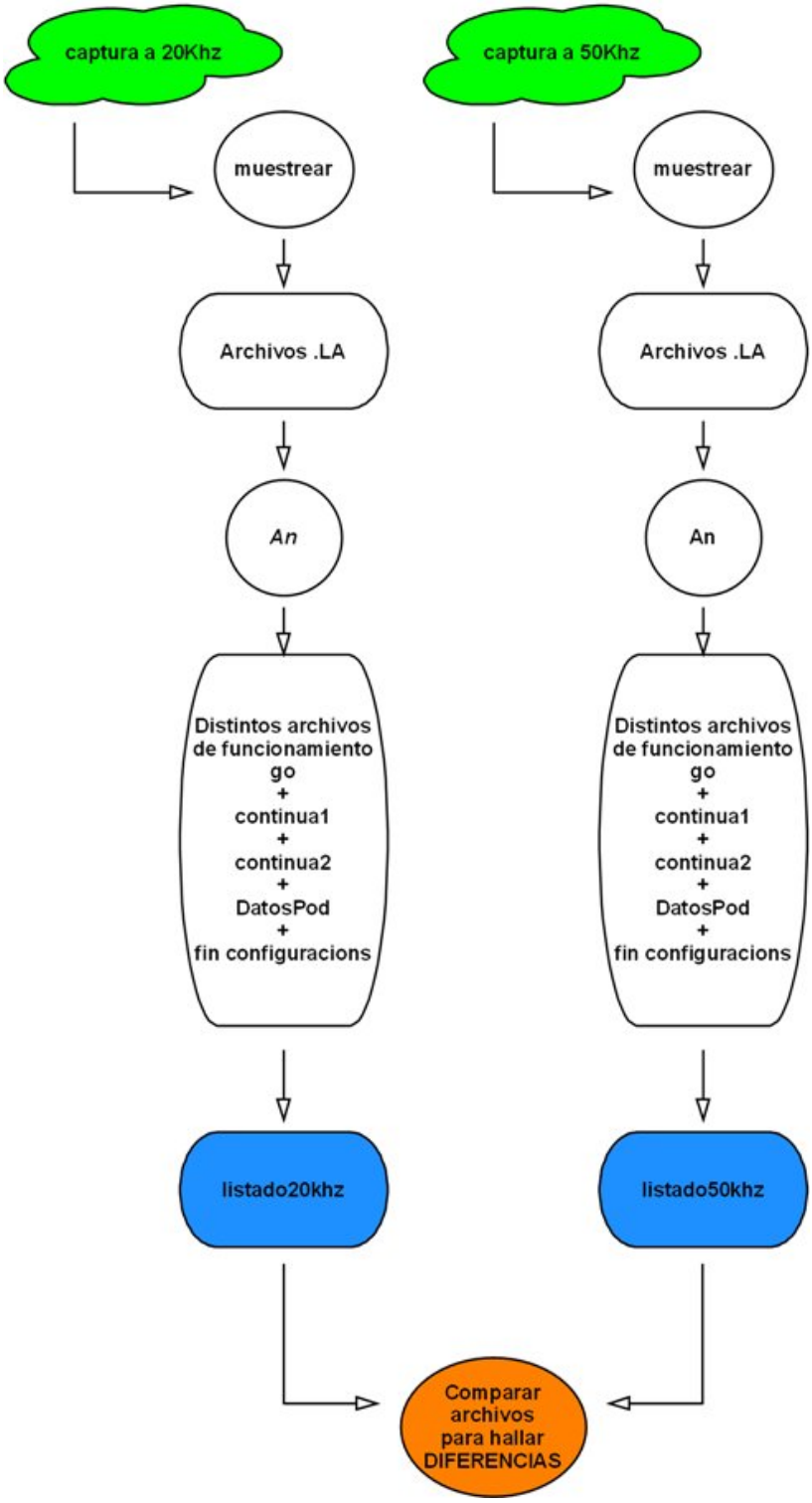


Figura 3.20: Comparación de archivos

Tras realizar varias capturas de archivos `.log` de distintas frecuencias se obtuvieron las siguientes conclusiones:

- La zona de código para indicar la frecuencia de muestreo se indica en el archivo continua 1 y en fin de configuración.

- Existen 4 formas de indicar esa información, dependiendo del grupo de frecuencia que se indique. Estos grupos de frecuencia se definen como:

  - Grupo C: frecuencias $\subseteq [1Hz, 100Hz]$.
  - Grupo A: frecuencias$\subseteq [200Hz, 5MHz]$.
  - Grupo B: frecuencias$\subseteq [10MHz, 100MHz]$.
  - Grupo 250Mhz: frecuencia de 250 MHz.
  - Grupo 500Mhz: frecuencia de 500 MHz.

- Aunque existen instrucciones donde se dice explícitamente la frecuencia, en todo el código se pueden ver sutiles diferencias debidas al parámetro frecuencia.

La tarea de captura de archivos `.log` para conseguir los listados a cada frecuencia llevó varias jornadas de trabajo y, como no siempre se partía del mismo estado inicial, se obtuvieron **distintos** listados para una misma frecuencia. Además, el ir obteniendo tantos ficheros de funcionamiento nos dio una idea general de como funcionaba el LA.

Observando el diagrama de estado del LA, figura 3.18, se puede ver que una vez realizada una captura nunca más se vuelve al estado 0, entonces en este estado no se realizaron comandos de captura.

Ahora se han disminuido los listados, solo hay dos tipos para cada frecuencia. Se puede distinguir entre vez=1 y vez=2, entendiendo que cuando se cambia a una nueva frecuencia y se realiza una captura nos encontramos en el estado 1(vez=1) y si se vuelve a realizar una captura será desde el estado 2 (vez=2), y se continua en este estado mientras se realicen capturas sin cambiar de frecuencia u otro parámetro.

En el diagrama de bloques de la figura 3.3.4se muestran los archivos necesarios para el funcionamiento del LA-4540 para todos los grupos excepto para el grupo C.

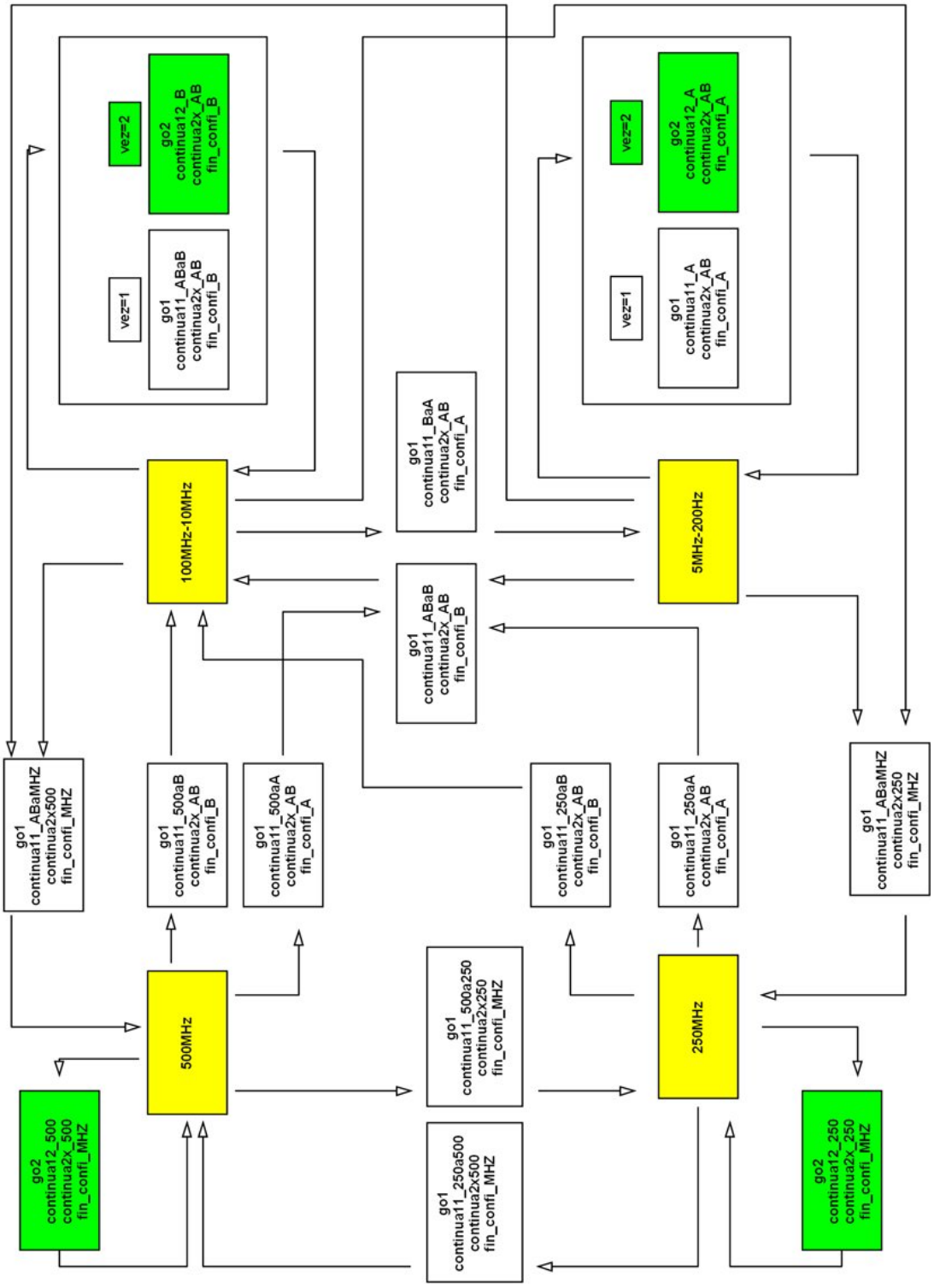


Figura 3.21: Ficheros de funcionamiento para el LA-4540

En este diagrama se pueden ver los distintos *grupos de funcionamiento* y los *distintos archivos* necesarios para hacer funcionar al LA-4540, y su variedad depende:

- Del estado del que se parte: por eso se enumeran continua11 - parte del estado 1 - o continua12 - parte del estado 2 - , y si el continua11 y continua12 coincidiesen se llama continua1x.
- Del grupo de frecuencia desde donde se parte y el grupo de frecuencia donde se llega. El archivo continua11_250aA indica que es el primer archivo después de un "Go" y por eso se llama continua1. Indica que se parte del estado 1, y que el analizador hizo una captura a 250Mhz y ahora va a realizar otra del grupo A.

Mientras se obtenían y configuraban todos los archivos mencionados en la figura 3.3.4 se empezó a programar la aplicación de usuario. Cuando esta aplicación estuvo lo suficientemente preparada para cambiar de frecuencia y dibujar por pantalla los datos se hicieron las primeras pruebas para hacer funcionar al LA en modo ISA desde el software propio.

En estas primeras pruebas se encontró un error: si se cambiaba de frecuencia el analizador no realizaba la captura la primera vez que se le ordenaba sino la segunda vez, faltaba algo.

Faltaba el tiempo necesario para que el propio hardware del LA-4540 llenase la memoria dedicada a las muestras. Por ejemplo, si se manda muestrear a 20KHz y la memoria es de 8K se necesita un tiempo mínimo de espera entre que se manda el archivo continua1 y se empieza a pedir la información de los pods.

$$tiempo\,de\,captura = \frac{1}{frecuencia} \bullet memoria = 50us \bullet 8 \bullet 1024 = 0,4096\,segundos$$

Si se piden los datos antes de tenerlos el analizador o bien no manda nada o bien manda los datos de la captura anterior. La obtención del archivo de espera de tiempo se explica en 3.3.5 y su obtención facilitó mucho la aplicación software ya que tras varias pruebas se probó que aunque el software `main.exe` mandaba el mismo comando de tres formas diferentes se le podía hacer funcionar mandando únicamente los archivos que partían del estado2.

Esto facilitaba enormemente la labor de obtener los archivos pues ahora sólo interesaban los denominados go2, continua12 y continua22 en cada grupo de frecuencias.

En estos archivos hay una gran dependencia de la frecuencia y tienen un número elevado de diferencias entre ellos. Sin embargo, mediante el uso de programas que permiten comparar archivos como son Total Comander y Csdiff se consiguió pasar de tener cuatro tipos de continua12 a uno solo, y lo mismo ocurre con el continua22 y el finconfi. De esta forma el funcionamiento de la aplicación era mas sencillo.

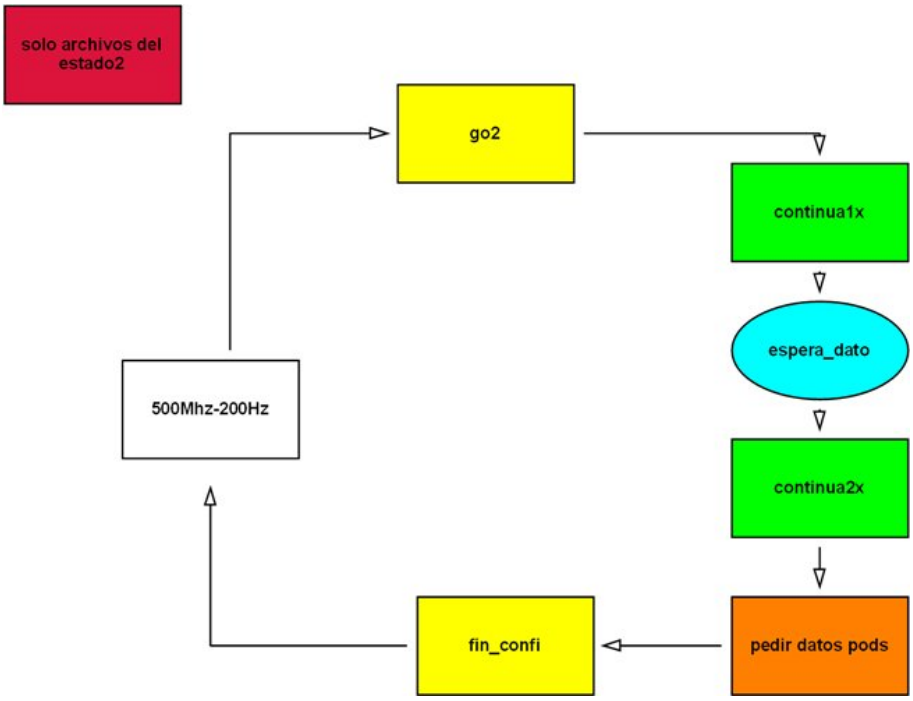


Figura 3.22: Archivos para el funcionamiento del LA-4540

3.3.5. Obtención de las esperas

Se denomina **espera** el tiempo que transcurre *desde* que se ha configurado correctamente al analizador *hasta* que el analizador tiene los datos muestreados y preparados para ser transmitidos hacia el PC.

En el modo de analizador lógico existen dos tipos de esperas:

- Espera sin trigger
- Espera con trigger

Durante la espera el software realiza in's constantemente hasta que recibe el valor esperado. Los in's se hacen a la dirección 0x211- inport(0x211) - y el LA puede responder con alguno de los 2^8= 255 valores posibles. Aquí radica el problema de cómo saber qué valores determinan que la espera ha terminado.

Soluciones

1. Espera sin trigger: *solución esperar un tiempo fijo.*

Si no hay condición de disparo el tiempo de espera es siempre fijo. Conocida la frecuencia de muestreo y la capacidad de memoria la expresión que nos da el tiempo a esperar es:

$$tiempo\,de\,captura = \frac{1}{frecuencia} \bullet memoria$$

Entonces después de mandar el archivo continua1 se realiza un tiempo de espera igual a $\frac{1}{frecuencia} \bullet memoria$ segundos y después se procede a pedir los datos.

El problema de esta solución es que estamos obligados a esperar un tiempo fijo y no se puede abandonar la espera (pulsando una tecla, por ejemplo). Si la frecuencia es de f=200Hz y la memoria de 128K tendríamos que esperar aproximadamente 11 minutos y no habría ninguna posibilidad de poder salir del programa de forma correcta hasta esperar ese tiempo.

Con ese problema esta solución queda para frecuencias muy altas tales como 500Mhz, 250 MHz ó 100Mhz donde el tiempo de espera es inapreciable para un hipotético usuario de la aplicación.

2. Espera sin trigger: *solución consultar el estado del analizador.*

   En esta solución se ha optado por hacer lo mismo que hace el software del fabricante, es decir, preguntar al analizador.

   El problema está en que aún no sabemos que IN indica que la información está preparada. Y además puede ocurrir que cada *grupo de frecuencias* indique que la captura ha finalizado mediante un IN distinto.

   Para poder ver el IN que interesaba se podían seguir dos métodos

   a) método 1:

      La premisa para este método es que al realizar una captura con condición de disparo el analizador proporciona alguna muestra anterior a la condición de disparo.

      Entonces, como sabemos que justo después del IN correcto va el archivo continua2. Y este archivo se conoce como empieza, se podría poner en el PC2 como condición de disparo las primeras lineas del archivo continua2, y con un poco de suerte podríamos ver no sólo la condición de disparo sino algunas muestras anteriores.

      Este método no dio resultados ya que el analizador después de dar el IN correcto hay un tiempo durante el cual no hay comunicación entre el PC-AL, es decir, que el software pregunta y se toma un tiempo antes de hacer otra consulta. Y durante

ese tiempo el analizador sigue muestreando por lo que llena sus 128K sin ninguna información relevante.

Sin embargo, prestando atención a la última linea del archivo continua1 se observaba que este archivo siempre terminaba con un inport(0x211), y no siempre el valor era el mismo. Sino que había dos variantes: 0x30 y 0x10. Por tanto, ya se conocían dos valores de IN que indicaban que el analizador no estaba aún listo.

b) método 2:

Esta vez la premisa es que el software continua ejecutando la instrucción inport(0x211) mientras no reciba algo distinto de 0x30 ó 0x10.

Entonces sabiendo que mientras recibiese 0x30 ó 0x10 no se abandonaba la espera, sólo faltaba saber qué valor hacía que se saliese de ella. Para conocer ese o esos valores se fueron configurando distintas condiciones de disparo en el PC2 para que se pudiera reconocer el IN que permitía salir de la espera.

La condición de disparo debía contener las ultimas lineas del archivo continua1 + un IN + las primeras lineas del archivo continua2. De esta forma se aseguraba que se buscaba el IN de la espera. Y aunque fuese un poco lento el método era exacto y excluyente. Consistía en configurar el disparo con los distintos posibles valores que podía recibir un in, es decir, desde 0x00 hasta 0xFF.

Tras buscar el valor del IN de salida se comprobó que su valor podía ser 0x70 ó 0x50.

Para poder comprobar que se había realizado correctamente el código para realizar la espera se creo un programa en C. La idea era comprobar las esperas al hacer el inport(0x211), viendo que era correcto lo visto al muestrear el bus ISA, es decir, que la espera terminaba con un 0x50 ó un 0x70.

```
inicioLA();
got();
continua11t(salida1,salida2);
do{
   outportb(0x0215,0xC2);
   outportb(0x0212,0xFD);
   rdata=inportb(0x0211);
   if(rdata==0x00) cont00++;
   if(rdata==0x20) cont20++;
   if(rdata==0x50) cont50++;
```

```
    if(rdata==0x70) cont70++;
    printf("\nEl ultimo in fue: 0x%04x",rdata);
    var++;
  }while(rdata!=0x30);
  printf("\nSe han escrito en modo trigrer:\n\t0x00 %d\n\t0x20 %d\t
  Total esperas %d\n\t0x50 %d\n\t0x70 %d"
  ,cont00,cont20,cont00+cont20,cont50,cont70);
  printf("\nEl ultimo in fue: 0x%04x",rdata);
```

3. Espera con Trigger: *solución consultar el estado del analizador.*

   Ahora no sabemos nunca el tiempo de espera, pues hasta que no se cumpla la condición de disparo de trigger el tiempo es aleatorio. Ahora solo hay una opción: preguntar continuamente al analizador el IN que indica que ya se tienen los datos y que podemos pedírselos.

   La forma de proceder fue similar a la descrita en el apartado de *espera sin trigger*. En primer lugar se usó la opción de configurar una condición de disparo con las primeras lineas del archivo continua2, pero como ocurría en la opción sin trigger no se conseguía obtener ninguna muestra anterior a la primera instrucción del archivo continua2.

   En segundo lugar se opto por poner como condición de disparo cualquier IN genérico después del archivo continua1. Gracias a esta estrategia se comprobó la estructura de la espera del analizador; ésta constaba de 3 instrucciones:

   - outport(0x0215,0xC2)
   - outport(0x212,dato)*
   - inport(0x211)*
   - después si no es el IN adecuado pasa un periodo de tiempo sin comunicación PC-LA y vuelve a preguntar.

   *El dato transmitido y el dato esperado dependía del grupo de frecuencias donde se estuviese trabajando.

Cuadro 3.3: Dato a enviar y a recibir en una espera con trigger

Dato/grupo	grupo 500	grupo 250	grupo B	grupo A
Dato outport	0xBD	0xBD	0xFD	0xFD
Dato inport	0x50	0xE0	0x70	0x70

Aunque en la tabla 3.3 ya se indica cual es el dato IN que indica el final de la espera aún no se explicado como se llego a su obtención. La metodología es la misma que en el caso anterior: saber cuál de los 255 posibles valores que salen con 8 bits indican que se ha terminado la espera.

La condición de trigger incluiría las últimas instrucciones del archivo continua1 + los dos out de la secuencia de espera+IN variable y de esta forma se conseguiría obtener el archivo continua2 si el IN seleccionado es el correcto.

Como ocurría en el caso de espera sin trigger lo exacto es recorrer todos los posibles valores; pero analizando los archivos en los que se consiguió ver la estructura de salida (out, out, in) si se observa qué valores indican que no ha terminado la espera son todos de la forma 0x30, 0x20, 0x10... , es decir, que el analizador solo cambia los bits D4 a D7. Por tanto solo hay 8 posibilidades. De todas formas se comprobaron todas las posibilidades por si acaso podía haber algún fallo.

Una vez que ya se conocían estos valores de salida se creo un programa en C con los ficheros provenientes de una captura con trigger (go, continua1, continua2, petición de pods y fin_confi) y se probó que el código de la espera era correcto.

Una vez creado este código la forma de proceder era la siguiente:

Iniciar el `main.exe` y escribir una condición de disparo. Cada vez que se pulsa un tecla perteneciente a un evento el software transmite toda la secuencia al LA[6]. Éste código no se había incluido en el archivo anterior.

Se realiza una captura con trigger y se sale del `main.exe`.

Cuando uno cierra la aplicación `main.exe` no se le indica nada al hardware. Por tanto si ahora se ejecuta el código antes mencionado habrá una ejecución correcta.

Por ultimo, lo que en verdad era lo mas importante, una vez que el analizador tiene en su memoria la condición de disparo se puede ir modificando el programa en C y así comprobar si funcionan correctamente las instrucciones de espera.

3.3.6. Cambio de parámetros de muestreo

Hasta ahora todo el proceso de ingeniería inversa ha estado centrado en hallar el código para realizar una captura, ver que instrucciones hay que modificar para cambiar de frecuencia,

[6]Desde nuestro punto de vista esto no es muy óptimo, podría mandárselo solo al final cuando sepa toda la información, pero como estábamos siguiendo la metodología de este software la aplicación software creada por nosotros también lo hace así.

y analizar cómo saber esperar a que se llene la memoria de captura para poder pedir los datos. Sin embargo el LA-4540 permite cambiar varios parámetros estos son:

- Frecuencia de muestreo.
- Capacidad de memoria de muestreo.
- Tensión umbral del pod1 y pod2.
- Tensión umbral del pod3 y pod4.
- Tensión umbral del pod5.
- Condiciones de disparo - trigger-
  - Eventos - hasta 16 eventos distintos -.
  - Secuencia de eventos - secuencia máxima de 16 eventos -
  - Modos de funcionamiento del trigger:
    - muestrear cuando se repita la secuencia n veces.
    - muestrear cuando se repita n veces o más.
    - muestrear mientras no se halla repetido n veces.
  - Condición falsa o verdadera.

Además de poder cambiar de modo de funcionamiento como generador de patrones y generador de patrones + analizador.

Una vez que sabemos los parámetros es necesario saber en que zonas de código transmite esa información: la respuesta es en múltiples sitios.

1. Cuando se cambia de una frecuencia 'x' a otra 'x±1' el software `main.exe` o cada vez que se toca algún valor de los eventos del trigger el software se encarga de transmitir información al LA-4540 como:
   - La condición false o true de la ultima captura realizada.
   - Condiciones de disparo actuales:
     - modo de funcionamiento.
     - los eventos ordenados secuencialmente.

2. Los archivos que componen una captura completa informan de todos los parámetros actuales:

   a) continua1:

   - frecuencia
   - si hay trigger o no
   - modo de funcionamiento: false o true
   - número de repeticiones del trigger
   - modo de funcionamiento del trigger: = =,≥ ó≤
   - capacidad de la memoria de captura 8K o 128K
   - tensiones umbrales de cada pod

   b) continua2:

   - información que depende del grupo de frecuencia

   c) espera:

   - existen 2 tipos de espera, con trigger o sin trigger, en ambos el software realiza in's hasta recibir el valor esperado para después pedir los pods

   d) peticiones de pods

   - si hay trigger o no
   - modo de funcionamiento del trigger : = =,≥ ó≤
   - modo de funcionamiento: false o true

   e) fin_confi

   - manda la misma información que el archivo continua1 pero realiza máscaras diferentes

Por ultimo falta comentar la metodología para ir obteniendo esto archivos.

3.3.7. Método de comparación entre archivos

Hasta ahora se ha comentado que se para ir sacando el funcionamiento del analizador a distintas frecuencias se tomaban los listados de capturas a distintas frecuencias, comparando listados dos a dos para ir anotando las diferencias. Pero aún no se ha dicho qué programas se han usado para poder realizarlas. Estos programas han sido:

- Total Comander : opción comparar por contenido
- Csdiff

La mayoría de las veces se optó por usar el programa Total Comander ya que la versión 7.02a permite editar los archivos cuando se están comparando. En ocasiones este programa no daba buenos resultados si había permutaciones de código, en cuyo caso se usaba el programa Csdiff.

Anteriormente en la figura 3.3.4 se ha indicado que en el proceso de comparación se analizaba toda una captura completa. Se puede hacer así, sin embargo resulta más ventajoso ir comparado uno a uno por separado los archivos en los que se divide una captura.

Cuando se busca la *zona de código* donde se indica la frecuencia o la capacidad de memoria, no se puede saber a priori, y solo después de ir comparando varios archivos se puede conocer la *posición de instrucción* de un archivo que contiene esa información.

Para saber la posición que ocupa un parámetro en concreto en un archivo "x" se puede proceder de las siguiente forma:

- Resulta conveniente que cuando el programa *AN* devuelve el `listado.txt` de un archivo no incluya el numero de muestra. La razón es la siguiente: aunque dos archivos que correspondan a distintas frecuencias tengan siempre las mismas instrucciones, al muestrearlo es muy posible que no ocurran en la misma muestra. Por tanto, el programa Total Comander va a indicar que existe una diferencia cuando no la hay.
- Ya que el programa Total Comander da *información de la posición de una linea dentro de un archivo,* se anotan que posiciones son las que tienen información distinta.

  Dado que el archivo `listado.txt` lo genera *AN.exe* y que si un listado contiene 100 lineas, la primera siempre es la cabecera y el resto corresponden a los 99 fprintf que *AN.exe* se ha visto obligado a realizar. Se puede hacer una correspondencia entre la posiciones diferentes que se observan con el Total Comander y el numero de veces que *AN.exe* ejecuta un fprintf para ir escribiendo el `listado.txt`.

  Si analizados dos o más archivos se conocen todas las posiciones de las instrucciones que configuran un parámetro se le indicar al programa *AN.exe* que vaya contando las veces que realiza un fprintf y que cuando vaya a escribir la información de una linea de interés lo haga en otro fichero.

Llegados aquí se se pueden dar dos casos:

1. Puede ocurrir, como ha sido el caso de las frecuencias al analizar el grupo denominado B o la capacidad de memoria, que cuando se van comparando los archivos `listado.txt` dos

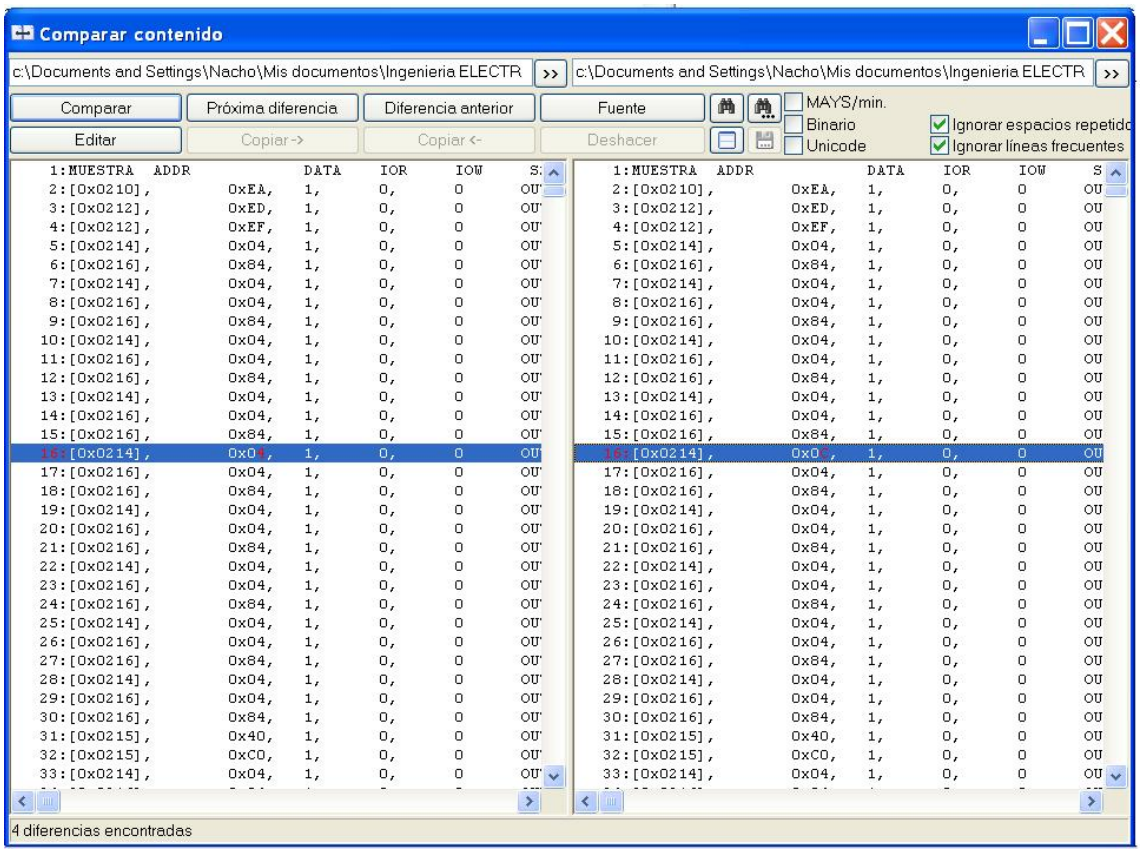


Figura 3.23: Comparación de dos listados.txt con Total Comander

a dos, por ejemplo sólo de continua1, además de anotar la posición de la instrucción se va anotando el contenido y sin darse uno cuenta se obtiene toda la información necesaria escrita en papel.

El siguiente paso sería generar el código en C de ese tipo de archivo, siguiendo con el ejemplo anterior continua1, y al generarlo que tuviese el mismo formato que los archivos `listado.txt`, es decir, una línea de cabecera y el resto correspondiente a las instrucciones inport u outport. El programa DevC++ incluye la posición de la instrucción dentro de un archivo.

Por último, en ese archivo `.c` habría que ir a las lineas que se anotaron anteriormente, crear una variable para que el valor que tiene que ir sacando pueda ser distinto dependiendo de un factor exterior, como puede ser la frecuencia. Y así se tendría que con el mismo código de continua1 se puede indicar más de una frecuencia.

2. Por otro lado puede ocurrir que la información este siempre en una determinada posición

de código, como es el caso de la frecuencia en el grupo A o la información de las tensiones umbrales de los pods, y con solo comparar dos archivos se sepa la posición de las instrucciones que contiene esa información.

Poniendo como ejemplo la frecuencia del grupo A, tras comparar el listado continua1 a 20khz y 10khz se ve que la información siempre esta en la posición de líneas 724 y 725.

Entonces una opción seria ir sacando el listado para 50KHz y anotar el valor de las líneas 724 y 725. Después se obtendría el listado de 100KHz, y así consecutivamente. Pero aún mejor es indicarle al ejecutable *AN.exe* que cuando vaya a realizar el fprintf 724 y 725 escriba también esa información en otro `archivo.txt`.

Una vez que se le ha indicado a *AN.exe* que escriba esa información en otro archivo hay que suministrarle todos los continua1 correspondiente a las frecuencias del grupo A. Y ahora con sólo ejecutar una sola vez el programa *AN.exe* se consigue saber la información que llevan cada frecuencia en esas líneas.

Está claro que el segundo método es más sutil e ingenioso que ir comparando uno a uno los archivos `.txt` y anotarlo en un papel. El problema radica que saber en qué posición de un archivo está tal información es muy complicado. Ya que aunque todos los archivos siempre tienen la misma secuencia de direcciones no ocurre lo mismo con los datos porque varían mucho dependiendo del grupo de frecuencias del que se trate.

En ambos métodos la finalidad es la misma obtener la posición línea de la instrucción y el dato que varía al ir cambiando los parámetros del analizador.

3.3.8. Descripción de los archivos de funcionamiento del LA-4540

Una vez se sabe como envía el programa `main.exe` cada uno de los comandos con los diferentes parámetros de configuración se puede ir creando un programa propio de funcionamiento del LA-4540.

A continuación se hace una descripción de los procedimientos y funciones necesarias para la aplicación software propia. La descripción no incluye los ni los *in* ni tampoco los *out* pero si una descripción de su funcionamiento, descripción de parámetros E/S y de las distintas variables propias.

En la nomenclatura se pueden dar varios casos.

- Archivos con nombres sin ningún número ni letra de grupo.

  Son archivos válidos para cualquier frecuencia de muestreo y también para cualquier estado del que se parta.

- Archivos en cuyo nombre existe un numero.
  - nombre'1': indica que ese archivo es propio del estado1: se ha cambiado algún parámetro y al hacer un Go se parte desde el estado1 y por tanto los datos de los outport y los inport son distintos de si se parte de otros estados.
  - nombre'2': indica que ese archivo es propio del estado2.
  - nombre'x': indica que es un archivo genérico para todas las frecuencias. Son del estado2.

Como se ha explicado al final de las sección 3.3.4 en un principio se pensó que eran necesarios los archivos tanto del estado1 como los del estado2 - los del estado0 se quitaron ya que se podía inicializar al analizador y después realizar una captura partiendo del estado0 sin que el usuario se enterase. De de esta forma se continuaba haciendo un diagrama de funcionamiento idéntico a la aplicación `main.exe` - sin embargo, cuando se obtuvo los archivos de espera se comprobó que sólo hacían falta los archivos del estado2. En consecuencia en la aplicación software sólo se usan los archivos del estado2, y en la nomenclatura de los archivos ocurre que algunos archivos continúan con ese sufijo e indican de que estado vienen, aunque se repite *todos son del estado2.*

La información de los parámetros de salida del analizador se encuentran en variables externas a estos procedimientos y funciones obtenidos del análisis de archivos del `main.exe.` Las variables externas son:

- frec: indica la frecuencia de muestreo a la que se quiere hacer la captura actual.
- frec_Ant: indica la frecuencia de muestreo de la última captura que se hizo.
- memoria: indica la capacidad de memoria de los pods, 8K ó 128K.
- método: hay tres posibilidades de funcionamiento del analizador:
  - sin trigger.
  - con trigger:
    - $==$: igual al numero de repeticiones indicado por la variable nveces.
    - $>=$
    - $<=$
- modo: hay tres modos de funcionamiento, modo true o modo false. Hacen referencia al evento de condición de disparo.

- PUERTO: indica tres posibilidades: ISA, USB o PARALELO.

Los procedimientos y funciones para realizar las capturas en modo analizador son:

1. **unsigned char inicioLA()**

   Esta función es una secuencia fija de outportb e inportb cuya finalidad es la de inicializar el analizador a un estado conocido.

   La función que devuelve un valor en hexadecimal que informa de si el analizador esta conectado o no.

2. **void espera_dato()**

   Procedimiento que se usa para esperar el tiempo necesario para que el LA llene los 128K ó 8K de cada pod.

   Hay que dar la posibilidad de no tener que esperar todo el tiempo, si la frecuencia de muestreo es muy baja o queremos cambiar de opción de muestreo antes de que los pods se llenen; esa posibilidad estaría refleja como la acción de pulsar una determinada tecla para permitir salir de la espera.

   Listing 3.2: listado del procedimento de espera

```
void espera_dato()
{

    if(metodo!=0)
    {
        do{
                Espera2(int(tiempo/10));
               _outport(0x0215,0xC2);
               _outport(0x0212,var[indice_frec]);
               if(PUERTO==2)      rdata=USBin80();
               else               rdata=_inport(0x211);
        }while(rdata!=var_a_esperar_ST[indice_frec]);

    }
    else
    {
```

```
        if(frec<2)      Espera2(int(tiempo));
        else
        {   //Espera2(int(tiempo*1.5));
            //rdata=USBin80();

            do{

                Espera2(int(tiempo/10));
                if(PUERTO==2)   rdata=USBin80();
                else            rdata=_inport(0x211);

            }while((rdata==0x10)||(rdata==0x30));

        }
    }
}
```

Hay dos tipos de espera:

1)sin trigger: existen dos opciones a usar.

La primera se basa en que sabemos el tiempo de espera de manera fija:

$tiempo = capacidad\,de\,memoria \cdot \frac{1}{frecuencia\,de\,muestreo}$

La segunda se basa en la ingeniería inversa. Se sabe que después del archivo continua1x se hacen IN's; estos IN's seguidos se repiten hasta que el dato suministrado por el analizador al hacer el IN es el deseado.

PROBLEMA:

La primera opción no permite salir de la espera, al tocar un tecla por tanto se opta por la segunda opción para frecuencias bajas y por la primera opción para frecuencias altas tales como 500MHz, 250MHz o 100MHz.

2)Con trigger:

Ahora no se sabe nunca el tiempo de espera, pues hasta que se cumpla la condición de disparo de trigger el tiempo es aleatorio.

Sólo existe la opción de estar preguntando constantemente al analizador el IN que se debe recibir cuando se ha cumplido el trigger y tiene los pods listos para ser pedidos.

PROBLEMA:

El hacer consecutivamente inport hacia el analizador ocasiona que en el modo de funcionamiento USB los tiempos de acceso - comunicación entre PC, interface USB y analizador - se hagan excesivamente largos, lo que ocasiona un rendimiento de la aplicación software deficiente. La solución adoptada es no preguntar - realizar un inport- continuamente sino a intervalos de tiempo variables; de esta forma el tiempo de acceso en modo USB es similar al obtenido cuando se usa ISA. Esta solución se adopta tanto si hay trigger como si no lo hay.

Este procedimiento tiene una dependencia de la frecuencia en la que se trabaje como ya se dispuso en la subsección 3.3.5. Como solución a esos distintos valores se han creado dos variables internas, en este caso vectores, que dependiendo del grupo al que pertenece esa frecuencia cambian de valor. Las variables son tipo unsigned char var[]={0xBD,0xBD,0xFD,0xFD} y unsigned char var_a_esperar_ST[]={0x50,0xE0,0x70,0x70} y dependen de la variable externa índice_frec.

3. **go2**

   Este procedimiento es el comienzo de una transmisión de petición de captura.

   Distingue entre varios casos según el valor de la variable global frec

   A) que la frecuencia sea f=500Mhz

   B) que la frecuencia sea f=250Mhz

   C) que la frecuencia sea menor de f=250Mhz

Listing 3.3: listado go2()

```
void go2()
{
      if(frec==0)
         {
                  _outport(0x0210,0xF2);
                  _outport(0x0212,0xAD);
                  _outport(0x0212,0xAF);
         }
      if(frec==1)
         {
                  _outport(0x0210,0xEA);
                  _outport(0x0212,0xAD);
                  _outport(0x0212,0xAF);
```

```
        }
    if(frec>1)
        {
                _outport(0x0210,0xEA);
                _outport(0x0212,0xED);
                _outport(0x0212,0xEF);
        }
}
```

4. **void rec_pod1()**

   Procedimiento que permite recoger los datos del pod1.

   Para pedir los pods hay que tener en cuenta parámetros como tipo de conexión, la cantidad de memoria y el tipo de condición de trigger si la hubiese.

   Listing 3.4: procedimiento recpod1

```
void rec_pod1(){
    /*variables*/
    unsigned long i;
    int j=0;
    //var1 distingue entre si es sin trigger o con trigger
    char var1[]={0xA9,0xAD,0x78,0xA9,0xAD,
                 0xB9,0xBD,0x78,0xB9,0xBD};
    //var2 distingue entre el metodo
    char var2[]={0xC6,0x86,0x1E,0x7E};//depende de metodo
    //           NTrig,  ==,  >=,  <=

    /*programa*/
    if(metodo!=0) j=1;

    _outport(0x0212,var1[0+j*5]);
        _outport(0x0212,var1[1+j*5]);
        _outport(0x0210,var1[2+j*5]);
        _outport(0x0212,var1[3+j*5]);
        _outport(0x0212,var1[4+j*5]);
        _outport(0x0213,var2[metodo]);
```

```
        if (PUERTO!=2){
        for (i=0;i<memoria*1024;i++)
                POD1[i]=_inport(0x210);
    }
    else{
        USB_inPods(POD1);
    }
}
```

Para indicar que se quieren los datos del pod1 se hace una serie de outport con los siguientes datos y direcciones.

pod\dirección	0x212	0x212	0x210	0x212	0x212	0x213
pod1	0xA*9	0xA*D	0x78	0xA*9	0xA*D	x1

variable\método	no trigger	= =	>=	<=
x1	0xC6	0x86	0x1E	0x7E

Cuadro 3.4: Inicio de secuencia del pod 1

NOTA 1: los valores que en los que está el símbolo '*' cambian la A por una B cuando hay trigger.

NOTA 2: x1 es un variable que dependen de la variable método, esta variable analiza las parámetros de la configuración de la captura y decide si es con trigger o sin él, y en caso de ser con trigger que tipo de trigger es.

Otro aspecto importante es saber como actúa el analizador cuando la frecuencia de muestreo es de 500MHz o 250MHz. En (1) se indica que si la memoria es de 128K y se muestrea a 500MHz se obtienen 512K a 500MHz y 128K a 125MHz y en el caso de muestrear a 250 MHz ocurre que se obtiene 256K + 256K a 250MHz y 128K a 125MHz.

La duda viene en saber como recoger la información de cada pod, es decir, si solo he conectado 2 pods para el caso de 500MHz como recoger la información para obtener los 512K y los 128K.

Para este análisis se utiliza la información siguiente: cuando se recoge la información de un pod este contiene 8K ó 128K de muestras. Cada muestra es la codificación de la suma de los canales de muestreo en cada instante de muestreo. Ahora bien solo falta saber como se codifica cada muestra. Si cada canal puede valer '0' ó '1', entonces el valor de la muestra en cada instante 'n' corresponde a la expresión:

$$muestra[n] = canal\,0[n] \cdot 2^0 + canal\,1[n] \cdot 2^1 ... + canal\,6[n] \cdot 2^6 + canal\,7[n] \cdot 2^7$$

$$muestra[n] = \sum_{i=0}^{7} canal\,i[n] \cdot 2^i$$

Por tanto si en el pod 1 se pone el canal 0 y el canal 3 a Vcc y el resto a masa, al muestrear el pod 1 todas las muestras tendrán el valor 0x09. Todas las muestras valen lo mismo porque se ha puesto una señal continua y vale 0x09 porque $0x09 = canal\,1 \cdot 2^0 + canal\,3 \cdot 2^3 = 1 \cdot 1 + 1 \cdot 8 = 9_d = 0x09$

Análisis de como recoge los datos a 500Mhz el LA-4540: Lo primero es configurar el analizador del PC1 como se indica en la figura 4 esta información se obtiene de (1).

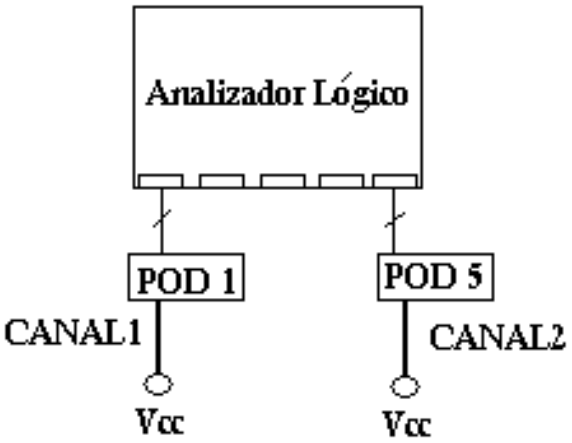


Figura 3.24: Configuración del analizador para 500MHz

En este caso el pod 1 solo tiene el canal 1 con una señal a Vcc y el pod 5 tiene el canal 2 también a Vcc. Entonces cuando se lean las muestras sabremos que solo hay dos posibles valores: o bien 0x02 o bien 0x04.

Ahora se necesitan los listados donde se pueden ver los datos de cada pod, para ellos se pueden seguir los siguientes pasos:

a) PC2: condición de disparo para obtener el pod 1: está podría ser - viendo la tabla 3.4-

evento 0: out(0x212,0xX9)

evento 1: out(0x212,0xXD)

evento 2: out(0x210,0x78)... el 0x78 es el indicador del pod1

Esto sería secuencia [0,1,2] y la codificación de eventos:

pod	POD 3		POD 2								POD 1							
evento	A11	A12	A3	A2	A1	A0	-	$\overline{IOR}$	$\overline{I0W}$	$\overline{SEN}$	D7	D6	D5	D4	D3	D2	D1	D0
evento 0	0	0	0	0	0	0		1	0	0	x	x	x	x	1	0	0	1
evento 1	0	0	0	0	0	0		1	0	0	x	x	x	x	1	1	0	1
evento 2	0	0	0	0	1	0		1	0	0	0	1	1	1	1	0	0	0

Cuadro 3.5: Codificación inicio datos POD 1

b) En PC1: una vez iniciado el programa `main.exe` se pulsa 'Go yes' - da igual con trigger o sin él porque no se ha puesto una condición tan estricta en PC2 sino una que abarca a los dos casos. Pero si debe determinar un caso que sea sin trigger.

c) En PC2: Se obtiene un fichero de captura `.log`. A continuación se ejecuta la aplicación *AN.exe* y por último se analiza el archivo `listado.txt`. *Tras su análisis se comprueba que el valor de las muestras es 0x02.*

d) En PC2: Se pone como condición de disparo los eventos necesarios para la obtener el pod 2 - estos se pueden obtener de la tabla 3.6- .

evento 3: out (0x213,0xC7)

evento 4: out (0x210,0xB8)

Siendo en este caso la secuencia [0,1,2,3,4]. De esta forma el analizador del PC2 se dispararía cuando el PC1 empiece a recibir los datos del pod2 desde su analizador.

e) En PC1: Se vuelve a pedir una captura.

f) En PC2: se vuelve a obtener una captura que tras aplicar el archivo *AN.exe* permite obtener el archivo `listado.txt`. Al mirar las muestras relativas al contenido de las muestras se comprueba que valen 0x02.

g) Ahora habría que poner la condición disparo del PC2 para obtener los datos del pod3, esto es tan simple como cambiar el evento 4 a : out (0x210,0xD8).

Ahora habría que continuar hasta comprobar la información que lleva cada pod, y así saber como funciona el analizador para el caso de 500MHz.

Tras realizar el análisis de los fichero se comprobó que el los pod 1, 2, 3 y 4 contenían como muestras siempre el valor 0x02 y que el el pod 5 contenía muestras con el valor 0x04.

Número de pod	pod 1	pod 2	pod 3	pod 4	pod 5
Valor de sus muestras	0x02	0x02	0x02	0x02	0x04

Esta información llevo a las siguientes conclusiones:

- Las 512K muestras se obtienen de la concatenación de los pod 1, 2, 3 y 4.
- Las 128K muestras se obtiene del pod 5.
- El analizador a la frecuencia de 500MHz funciona de la siguiente forma: El analizador va muestreando a 500MHz, es decir, cada 0'002 us. Estas muestras se van guardando consecutivamente en las posiciones asignadas a los pods 1, 2 , 3, y 4. De forma que primero se llena el pod 1, luego el pod 2, el pod 3 y por último el pod 4. Ahora bien cada 4 muestras a esa velocidad el pod 5 también guarda una

muestra, de forma que al final también se llena el pod 5 pero a una velocidad de muestreo de 0'008 us, es decir, a 125MHz.

- Al conocer como guarda los dato el LA será mas fácil como el representar los datos por pantalla con esta frecuencia. Se podría guardar en una variable A de tamaño 512k la información de pod1 + pod2 + pod3 + pod4 y en otra variable B de 128K la del pod 5. Y cada cuatro muestras dibujadas de la variable A se pinta una de la variable B, manteniendo su valor durante el tiempo que no se vuelve a saber lo que vale.

Análisis de como recoge los datos a 250Mhz el LA-4540: Como en el caso anterior hay que configurar el analizador de una forma determinada, en este caso como en la figura 3.25.

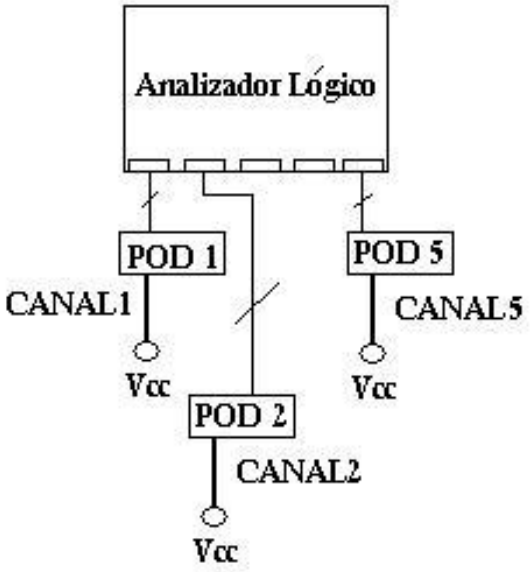


Figura 3.25: Configuración del analizador para 250MHz

La configuración de los canales de cada pod son:

- Pod1: todos a masa excepto el canal 1 que esta a Vcc; se codifica como 0x02.
- Pod2: todos a masa excepto el canal 2 que está a Vcc; se codifica como 0x04.
- Pod5: todos a masa excepto el canal 5 que está a Vcc; se codifica como 0x20.

Ahora hay que volver a realizar el mismo proceso de análisis que en el caso de 500Mhz. Al analizar los listados de cada pod se comprueba:

número de pod	pod 1	pod 2	pod 3	pod 4	pod 5
valor de sus muestras	0x02	0x04	0x02	0x04	0x20

Con esta información se llego a las siguientes conclusiones:

- La información del pod 1 se obtiene de concatenación del pod 1 y pod 3; 128K·2= 256K.
- La información del pod 2 está concatenada en los pod 2 y pod 4.
- En el pod 5 se obtiene la información del pod 5.
- Esta vez el funcionamiento a 250 MHz consiste en muestrear a esa frecuencia consiguiendo hasta 256K muestras usando dos pods diferentes. Y cada 2 muestras para el pod1 y pod2 se muestrea una vez en el pod 5, de esta forma se entiende como a la frecuencia de 250Mhz se pueden conseguir 256K muestras cuando se tiene solo memoria para cada pod de 128K como máximo.

5. **void rec_pod2()**

   Procedimiento que permite recoger los datos del pod2.

   Es similar a void rec_pod1() pero las instrucciones outportb para identificar al pod se hacen a otras direcciones.

```
void rec_pod2()
{
      unsigned long i;
      int j=0;
      char var1[]={0xC7,0x87,0x1F,0x7F};//depende de metodo
      char var2[]={0xC6,0x86,0x1E,0x7E};
      char var3[]={0xA9,0xAD,
                   0xB9,0xBD};
      if(metodo!=0) j=1;


     _outport(0x0213,var1[metodo]);
     _outport(0x0210,0xB8);
```

```
        _outport(0x0212,var3[0+j*2]);
        _outport(0x0212,var3[1+j*2]);
        _outport(0x0213,var2[metodo]);
        if(PUERTO!=2){
            for(i=0;i<memoria*1024;i++)
                    POD2[i]=_inport(0x210);
        }
        else{
             USB_inPods(POD2);
        }
}
```

La forma de pedir los pod es similar para los cinco pods y los distintos parámetros de entrada. Las direcciones de salida son siempre las mismas para los pods 2, 3, 4 y 5 lo que cambia son los datos. En el caso de no trigger:

pod\dirección	0x213	$0x210^7$	0x212*	0x212*	0x213
pod2	x1	0xB8	0xA9	0xAD	x2
pod3	x1	0xD8	0xA9	0xAD	x2
pod4	x1	0xE8	0xA9	0xAD	x2
pod5	x1	0xF0	0xA9	0xAD	x2

variable\método	no trigger	= =	>=	<=
x1	0xC7	0x87	0x1F	0x7F
x2	0xC6	0x86	0x1E	0x7E

Cuadro 3.6: Codificación inicio datos POD 2,3,4 y 5

nota1=Las columnas con el símbolo '*' cambian su valor 0xA9 y 0xAD por 0xB9 y por 0xBD cuando hay trigger

nota2: x1 y x2 son valores aleatorios que dependen de la variable método.

6. **void rec_pod3()**

   Procedimiento para pedir los datos del pod3.

   Es idéntico a void rec_pod2 pero en este caso el identificador es 0xD8.

7. **void rec_pod4()**

   Procedimiento para pedir los datos del pod3.

Es idéntico a void rec_pod2 pero en este caso el identificador es 0xE8.

8. **void rec_pod5()**

   Procedimiento para pedir los datos del pod3.

   Es idéntico a void rec_pod2 pero en este caso el identificador es 0xF0.

9. **void continua1x()**

   Procedimiento que se transmite después de go2(). Contiene toda la información relativa a los parámetros de configuración del analizador a excepción del contenido de los eventos.

   La obtención de un único continua1 fue bastante complicado, observando el diagrama de funcionamiento del LA-4540 de la figura 3.3.4 se pueden contabilizar hasta 4 continua12 distintos[8], y conseguir que un solo archivo pudiese ser válido para todas las frecuencias fue un laborioso proceso de comparar archivos, crear nuevas versiones y su posterior prueba. Por esta razón, el continua1x , continua2x y fin_configuración antes de la zona de código donde van los OUT y los IN existe una zona de código para adecuar el valor de las variables a todas las frecuencias de muestreo posibles.

10. **void continua2x()**

    Procedimiento que se transmite después de continua1x().

    En continua2x se analiza el valor de la variable método y se envía o bien una versión para no trigger o bien una versión para trigger:

    - **void continua2x_NT**
    - **void continua2x_ST**

11. **void fin_confi()**

    Último procedimiento que se envía para la petición de una captura. Tiene una estructura similar a continua1x en cuanto que vuelve a configurar al analizador, pero los valores que manda son un tanto distintos.

12. **void threshold(unsigned char umbral[11],float tensión,bool archivo)**

    Threshold es un procedimiento que calcula los 11 valores que corresponden al valor umbral deseado.

    Entrada:

[8] no trabaja con los continua11 porque como se ha comentado anteriormente para hacer funcionar al LA-4540 basta con los archivos del estado2

- archivo==1 indica que llamó desde continua1x
- archivo==0 indica que llamó desde fin_confi
- tensión=indica que tensión umbral es la que hay que codificar

Salida:

- umbral[11]: vector donde se devuelve el valor correspondiente a la tensión umbral indicada en la variable tensión

El rango del voltaje umbral es de [-6'40, +6'40] y la precisión máxima del LA-4540 es de 0'05V.

Para poder obtener el parámetro que indica los voltajes fue necesario crear un nuevo procedimiento dentro del programa *AN.exe*. Dado que la información de los voltajes siempre se indica en las mismas instrucciones, y existe una forma de saber en que posición están - fijándose qué línea ocupan al abrir los listado con el programa Norton Comander - se le puede indicar al programa *AN.exe* que cuando vaya a escribir una instrucción que coincida con el número de línea en que sabemos que se indica esa información llame al nuevo procedimiento para que escriba la información de los voltajes.

Entonces ahora el método de análisis consiste en hacer una captura a -6'40V, -6'35, -6'30 ... para después ejecutar el programa AN.exe y obtener en un fichero como se indica cada voltaje.

Listing 3.5: Listado del fichero.txt donde se muestran los voltajes

```
//lineas726,729,732,735,738,741,744,747,750,751,752,
unsigned char matriz[27][11]={
N00:    [0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
N01:    [0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0,
N02:    [0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  0,
N03:    [0,  0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
N04:    [0,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,
N05:    [0,  0,  0,  0,  1,  0,  1,  0,  0,  0,  0,
N06:    [0,  0,  0,  0,  1,  1,  0,  0,  0,  0,  0,
N07:    [0,  0,  0,  0,  1,  1,  1,  0,  0,  0,  0,
N08:    [0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  0,
N09:    [0,  0,  0,  1,  0,  0,  1,  0,  0,  0,  0,
N10:    [0,  0,  0,  1,  0,  1,  0,  0,  0,  0,  0,
N11:    [0,  0,  0,  1,  0,  1,  1,  0,  0,  0,  0,
```

```
N12:    [0,  0,  0,  1,  1,  0,  0,  0,  0,  0,  0,
N13:    [0,  0,  0,  1,  1,  0,  1,  0,  0,  0,  0,
N14:    [0,  0,  0,  1,  1,  1,  0,  0,  0,  0,  0,
N15:    [0,  0,  0,  1,  1,  1,  1,  0,  0,  0,  0,
N16:    [0,  0,  1,  0,  0,  0,  0,  0,  0,  0,  0,
N17:    [0,  0,  1,  0,  0,  0,  1,  0,  0,  0,  0,
N18:    [0,  0,  1,  0,  0,  1,  0,  0,  0,  0,  0,
N19:    [0,  0,  1,  0,  0,  1,  1,  0,  0,  0,  0,
N20:    [0,  0,  1,  0,  1,  0,  0,  0,  0,  0,  0,
N21:    [0,  0,  1,  0,  1,  0,  1,  0,  0,  0,  0,
N22:    [0,  0,  1,  0,  1,  1,  0,  0,  0,  0,  0,
```

Al analizar este listado se pudo obtener la fórmula matemática para poder indicar los voltajes, para mas información mirar el archivo.cpp llamado Archivo_Umbral.cpp que contiene el procedimiento **threshold.**

13. **once**

Es el procedimiento que controla a todos los descritos anteriormente, se puede decir que esta un nivel por encima de ellos. Se encarga de actualizar las variables externas para después ir llamando a los procedimientos anteriores y conseguir realizar una captura,

Distingue entre tres posibilidades:

- Funcionamiento en modo analizador
- Funcionamiento en modo generador de patrones
- Funcionamiento en modo generador de patrones + analizador lógico

Listing 3.6: Listado de procedimiento once

```
void once(int vez)
{
      if(frec==0)                    indice_frec=0;
      if(frec==1)                    indice_frec=1;
      if((frec>=2)&&(frec<=5))       indice_frec=2;
      if((frec>=6)&&(frec<=19))      indice_frec=3;

/*opcion primera: muestrear por los 5 podŽs
    if(modoLA==0)
```

```
    {
        go2();//depende de frec
        if((frec>=0)&&(frec<=19))
        {       //permiso=1;
                continua1x();
                //nombre++;
                //permiso=0;
                espera_dato();

                //permiso=1;
                continua2x();
                //nombre++;
                //permiso=0;
        }


         rec_pod1();
         rec_pod2();
         rec_pod3();
         rec_pod4();
         rec_pod5();
         permiso=1;
         //terminamos la transmision
         fin_confi();
         nombre++;
         permiso=0;
         comprobacion(0,0,0,nombre,permiso);


/*variables para necesarias para poder utilizar
correctamente inicio_trigger y fin_trigger  */
         ciclo=1;
         ant_metodo=metodo;
         ant_modo=modo;
         ult_frec_go=frec;
         ult_memoria=memoria;
```

```
        }

    /*opcion segunda: solo generar un patron
     continuo por POD 1 y POD 2*/
        if(modoLA==1)
        {
              if((metodoSalida==1)&&(iteracion==1))
                    {//fe[200Hz,100MHz]
                     patron_continuo();
                     iteracion=2;

                    }

    /*opcion tercera:generar un patron
    por POD1 y POD2, muestrear por POD3 POD4 y POD5*/
              if(metodoSalida==0)
                   {
                    //fe[200Hz,5MHz]
                    if (metodo==0)      patron_muestreo_NT();
                    else                patron_muestreo_ST();

                   }
         }

    }
```

14. **void enviar_trigger()**

    enviar_trigger es un procedimiento que indica al LA cuales son las condiciones de disparo deseados deseados. Este procedimiento sigue los pasos realizados para el analizador LA-32000 del cual existe código en ensamblador.

    Este procedimiento se llama cada vez que se cambia de frecuencia o se llama al menú trigger.

    Lo primero que hace es mirar si existe algún valor de la secuencia eventos distinto de 'x'. En caso de existir comprueba que el evento o eventos indicados en secuencia tienen algún valor distinto de 'x'. De existir alguno codifica la información ya se la manda al

analizador.

Para poder hacer la codificación de eventos fue necesario recurrir al código en ensamblador de la serie LA-32000. Como se comento al principio de esta memoria ese código no es válido para este analizador, sin embargo, con un pequeño añadido se puede conseguir la forma en que se manda la codificación de eventos.

enviar_trigger() esta compuesto por cuatro procedimientos llamados:

- void comprobar_secuencia(int *p,char sec_v[16],unsigned char contenido[16]):

  comprobar_secuencia analiza la variable secuencia e indica el numero valores distintos de 'x' y los ordena en una variable llamada sec_v.

- void iniciotg(int n,unsigned char contenido[16])

  iniciotg manda una serie de outportb e inportb que hacen referencia al numero de eventos a transmitir, variables propias del grupo de frecuencias donde estemos e información relativa a los parámetros del estado anterior.

- void txbloques(int índice,char sec_v[16])

  txbloques transmite 255 bloques de información, esta información hace mención a la condición de disparo relativa a un evento.

- void fintg(int n,int ind,unsigned char contenido[16])

  fintg se transmite después de cada bloque de información transmitida, en una de sus lineas indica al LA del numero de eventos que quedan por ser transmitidos.

15. **void patrón_continuo()**

    patrón_continuo() es un procedimiento que sirve solo y exclusivamente para generar patrones. En este caso el LA-4540 no funciona como analizador sino como generador.

    El usuario tiene dos opciones para introducir un patrón - solo funciona con los pod1 y pod2, el rango de frecuencias va desde 200Hz hasta 100Mhz, y con 8K de memoria(la opción de 128K esta por implementar)-.

    a) Muestrear una señal con los pod 1 ó 2 en modo analizador, esta información se queda en las variables que guardan los datos muestreados, y después pasar al modo generador y al ejecutar el procedimiento patrón_continuo() este manda la información que se encuentra en esas variables.

    b) Leer un patrón desde un archivo de texto. El archivo de texto tiene una estructura determinada.

La primera linea es una cabecera para identificar que es un archivo de patrones válido para ser leído desde la aplicación software.

Después viene una etiqueta con la forma '[POD1]' y a continuación los datos en valor hexadecimal, separados por un espacio, de los 8 canales del pod1.

Por último, otra etiqueta '[POD2]' y a continuación los datos del pod 2.

El numero máximo de datos por pod es de 8K, entendiendo por dato 8bits. El final de los datos viene indicado por el carácter '*'.

Para mas detalles del fichero para cargar patrones consultar 3.41 en la página 151.

Siguiendo con la analogía de dividir el comando de generar patrones en distintos archivos se muestra la figura 120 con dicha secuencia:

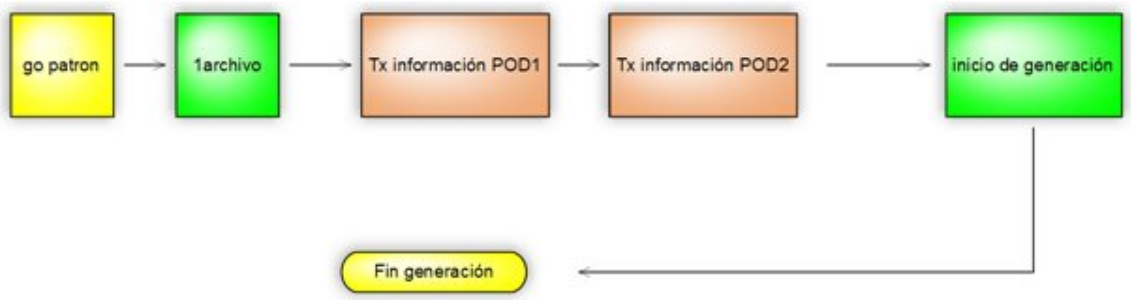


Figura 3.26: Diagrama de bloque de generación de patrones

16. **void patrón_muestreo_NT()**

    patron_muestreo_NT() es un procedimiento que permite muestrear por los pods 3, 4 y 5 y generar un patrón por los pods 1 y 2.

    Como se ha indicado anteriormente en el modo generador de patrones las frecuencias válidas son desde 200Hz hasta 100Mhz, y la opción de memoria sólo se ha implementado la de 8K

    Las opciones para introducir los patrones son las mismas que en el caso de patron_continuo().

17. **void patron_muestreo_ST()**

    patron_muestreo_ST es idéntico a patron_muestreo_NT() pero permite que se empiece a muestrear cuando se cumpla la condición de disparo indicada para los pods 3 , 4 y 5.

Tanto para patron_muestreo con trigger o sin él, los posibles archivos en que dividirse serían los mostrados en la figura 121:

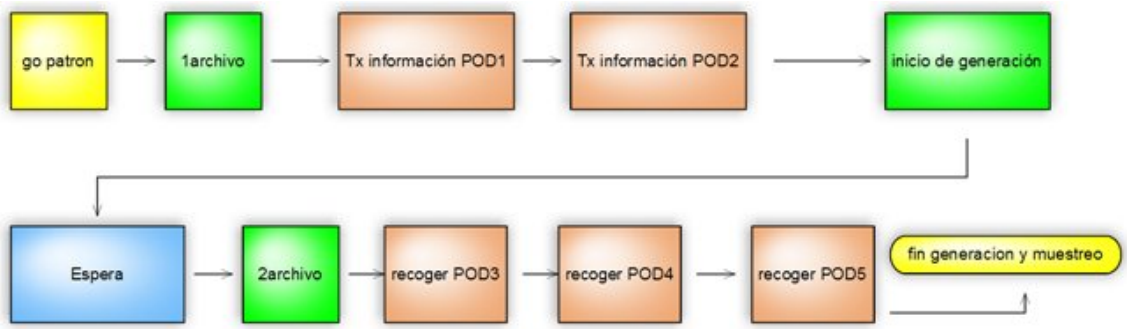


Figura 3.27: Diagrama de bloques generación patrón y muestreo

Nota: los 3 procedimientos referentes a generar patrones son muy similares, y podrían juntarse en uno sólo, sin embargo debido a la falta de tiempo no se ha podido realizar esa mejora.

3.3.9. Realimentación en el proceso de ingeniería inversa

La realimentación sirve para comprobar que los procedimientos y funciones que se van creando funcionan correctamente. A lo largo que el proyecto iba creciendo se han ido creando distintos métodos de realimentación.

La primera realimentación que se hizo fue al conseguir los archivos de captura correspondientes a la obtención de datos - rec_pod1(), rec_pod2(), rec_pod3(), rec_pod4() y rec_pod5() -. Se desea conocer si al poner como condición de disparo del procedimiento rec_pod1() los datos que se obtienen corresponden a la señal que se está muestreando y cuantos datos se han recogido; con esto se llega a comprender como recoge los datos el PC de cada pod.

En primer lugar hay que obtener un fichero con una condición de disparo con la que obtener el `archivo.la` correspondiente a rec_pod1[9] , a continuación tras ejecutar *AN.exe* se obtiene un listado compuesto por unos outport que hacen referencia a el encabezamiento de petición del primer pod, Después una serie de inportb a la dirección 0x211, outportb como referencia del segundo pod y por último otra serie de inportb a la dirección 0x211.

En el análisis de este listado se supone:

- Que la primera serie de inportb son los datos del primer pod

[9]Para ver el trigger mirar la tabla 3.4

- Que se han conseguido todos los datos de ese muestreo
- También se puede suponer que la segunda serie hace referencia a los datos muestreados en el segundo pod pero no están todos, ya que no se ha podido obtener el encabezamiento del tercer pod.

Ahora falta saber si los datos muestreados en el pod1 se han recuperado correctamente o falta algo. Para ello se parte de una situación conocida.

En el PC1 sólo se muestrea con el pod1 y sólo con el canal 0, y se está muestreando una señal del entrenador de la cual se conoce su periodo ya que se ha comprobado con el osciloscopio. Por último hay que seleccionar una frecuencia de muestreo para el PC1 que cumpla el teorema de Nyquist para poder muestrear la señal anterior y la memoria seleccionada debe ser de 8K.

En el PC2 la frecuencia de muestreo es siempre la misma 10Mhz, ya que el bus ISA va a 4.77Mhz y la memoria seleccionada es de 128K.

Al obtener el archivo listado hay que fijarse en la zona comprendida entre la última línea del encabezamiento del pod1 y la primera del encabezamiento del pod2 ,es decir, solo los inportb correspondientes al pod1.

Recordando que cada línea del archivo listado corresponde con un comando outportb o inportb se obtiene que:

- El número de inport corresponde a la capacidad de memoria elegida en PC1, en este caso memoria vale 8K, y por tanto habrá 8192 inportb a la dirección 0x211. Para comprobarlo vasta con contar los inport del listado obtenido. Como cada línea es un comando se puede seleccionar las líneas de interés , copiarlas y pegarlas en un nuevo archivo generado por el programa DevC++. De esta forma se puede saber rápidamente el número de inportb existentes
- El osciloscopio ha servido para saber la frecuencia de las señal cuadrada a muestrear, y como sólo puede haber dos valores - 0x01 ó 0x00 - si se cuenta una secuencia de 0x01 más otra de 0x00, se suman y se multiplica el resultado por el inverso de la frecuencia de muestreo del PC1 se puede conocer la frecuencia de la señal originalmente muestreada.
- Para que quede mejor explicado esta realimentación se explica un ejemplo. Sea una señal cuadrada de frecuencia 100Hz muestreada desde el PC1 a una frecuencia de 500Hz como se muestra en la figura 3.28; esta señal se muestrea en el canal 0 del pod1. Al analizar el listado obtenido correspondiente a la captura de fichero de recogida de datos - recpod1- en el PC2 se obtendría una secuencia de inportb cuyos datos serían :

- datos:[0x01, 0x01, 0x01, 0x00, 0x00, 0x01, 0x01, 0x01, 0x00.....].

Entonces, un periodo de la señal está compuesto por 3 veces 0x01 más 2 veces 0x00, es decir, cada 5 muestras se repite la señal y ese es el número de muestras de un periodo de esa señal. Si multiplicamos el número de muestras de un periodo por el inverso de la frecuencia de muestreo se obtendrá el periodo de la señal que se esta muestreando.

$$5muestras \cdot 0{,}002s = 0{,}01s$$

Que efectivamente es el periodo de las señal cuadrada.

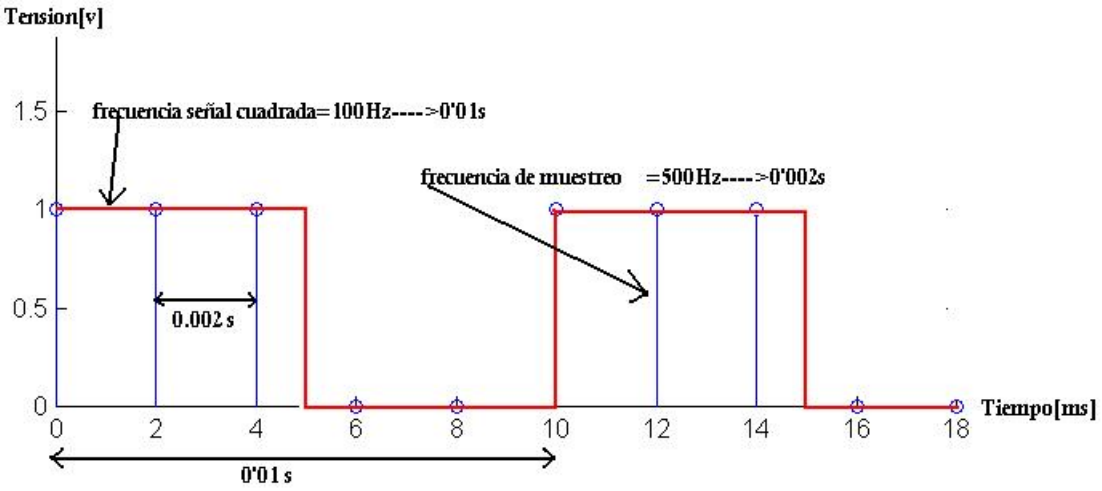


Figura 3.28: Ejemplo de señal a muestrear

La segunda manera para conseguir realimentación consiste en una vez conseguido el código para realizar una captura crear un archivo en lenguaje C, que llevado al PC1 y compilado en Borland Compiler permita poder manejar el analizador desde MS-DOS.

El problema de este método es que no se puede saber directamente si el LA está funcionando correctamente, por lo que es necesario muestrear el bus ISA del PC1 desde el PC2 poniendo la condición de disparo que hace referencia a la zona de recogida de datos de los pod, rec_pod1 por ejemplo, y analizar si se han recogido correctamente las muestras.

De todas formas este método de realimentación es totalmente válido para comprobar otras funciones del analizador, como son la generación de patrones. Una vez conseguido el código que genera un patrón determinado se puede llevar el código el PC1 y ejecutarlo directamente

y comprobar si está funcionando poniendo la salida de los pod generadores en los diodos leds del entrenador.

El tercer método para conseguir la realimentación proviene de la necesidad de poder comprobar que los procedimientos de la aplicación propia mandan la información correctamente sin necesidad de probarlos directamente sobre el analizador LA-4540.

Como ya se ha ido explicando anteriormente cada vez que se quiere conocer un nuevo parámetro - la tensión de salida, la memoria, el trigger ... - hay que hacer una captura del comando de forma completa - continua1, continua2 , recpodx, fin_confi - ya que el software de la aplicación no manda lo mismo para indicar 8K ó 128K, o cualquier otro parámetro.

En el proceso de captura de los parámetros del LA-4540 se van guardando los distintos `listados.txt` correspondiente a cada procedimiento - continua1, continua2... - estos listados están compuestos de lineas en las que se indica la dirección, el dato y el tipo de instrucción. Habrá un continua1 de 8K y otro de 128K , etc...

Dado que ya se tiene una aplicación software propia que cuando se ordena realizar una captura ejecuta instrucciones inportb o outportb; la solución adoptada consta en que la aplicación al leer la instrucción inportb o outportb haga el inportb o outportb y además escriba en un archivo de salida .txt información relativa tipo de instrucción, dirección y dato. De esta forma se obtiene un archivo idéntico a los listados creados con *AN.exe*.

Entonces cada vez que se modifica un procedimiento para incluir un nuevo parámetro, por ejemplo 128K en continua1, se ejecuta con la aplicación propia cambiando ese parámetro a 128K y se obtendrá un archivo de salida que debe ser idéntico al obtenido al muestrear el comando captura a 128K desde el PC1.

Para poder realizar esta realimentación se crea un procedimiento llamado: *void comprobación(int port, int dat,int tipo,char nombre,bool permiso).*

Comprobación es un procedimiento que crea un fichero de texto y escribe en él. Sus parámetros son:

- int port, int dat: la dirección y el dato a los que hace referencia la instrucción inport o outport.
- int tipo: sirve para saber si se llama desde inport o outport, y así saber si es IN o OUT.
- char nombre: nombre es un numero, y es el nombre del archivo de salida.
- bool permiso: es una variable global que cuando vale 1 permite escribir en el fichero .txt de salida.

Este procedimiento llamado comprobación se añade a las llamadas inport y outport; de forma que inport no solo llama a inportb sino también a comprobación pasándole el dato y la dirección que se tenia que transmitir por el bus ISA.

La forma de utilizarlo es la siguiente: *cuando se desea ver lo que manda un procedimiento, continua1- o únicamente alguna instrucción outport o inport- se debe añadir antes del procedimiento una linea en la que ponga 'permiso=1' y después del procedimiento otra donde ponga 'permiso=0' y otra con 'nombre++' - de esta forma si se desea comprobar mas de un procedimiento cada uno tendrá un nombre.* En el listado de once se puede ver un ejemplo de este método para poder conocer lo que manda fin_confi().

3.4. Desarrollo de la aplicación de usuario

3.4.1. Entorno de desarrollo

Las especificaciones del proyecto indican que la interfaz debe desarrollarse en lenguaje C utilizando la librería SDL. Una de las posibles herramientas de programación que permiten cumplir con estos requisitos es el entorno de desarrollo Bloodshed Dev-C++.

Dev-Cpp es un entorno integrado de desarrollo para C / C++ que presenta unas características más que aceptables para ser un compilador gratuito. Es accesible, totalmente compatible y multiplataforma (éste es un detalle importante). El código que use librerías para diferentes plataformas, puede ser recompilado en ellas sin ningún problema. Utiliza el compilador incluido Mingw, con todas las herramientas de compilación que proporciona mingw típicamente (gcc y g++), aunque puede utilizarse con otros compiladores como Cygwin. También es posible editar opciones del linker y opciones para la línea de comandos, o utilizar una avanzada función de autocompletar. La navegación entre clases, funciones, y ficheros es muy fácil, muy intuitiva.

Permite programar de forma profesional, incluso se puede aprender de cero si no se sabía nada. El manual incluido en su distribución se encuentra en inglés, pero no es en absoluto un problema ya que la interfaz es tan simple que es posible realizar aplicaciones, ventanas y DLL's como el mejor profesional. La versión utilizada para el proyecto de desarrollo de la interfaz gráfica es Dev-C++ 4.9.9.2.

El desarrollo de la interfaz se basa en Simple DirectMedia Layer (SDL), que es un conjunto de librerías desarrolladas en lenguaje C capaces de proporcionar funciones básicas para realizar operaciones de dibujo 2D, gestión de efectos de sonido y música, y carga y gestión de imágenes. Aunque también proporciona herramientas para el desarrollo de videojuegos y aplicaciones multimedia. Una de sus grandes virtudes es que se trata de una librería multiplataforma, soportando oficialmente los sistemas Windows, Linux, MacOS y QNX, además de otras arquitecturas/sistemas como Dreamcast, GP32, GP2X... Por otra parte, la librería se distribuye bajo la licencia LGPL, lo que ha provocado el gran avance y evolución de las SDL.

Sin embargo, existen disponibles en la red diversas librerías desarrolladas a partir de SDL que implementan las operaciones básicas a realizar sobre una ventana gráfica, por lo que se decide explorar el uso de alguna de ellas. Estos desarrollos simplifican enormemente el uso de SDL y suponen un buen punto de partida para el desarrollo de la interfaz gráfica. Aprovechan-

do la experiencia de nuestro tutor en el desarrollo de aplicaciones basadas en SDL se parte de otro proyecto desarrollado previamente mediante la librería QuickCG.

La librería QuickCG contiene algunas funciones básicas para dibujar sobre una ventana como son: funciones primitivas 2D, conversiones de color, carga de imágenes bitmap, escritura de texto, o lectura de pulsaciones desde teclado. Para el desarrollo del proyecto ha sido necesario estudiar en primer lugar el contenido de la librería para poder sacar el máximo partido a las funciones presentes. En el anexo del CD se incluyen sendos manuales que indican como configurar el entorno de desarrollo Dev-C++ para compilar la librería QuickCG, así como el principal contenido de la librería: variables, tipos de datos existentes, funciones, y algunos ejemplos sencillos. Ambos manuales son útiles para dar unos primeros pasos en el entorno de desarrollo y entender la filosofía de trabajo con QuickCG. Complementando la librería pueden encontrarse funciones adicionales que contienen pequeñas modificaciones respecto a las estándar. Han sido creadas para resolver problemas específicos que han surgido en el desarrollo de la aplicación y por su similitud no requieren de un comentario adicional.

Lo expuesto anteriormente es el punto de partida para la creación de la aplicación de usuario. Se necesita de un entorno de desarrollo integrado (IDE) como es Dev-C++, de los ficheros que conforman la librería SDL, y de la librería QuickCG. A continuación se crea un proyecto mediante Dev-C++ y se van añadiendo ficheros según las necesidades del proyecto. Al añadir los ficheros al proyecto éstos se enlazan y compilan de forma conjunta, creándose un fichero ejecutable que es la propia aplicación de usuario. Todos estos archivos se encuentran contenidos en una carpeta para que el proyecto sea portable sin necesidad de instalación alguna.

3.4.2. Ficheros del proyecto

El proyecto software de desarrollo de la interfaz gráfica se compone por tanto de un conjunto de ficheros que se enlazan y compilan en el entorno de desarrollo Dev-C++ conforme a su interdependencia. Los ficheros pueden agruparse según su cometido en varios grupos: librerías de menús, librerías de dibujo, librerías de configuración del LA, y librerías de comunicación con el LA. Cada archivo estará orientado a una de esas tareas dentro del proyecto.

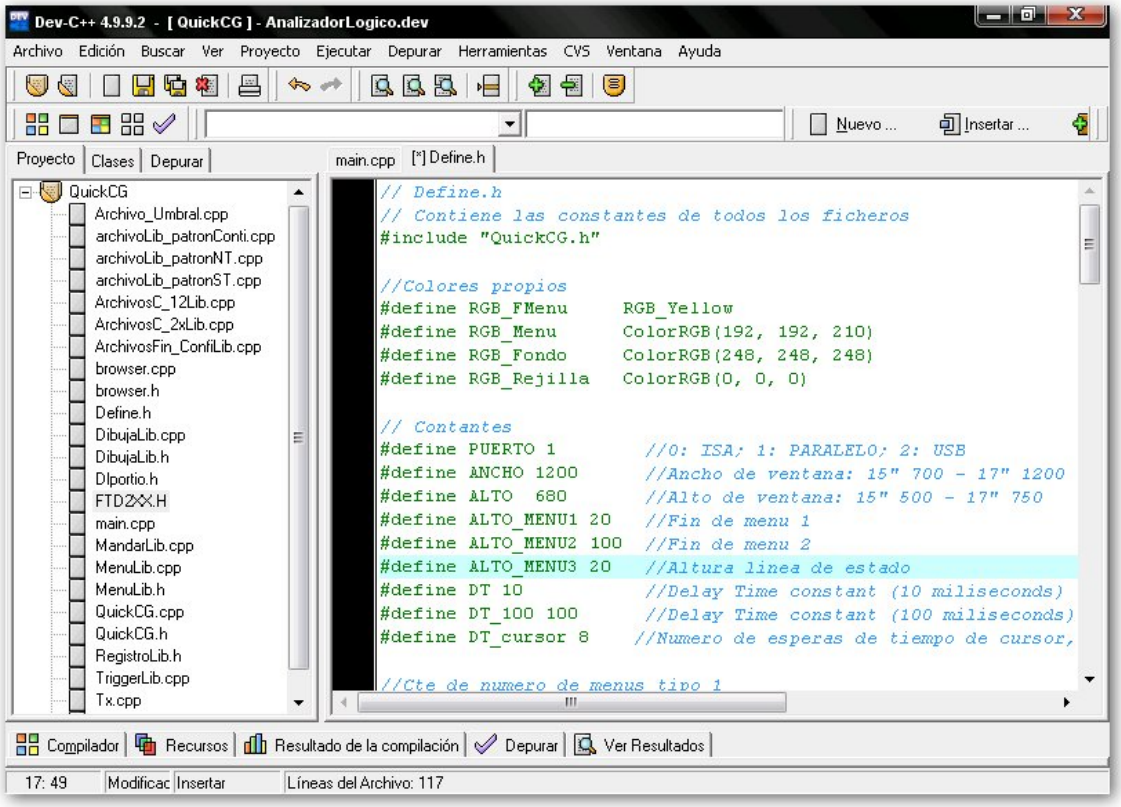


Figura 3.29: Entorno de desarrollo Dev-C++

A continuación se describen los ficheros del proyecto detallando su contenido e indicando su principal cometido. Puesto que el tamaño de los ficheros es grande y están comentados no se estima necesario que aparezcan impresos en esta memoria. Los ficheros pueden consultarse en el CD adjunto al proyecto. Para su posterior uso bastará con leer cada fichero y apoyarse en los comentarios de cada función para entender las líneas de programación que contiene. No obstante, se describen de forma genérica para justificar su existencia y finalidad.

Los ficheros .h contienen la declaración del prototipo de las funciones de los ficheros .cpp que deban ser accesibles desde otros ficheros. Siguiendo con la estructura típica de los ficheros en lenguaje C, si se incluye un fichero mediante la directriz de inclusión *#include* las funciones declaradas en el fichero .h son ahora accesibles. Estas relaciones modifican la forma en que los ficheros se enlazan.

Main.cpp

Es el fichero que contiene la función principal de programa. Se encarga de crear la ventana de programa, dibujar el contenido de la aplicación, y realizar el control del flujo de programa, inclusive ordenar las capturas. Es el punto de entrada al programa y también el punto de salida.

Se hace necesario almacenar de forma estática la información de los objetos que se dibujan en pantalla. Esta información es el contenido de los menús, en qué estado inicial se encuentran, en qué posición se dibujan en pantalla, contenido de menús emergentes, etc. Cualquier texto que deba imprimirse en pantalla se encuentra convenientemente almacenado como elemento o ítem de una estructura de datos. Dichas estructuras contienen información relativa a: menú activado, número de ítem del menú, ítem activo por defecto, posición de pantalla en la que se dibuja el menú, tecla de activación de menú, nombre de menú, y los ítem que contiene cada menú. Diferentes funciones se encargarán de dibujar el contenido en la pantalla según proceda.

Las funciones que se encargan de dibujar los menús actualizan la pantalla con cada acción que el usuario introduce por teclado. El cambio de una opción sobre la pantalla implica un cambio en el ítem activo de un menú. Puesto que la información de un menú está contenida en una estructura, el cambio de ítem activo se refleja directamente sobre el campo adecuado de dicha estructura (siempre que sea posible). Posteriormente, unas variables globales que contendrán el valor del ítem activo dentro de cada menú se actualizarán con la información de las estructuras. La pantalla se refrescará por zonas y cada función se encargará de refrescar una parte concreta de pantalla. El orden con el que se refrescan los diferentes objetos es importante.

Las variables globales se utilizan en este caso para almacenar información del programa en ejecución en otro formato. Pueden incluirse facilmente en otros ficheros y ser consultadas por cualquiera de sus funciones, que nunca alterarán su valor. Es una forma fácil de poder consultar el estado del programa principal desde cualquier función de un fichero para tomar decisiones sin tener que conocer las estructuras de datos que contienen la información real.

El flujo de ejecución del programa principal estará basado en el inicio del analizador lógico a un estado conocido y en un bucle infinito donde se realizan las principales tareas de la aplicación. Éstas son la selección de las diversas opciones de funcionamiento del analizador lógico, el reflejo del cambio de las opciones en pantalla, la configuración del analizador, realizar capturas de datos, y dibujar los datos sobre la pantalla. Para mostrar a grandes rasgos el control de programa ejercido por la función principal se incluye el diagrama de la figura 3.30.

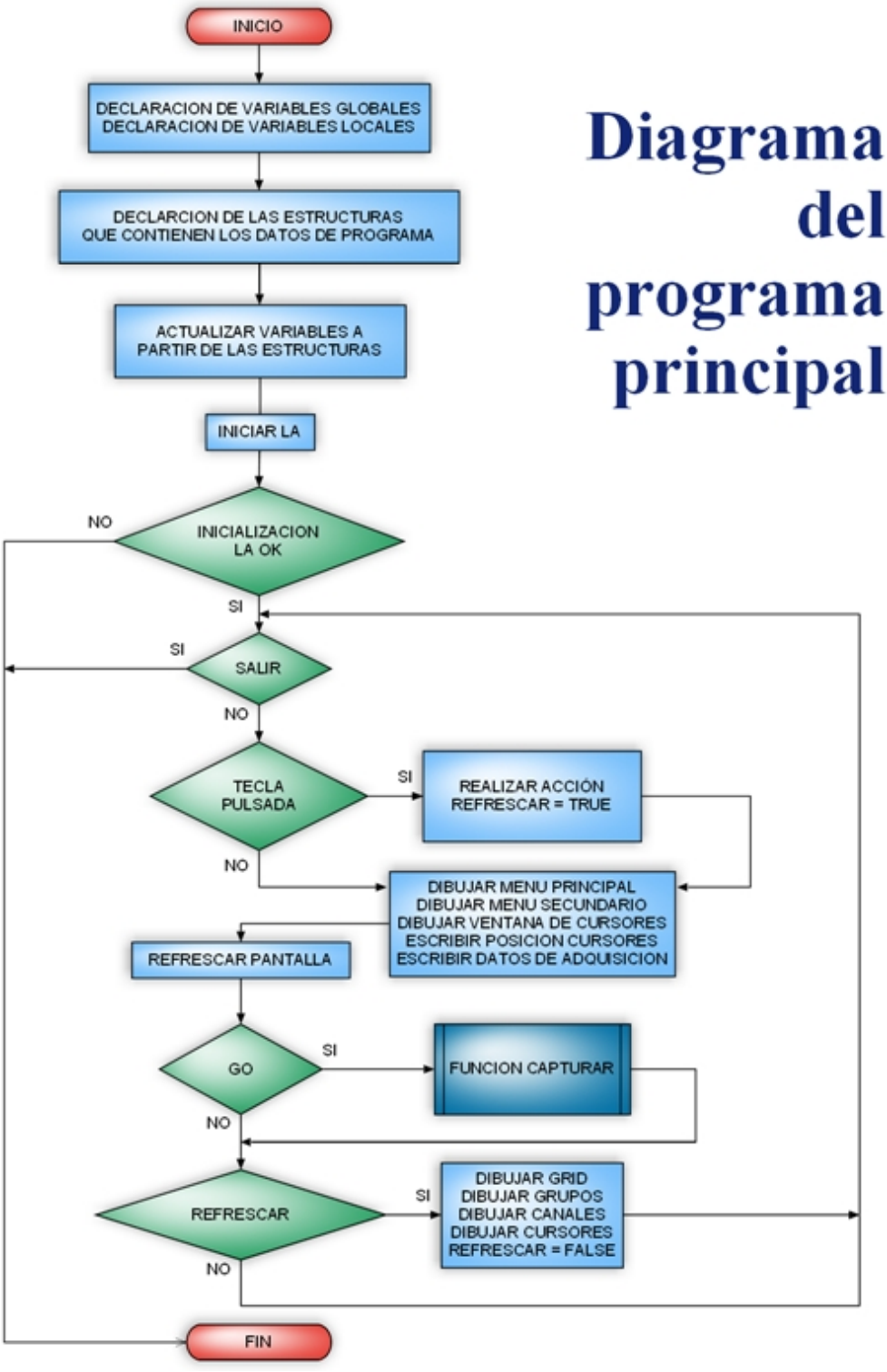


Figura 3.30: Diagrama de flujo general de la aplicación

Una vez conocidos los fines de la función principal a continuación pasan a describirse las distintas funciones que hacen posible estos objetivos. Para ello se analiza el contenido del fichero con más detalle.

En su comienzo se realiza la declaración de las variables globales que se consultarán en los diversos ficheros del proyecto. Estas variables contienen el estado en que se encuentran las opciones de configuración del analizador y estados de funcionamiento del programa principal. Para una mayor identificación de las mismas se las ha dado un nombre que recuerde a la opción del programa MS-DOS al que hacen referencia. Éstas opciones son tales como:

- reloj: fuente de reloj interna o externa.
- frec: frecuencia de captura.
- go: realizar una captura.
- captura: modo de captura una vez, repetir en trigger, o aleatorio.
- memoria: tamaño del buffer de adquisición de datos.
- frecAnt: último valor de frecuencia con el que se realizó una captura completa.
- zoom: relación de tamaño con la que se pintan las muestras en la pantalla.
- matriz_trigger[16][40]: matriz que almacena las palabras de configuración de trigger.
- secuencia[16]: vector que almacena la secuencia de trigger words.
- modo_sec: almacena el modo de captura de trigger en un ancho de pulso.
- nveces: número de veces que se cuenta la repetición de la última palabra de la secuencia.
- modo: lógica de trigger verdadera o falsa.
- volt1, volt2, volt3: tensiones umbrales para POD1-2, POD3-4, y POD5.
- modoLA: modo de funcionamiento del analizador como lógico o generador de patrones.
- metodoSalida: modo de funcionamiento del generador de patrones como Output Until Trigger o Continuo.
- curApos, curBpos, curTpos, curPpos: posiciones de los cursores dentro del buffer de datos y posición de comienzo de dibujo de la pantalla respecto al buffer adquirido almacenado.

- POD1[128*1024], POD2[128*1024], POD3[128*1024], POD4[128*1024], POD5[128*1024]: variables globales tipo unsigned char que almacenan el contenido del buffer de adquisición de datos.

Todas estas variables hacen referencia a opciones de manejo del analizador lógico y la explicación de su uso puede ser encontrada en la guía de manejo del analizador. Si se analiza con detalle un fichero `log` generado por el software del LA se llegará a la conclusión de que el fichero y la aplicación de usuario poseen variables con la misma finalidad. Aunque la aplicación de usuario desarrollada contiene menor cantidad de esas variables ya que todas las opciones de funcionamiento no han sido implementadas, así como tampoco la aplicación gráfica tiene desarrolladas todas las funcionalidades de la aplicación `MAIN.EXE` de MS-DOS.

A continuación se encuentran las definiciones de las funciones que se encargan del dibujo en la pantalla de la aplicación de usuario. Las primeras se encargan de dibujar los menús, de refrescarlos, y de dibujar los grupos y canales con sus respectivos datos. Para entender el uso de cada función ha de adelantarse que la pantalla se ha dividido en tres zonas de arriba a abajo: menú 1, menú 2, área de representación de grupos y canales. Estas zonas son por similitud las presentes en la aplicación de MS-DOS. Las funciones que se han definido se orientan para el dibujo en una de estas tres zonas. También hay otras funciones que realizan operaciones relacionadas con los menús. Todas ellas se exponen a continuación:

void Menu1(tmenu1 menú[ ]) Dibuja sobre la pantalla el nombre de cada menú tipo 1 en la posición de pantalla adecuada.

void DibujaMenu1(tmenu1 menú[ ], int k) Despliega el contenido del menú k tipo 1 si está activo. Colorea el fondo de la acción activa actual.

void Menu2(tmenu2 menú[ ]) Dibuja sobre la pantalla el nombre de cada menú tipo 2 en la posición de pantalla adecuada. Colorea el fondo del nombre del menú sobre el que pueden realizarse acciones.

void DibujaMenu2(tmenu2 menú[ ], int k) Despliega el contenido del menú k tipo 2 si está activo.

void DibujaSlideScreen(tmenu2 menú[ ], int Tpos, int Apos, int Bpos, int origenP, int numeroMuestras) Dibuja sobre el espacio de menú 2 una caja que contiene líneas verticales con la posición de los cursores y una caja interior de tamaño y posición variable según la porción de buffer dibujado y la posición del primer dato del buffer que se dibuja en pantalla.

void RellenarX(tmenu2 menú[ ], int k) Procedimiento utilizado para rellenar la palabra de búsqueda.

void TriggerWord(tmenu2 menú[ ], int k) Función que se encarga de desplegar la ventana de trigger. Permite realizar cambios en las trigger words, en la secuencia, el número de repeticiones de pulso de trigger, lógica de trigger, y demás opciones.

void CursoresPosicion(int x_origen, int y_origen, int Tpos, int Apos, int Bpos, float Ts) Escribe sobre un espacio de menú 2 la posición numérica de los cursores dentro del buffer de datos.

void ConfigModoLA( ) Dibuja la ventana de menú que permite alternar entre el modo de funcionamiento de analizador lógico y generador de patrones.

void Voltajes(tmenu2 menú[ ]) Dibuja la ventana que permite el cambio de la tensión umbral de los pods.

void ActualizarEstructurasMenu(tmenu2 menu2[ ]) Se utiliza para actualizar las estructuras de los menús a partir de las variables globales después de la carga de un fichero en el programa.

void EditarNombreFich(char ficheroSalida[ ]) Despliega la ventana que permite escribir el nombre de fichero que se salvará en disco con la información del estado de las opciones de programa actuales y los datos del último buffer de adquisición capturado.

Después de esta primera parte de fichero se encuentra la función principal, responsable del control de todo el flujo de programa. Contiene las variables locales de programa y las estructuras que contienen los datos de los menús. Después de la inicialización de variables se procede a la apertura de la ventana de la aplicación. A continuación se realiza la inicialización del analizador lógico. Si es correcta se prosigue dibujando el contenido de la ventana de programa con todos sus elementos. Una vez equipo y software se han inicializado la aplicación entra en un bucle infinito que solo abandonará si la aplicación finaliza su ejecución.

Dentro del bucle la primera tarea es muestrear las pulsaciones sobre el teclado. Si la tecla pulsada corresponde a alguna acción dentro del programa se procede a ejecutar la tarea asociada. Por ejemplo, si se pulsa la tecla *"t"* se despliega la ventana de trigger.

Una vez realizada la acción se ejecutan las funciones relativas al refresco de los menús de la pantalla, la barra de desplazamiento, y la posición de los cursores dentro del buffer de

adquisición. Todos los cambios se escriben sobre el buffer de la librería SDL. A continuación se refresca la ventana con el buffer.

Si además la acción seleccionada a ejecutar por la aplicación es realizar una captura se llama a la función *once()*, la cual es la encargada de realizar el proceso de captura de forma transparente para la función principal. Como salida devolverá los datos capturados en las variables globales de los pods declaradas en la función principal.

Finalizada la captura se procede a sobreescribir los datos sobre el buffer de pantalla dedicado de la librería SDL. En este proceso se redibuja el *grid*, se calcula el ancho de cada dato en la pantalla en función del zoom, y se pintan los grupos y los canales seleccionados. Si los cursores se encuentran dentro de la porción de datos que se muestra en pantalla se dibujan. Los nuevos datos capturados se actualizarán en la siguiente iteración del bucle infinito.

Como última instrucción del bucle se comprueba si se ha pulsado el botón izquierdo del mouse sobre el área de datos. Si es afirmativo se recalcula la posición del cursor activo sobre la posición dentro de la pantalla para que se redibuje correctamente en la siguiente iteración del bucle.

Archivo_Umbral.cpp

Contiene la función que se encarga de calcular el valor de las 11 líneas de código que portan la configuración de una tensión umbral.

ArchivoLib_patronConti.cpp

Fichero que contiene los instrucciones de configuración del generador de patrones para el analizador para el caso de método de salida continuo.

ArchivoLib_patronNT.cpp

Fichero que contiene las instrucciones de configuración del generador de patrones para el analizador para el caso de método de salida continuo sin presencia de trigger.

ArchivoLib_patronST.cpp

Fichero que contiene los instrucciones de configuración del generador de patrones para el analizador para el caso de método de salida continuo con presencia de trigger.

ArchivosC_12Lib.cpp

Este archivo contiene el procedimiento void continua1x().

ArchivosC_2xLib.cpp

Este archivo contiene los procedimiento void continua2x(), void continua2x_ST() y continua2x_NT().

ArchivosFin_ConfiLib.cpp

Contiene el procedimiento fin_confi().

Browser.h, Browser.cpp

Contienen las funciones que se encargan de crear y manejar un explorador de ficheros. Una ventana es desplegada mostrando el contenido del directorio actual de trabajo, permitiendo navegar a través de los directorios existentes y elegir el fichero de la extensión adecuada que se desea abrir.

Cnc_com.h, Cnc_com.cpp

Proporcionan acceso a los puertos serie del PC protegidos por el S.O.

Cnc_lpt.h, Cnc_lpt.cpp

Proporcionan acceso a los puertos de la impresora protegidos por el S.O. Incluyen además procedimientos para averiguar la velocidad de la CPU del PC y usar esperas de tiempo sin consumir ciclos de la CPU.

Define.h

Se declaran todas las constantes que son utilizadas en el proyecto. Son constantes tales como los colores de fondo y de los menús, el ancho de la ventana gráfica, el puerto por el que se comunica el PC con el LA, o distancias prefijadas de dibujo de los menús.

Son declaradas las estructuras de tipo de menú 1 y menú 2 para que estén disponibles en todos los ficheros del proyecto.

También se encuentra la declaración de las estructuras referentes a los grupos y a los canales. Almacenan información a cerca de si un canal o grupo se dibuja, el color del canal o grupo, o el nombre.

El fichero Define.h se incluye en todos los ficheros del proyecto en los que se precise tener disponibles las constantes y las estructuras definidas.

Dibuja.h, Dibuja.cpp

Agrupan las funciones encargadas del dibujo en el área de datos en la pantalla. Las funciones realizan las tareas de dibujo del grid, escribir el nombre de los grupos y los canales, dibujar los datos de los canales según las opciones de zoom y posición de la pantalla, y dibujar los cursores si su posición se encuentra dentro del área de buffer mostrado.

La función encargada de mostrar los grupos y los canales es *PintaGruposCanales*, que obtiene la información de los pods a partir de la función *DATO(int i, int canal)*. La primera tarea para dibujar un canal es extraer los datos a representar de las muestras almacenadas en los pods. La función *DATO(int i, int canal)* devuelve el estado de un canal concreto para la muestra i. Después *PintaGruposCanales* trata la información obtenida y la dibuja en el buffer de pantalla.

Una vez dibujados todos los datos que caben en la pantalla se pintan los cursores mediante la función *PintaCursores*.

Dlportio.h

Define los prototipos de funciones Win32 para el DriverLINX Port I/O. Se utilizará para poder acceder al bus ISA y al puerto paralelo LPT.

Debido a que el proyecto se ejecuta bajo un SO se necesitan privilegios para poder acceder a los puertos del sistema. Para ello se ha instalado en el SO el driver LINX PortI/O, el cual realiza la interfaz de comunicación entre el software de usuario y el SO. Reduce la escritura y lectura de un puerto del sistema a las instrucciones *outportb(int addr, int data)* e *inportb(int addr)*.

FTD2XX.H

Es el driver nativo para dispositivos USB del fabricante FTDI para el chip FT2232, el cual está integrado en el prototipo PCB de comunicación diseñado para puerto USB.

Contiene la definición de las funciones que se utilizan para controlar la comunicación entre el chip FT2232 y el PC.

MandarLib.cpp

Contiene los procedimiento y funciones inicioLA(), arranque(), esperadato(), espera2(), go1(), go2(), once(), rec_pod1(), rec_pod2(), rec_pod3(), rec_pod4() y rec_pod5().

MenuLib.h, MenuLib.cpp

Incluye los procedimientos de manejo de ficheros. Estos son el procedimiento que guarda los parámetros de interés del AL 4540 y los datos adquiridos en un fichero, y el procedimiento que se encarga de cargar un fichero almacenado mediante el procedimiento anterior.

También incluye el procedimiento que se encarga de la búsqueda de una palabra dentro del buffer de datos.

QuickCG.h, QuickCG.cpp

Contiene las funciones básicas que se encargan del dibujo sobre el buffer de la pantalla. Para ello define la estructura de colores, colores básicos, funciones básicas de pantalla, funciones no gráficas, formas 2D, conversiones de color, funciones BMP, funciones de escritura de texto sobre pantalla, y funciones de entrada de texto.

RegistroLib.h

Es el fichero de cabecera que incluye todas aquéllas funciones relativas a la captura de datos presentes en los ficheros dedicados a tal fin.

TriggerLib.cpp

Es el archivo que permite utilizar el trigger del analizador lógico.

Comprueba la existencia de la configuración de trigger en forma de trigger words y secuencia en el programa principal, y envía la configuración adecuada al analizador para cada cada caso de trigger.

Tx.h, Tx.cpp

Es la librería de funciones encargada de la transmisión y recepción del protocolo a bajo nivel del PC con el analizador.

Se ha diseñado para permitir tres modos de comunicación del PC con el LA, cada uno de ellos utilizando el hardware apropiado. Los modos son: bus ISA, puerto paralelo, y puerto USB. Para seleccionar el protocolo de comunicación deseado se modifica la constante *PUERTO* definida para tal caso en el fichero Define.h

3.4.3. Aplicación software

La organización de la pantalla del programa principal se dispone de la forma que se muestra en la figura 3.31. Se ha realizado con cierta similitud respecto a la aplicación `MAIN.EXE` que se halla descrita en la guia de usuario del analizador lógico.

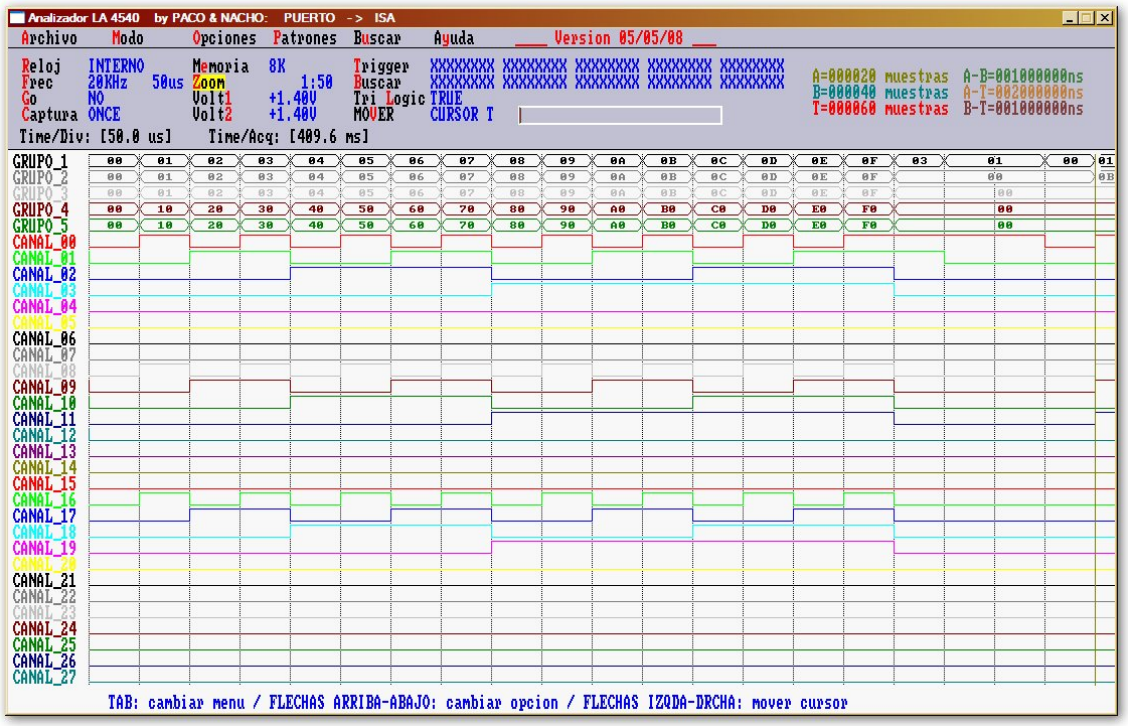


Figura 3.31: Ventana de aplicación

Las letras resaltadas se utilizan para seleccionar las correspondientes opciones. Son las denominadas "hotkeys". La pulsación de las teclas calientes desencadena inmediatamente una acción en el programa principal. Para un mejor conocimiento de de las teclas que son activas dentro de la aplicación se describen a continuación.

Hotkey	Función	Descripción
[TAB]	mover cursor	mueve la posición del cursor en la pantalla entre las diferentes opciones. También es usada en ventanas desplegables para cambiar entre opciones dentro de la pantalla.
[A]	**A**rchivo	menú dedicado a guardar ficheros, cargar ficheros, y guardar ficheros log de forma automática con las capturas.

Hotkey	Función	Descripción
[M]	**M**odo	menú que da acceso a la ventana de selección de modo de funcionamiento del analizador entre Analizador Lógico (AL) y Generador de Patrones (GP).
[O]	**O**pciones	selección múltiple del funciones incluyendo el voltaje umbral, selección de canales y/o grupos que se dibujan, y borrado de los datos existentes en los pods.
[P]	**P**atrones	permite cargar un fichero de patrones que contiene la secuencia del patrón que se repite.
[U]	B**u**scar	configuración del la palabra de búsqueda.
[Y]	**A**yuda	muestra la ventana de ayuda.
[R]	**R**eloj	selección de fuente de reloj.
[F]	**F**recuencia	selecciona la tasa interna de reloj para captura.
[G]	**G**o	habilita el modo de adquisición seleccionado.
[C]	**C**aptura	selecciona el modo de adquisición de datos.
[E]	M**e**moria	selección del tamaño de buffer de datos de adquisición.
[Z]	**Z**oom	selecciona el zoom para la representación de datos en la pantalla.
[1]	Volt**1**	configura el voltaje 1 (para los pods 1 y 2).
[2]	Volt**2**	configura el voltaje 2 (para los pods 3 y 4).
[T]	**T**rigger	despliega la ventana de configuración de trigger donde se configuran 16 trigger words desde el canal 0 hasta el 39. Cada bit puede configurarse como 1 lógico, 0 lógico, o estado lógico 'X' don´t care.
[B]	**B**uscar	edita el patrón a usar cuando se busca dentro de los datos.
[L]	Trigger **L**ogic	configura el modo de trigger true o false.
[V]	Mo**v**er	selecciona el cursor A, B, T, o la propia pantalla que puede ser desplazada dentro del buffer de datos de captura.
[F1]	Función 1	despliega el menú de ayuda.
[F2]	Función 2	acceso rápido al menú de carga de un fichero.
[F3]	Función 3	acceso rápido a la acción de guardar un fichero. El fichero queda guardado en disco con el nombre por defecto o con el nombre con el que se guardó el último fichero.
[CTRL+W]	For**w**ard	búsqueda hacia adelante dentro de los datos del patrón especificado.

Hotkey	Función	Descripción
[CTRL+S]	Reverse	búsqueda hacia atrás dentro de los datos del patrón especificado.
[↑] o [↓]	cambiar ítem	permite cambiar el cambio de ítem del menú que se encuentre seleccionado. En ventanas de submenús permite cambiar opciones o desplazar el cursor de caracter.
[←] o [→]	desplazar cursor	desplaza el cursor seleccionado en el menú Mover dentro del buffer de datos. En ventanas de submenús permite cambiar opciones o desplazar el cursor de caracter.
[MOUSE+LB]	mover cursor	cambia la posición del cursor seleccionado en el menú Mover dentro del buffer de datos que se muestra en pantalla.
[ESC] o [ENTER]	salir menú	abandonar el menú activo aceptando el último cambio.
[ALT+F4]	salir	salir del programa principal.

Cuadro 3.8: Teclas activas en la aplicación

La aplicación se ha estructurado en diferentes áreas. Se pueden distinguir dos tipos de menús, cada uno de ellos con diferente comportamiento.

El menú tipo 1 o menú principal es el correspondiente a la barra de menú de la aplicación. Su finalidad principal es la de dar acceso a otras ventanas de submenús aunque también permite ejecutar algunas acciones directamente. Cada menú consiste en una ventana de pop-up que se despliega al pulsar la tecla resaltada. Las flechas del teclado izquierda y derecha permiten desplazarse entre menús, mientras que las flechas arriba y abajo seleccionan diferentes opciones en cada menú. Para ejecutar cualquier cambio u opción elegida se presiona la tecla [ENTER]. La tecla [ESC] producirá la salida de cualquier menú sin la activación de opción alguna. Sin embargo, si se ha entrado en alguna ventana de submenú los cambios permanecerán configurados al abandonar la ventana.

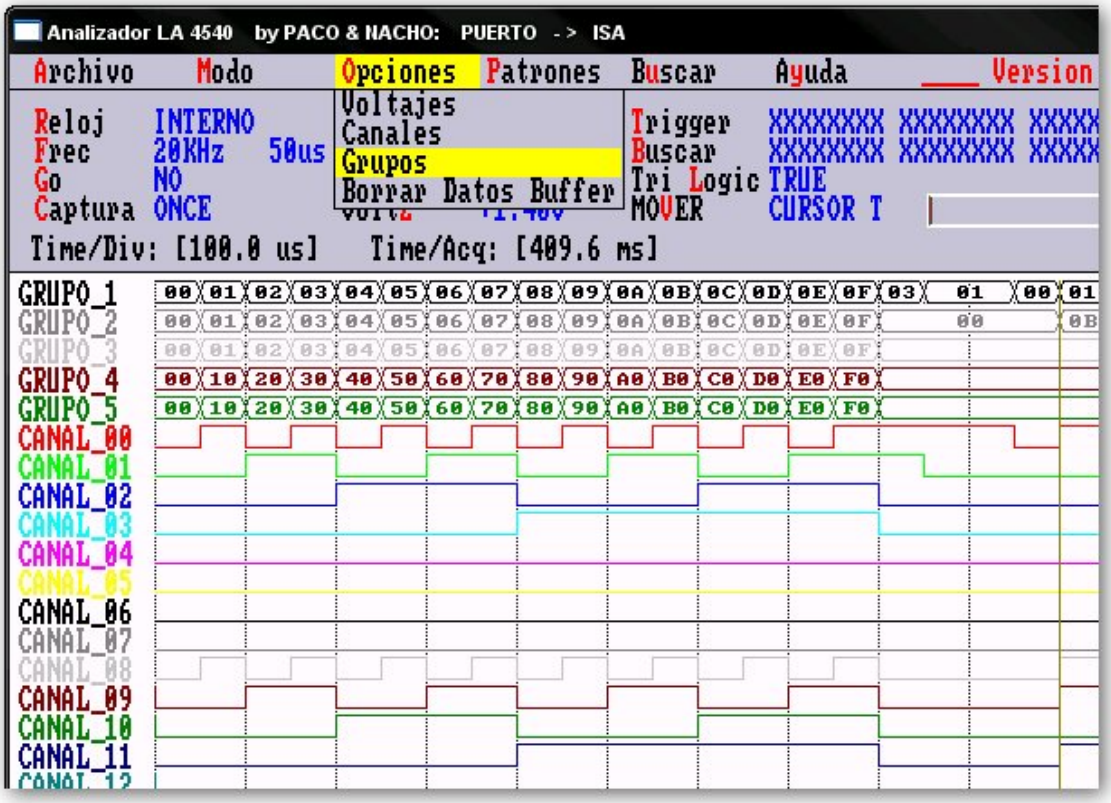


Figura 3.32: Ventana pop-up menú tipo 1

Debajo del área del menú tipo 1 se encuentra otro área de menú diferenciado. Es el menú tipo 2 o menú secundario, el cual muestra en pantalla la configuración actual de las principales opciones de funcionamiento del analizador. La pulsación de cada tecla resaltada hace bien que se despliegue una lista con las opciones de cada submenú, o bien que se despliegue una ventana de submenú. El fondo del submenú secundario seleccionado se resalta. Las flechas arriba o abajo permiten seleccionar las opciones dentro de cada lista. La tecla [TAB] permite seleccionar otro submenú activo. Las teclas [ENTER] y [ESC] confirman los cambios realizados.

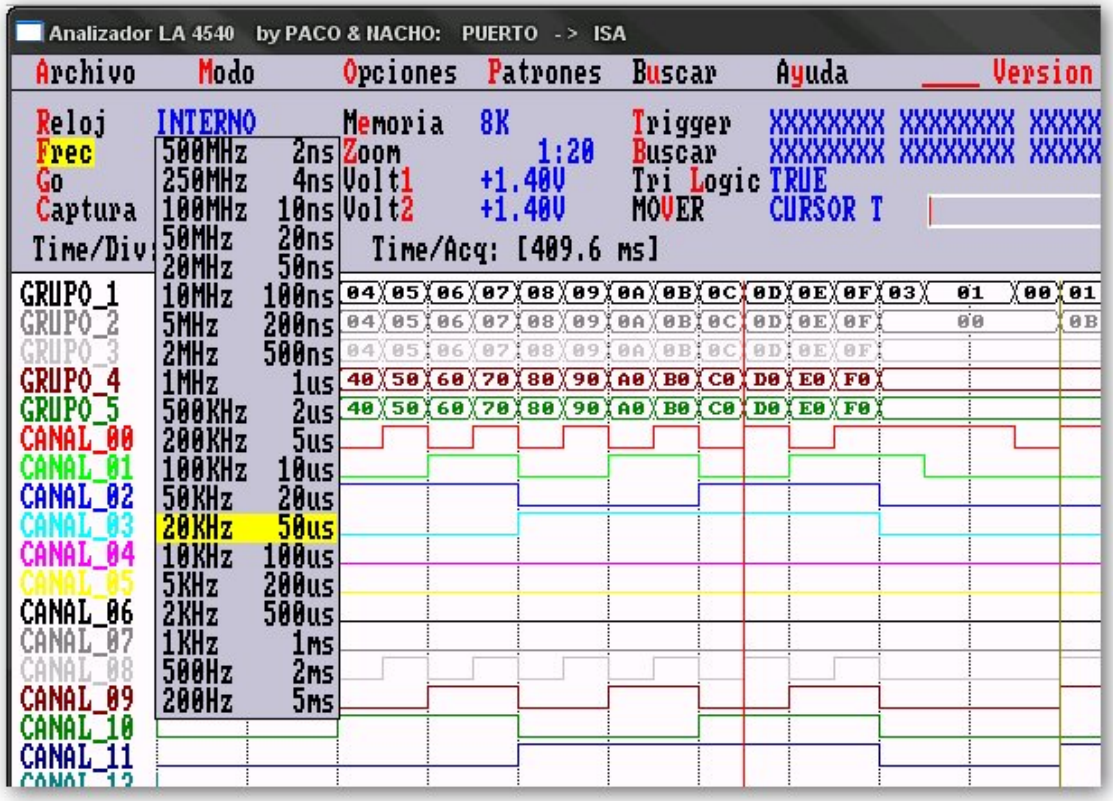


Figura 3.33: Ventana pop-up menú tipo 2

Más abajo en la parte izquierda se encuentra el área de nombre de grupos y canales. Por el momento no se ha permitido cambiar el nombre de los grupos ya que cada uno de los cinco grupos se corresponde con cada uno de los cinco pods disponibles. Sólo es posible cambiar el nombre de los canales cargando un fichero de configuración de programa previamente editado.

El área de dibujo del buffer de datos de adquisición se encuentra en la parte inferior derecha. La única acción permitida dentro del área de datos es mover la posición de los cursores dentro del buffer de adquisición. Para mover un cursor debe estar seleccionado en el menú Mover y pulsarse las flechas izquierda y derecha. Para un ajuste fino del cursor sobre una muestra concreta en la pantalla puede fijarse su posición pulsando el botón izquierdo del ratón [MOUSE+LB].

Finalmente en la parte inferior de la aplicación aparece una línea de ayuda en la que se muestran la teclas activas para cada ventana de la aplicación, ya sea la ventana principal o la ventana de cualquier submenú.

Las opciones de funcionamiento de la aplicación tienen particularidades diferentes respecto a la aplicación de MS-DOS de Link Instruments, mientras que las opciones de funcionamiento del analizador lógico son similares a las descritas en el manual de usuario. A continuación se describen las opciones de funcionamiento particulares de la aplicación.

3.4.3.1. Archivo

Cargar

Carga un fichero de datos previamente guardado. Esto incluye los datos que fueron capturados previamente y las opciones que estaban configuradas en el programa principal.

Se despliega un explorador de ficheros que permite navegar por los directorios existentes y elegir el fichero adecuado a cargar. El fichero se carga tras pulsar la tecla [ENTER].

Pulsando [ESC] finaliza la carga del fichero sin producir ninguna acción.

El formato de los ficheros debe de ser el descrito en el punto de formato de ficheros.

Figura 3.34: Ventana de Explorador de ficheros

Guardar

Guarda un fichero con el último nombre introducido en la carpeta de proyecto. Esto incluye los datos capturados y las opciones de programa configuradas.

Si no se ha introducido un nombre previamente se guarda con el nombre por defecto `configuracion.txt` especificado en el fichero `main.cpp` del proyecto.

En caso de que ya se hubiese guardado un fichero previamente mediante el submenú *Guardar como* el fichero se sobreescribe con el último nombre guardado.

El fichero se guarda por defecto al pulsar la tecla [ENTER] en la carpeta de programa salvo que se haya cargado previamente un fichero desde otra ubicación. En ese caso se guarda el fichero en dicha ubicación.

Guardar como

Guarda un fichero permitiendo especificar su nombre. Esto incluye los datos capturados y las opciones de programa configuradas.

Al seleccionar la opción aparece una ventana en la que se introduce el nombre de fichero. El fichero queda almacenado tras pulsar las teclas [ENTER] o [ESC].

El fichero se guarda por defecto en la carpeta de programa salvo que se haya cargado previamente un fichero desde otra ubicación. En ese caso se guarda el fichero en dicha ubicación.

Data Log

Activa el guardado automático de ficheros log. Los ficheros se guardan con extensión `txt` y con nombre numérico de seis dígitos que se va incrementando. El primer fichero es el `000000.txt`.

La opción se activa tras pulsar la tecla [ENTER].

Reset Data Log

Realiza un reset sobre el nombre de fichero log. El siguiente fichero log que se guarda es el `000000.txt`.

Salir

Provoca el abandono de la aplicación.

3.4.3.2. Modo

Esta función se usa para seleccionar cuales entradas del pod son usadas como analizador lógico y cuales son usadas como generador de patrones. El uso del modo generador de patrones requiere conectar el pod generador de patrones opcional.

Actualmente solo se ha desarrollado el funcionamiento de generador de patrones para los 16 primeros canales del instrumento. Se permite sin embargo seleccionar el método de salida.

El modo de funcionamiento del generador de patrones se haya descrito en la guía de usuario.

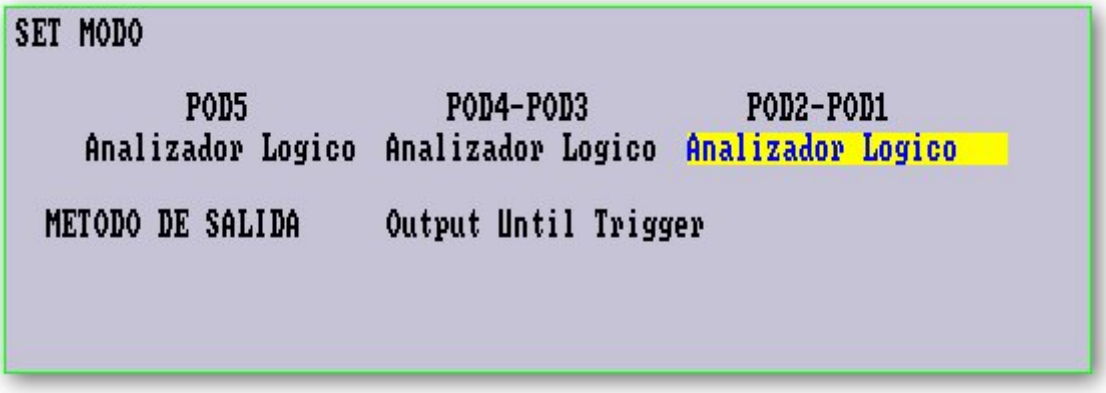


Figura 3.35: Ventana de Modo

3.4.3.3. Opciones

Voltajes

Permite acceder a la ventana de configuración de las tensiones umbrales, que pueden variar entre 6,40 voltios y -6,40 voltios (según el programa suministrado para Windows del fabricante).

Las tensiones se configuran tal y como se describe en la guía de usuario, en pods agrupados de la forma POD1-2, POD3-4, y POD5. Esta agrupación da lugar a las tensiones umbrales llamadas Volt1, Volt2, y Volt3.

Pulsando la tecla [TAB] se permite cambiar entre tensiones umbrales, mientras que su valor se modifica pulsando las flechas arriba y abajo.

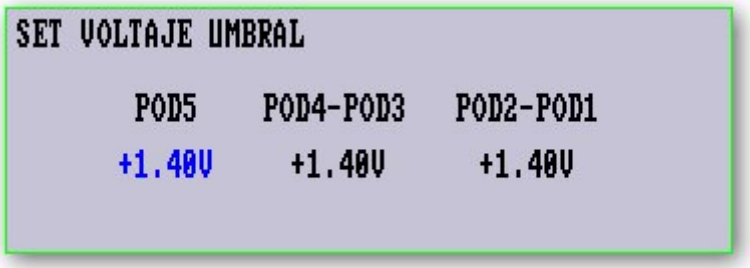


Figura 3.36: Ventana de Voltajes

Canales

La ventana de canales permite seleccionar qué canales son visibles en el área de dibujo de los datos.

Una vez se accede a la ventana la tecla [TAB] permite cambiar entre los diferentes pods. Las flechas arriba y abajo permiten seleccionar el canal, mientras que las flechas izquierda y derecha activan o desactivan el dibujo del canal.

Figura 3.37: Ventana de Canales

Grupos

La ventana de grupos permite seleccionar aquellos grupos que se visualizan en pantalla.

Para mejorar la visualización de las señales se ha permitido de forma sencilla que las señales se agrupen mediante pods. De este modo, existen 5 grupos de señales, uno por cada pod.

En la ventana de grupos las flechas arriba y abajo permiten seleccionar el grupo, mientras que las flechas izquierda y derecha activan o desactivan el dibujo del grupo.

Figura 3.38: Ventana de Grupos

Borrar Datos Buffer

Borra el área de datos de la pantalla y provoca la pérdida de los datos adquiridos en la memoria del programa.

3.4.3.4. Patrones

Permite cargar un patrón de repetición para generar señales de frecuencia conocida. El patrón se encuentra definido en un fichero de patrones que se carga en la memoria de programa y se visualiza en pantalla. El formato del fichero de patrones se describe más adelante.

El funcionamiento de esta opción en el instrumento es el descrito en la guía de usuario. Para nuestra aplicación se contempla el modo generador de patrones con un funcionamiento limitado: sólo funcionan como generador de patrones los 16 primeros canales. Además, la longitud del patrón que se repite es como máximo de 8K, la opción de 128K no se ha implementado. Por otro lado el rango de frecuencias para el modo generador va desde 200Hz a 100MHz.

El patrón se carga desde un fichero de formato adecuado y se almacena en un buffer de memoria. Si el tamaño del patrón es inferior al tamaño de la memoria, el buffer de memoria se rellena con ceros lógicos hasta completarse el tamaño total de 8K muestras. Cuando el analizador funciona en forma de generador de patrones se repiten bloques completos de memoria de 8K, por lo que es responsabilidad del usuario decidir si el fichero de patrones configura el buffer de memoria parcialmente o en su totalidad.

El método de salida determina la longitud de tiempo de la salida del patrón. Se puede configurar conforme se especifica en la guía de usuario. OUT UNTIL TRIGGER repite el

patrón de salida hasta que la condición de trigger se produce. CONTINUE repite el patrón continuamente hasta que se para manualmente por el usuario. ONCE produce la salida del buffer de patrón una única vez, desde el comienzo del buffer hasta el final y después se para.

3.4.3.5. Buscar

La función buscar permite localizar la existencia de un patrón binario entre los datos. La búsqueda se realiza de forma relativa a la posición del cursor T dentro de la pantalla.

El patrón se introduce en la pantalla de forma binaria con tres posibles estados para cada uno del los 40 canales: “1”, “0”, y ”X” Don‘t care. La búsqueda se produce de forma automática hacia la derecha tras aceptar un cambio en la palabra de búsqueda bien con la tecla [ENTER] o [ESC]. Para desplazarse dentro de la palabra pueden usarse las flechas izquierda y derecha.

Tras una búsqueda exitosa el cursor T se desplaza hasta la posición en la que se halla el siguiente dato que coincide con la palabra de búsqueda. Si no hubo coincidencia, el cursor permanece en la posición actual.

Los sentidos de búsqueda a partir del cursor T son hacia adelante o *Forward,* o hacia atrás o *Reverse*.

3.4.3.6. Ayuda

El menú de ayuda general describe de forma breve la forma de usar los menús de la pantalla, así como la función específica de las principales teclas hotkeys del programa necesarias para cada menú.

La ayuda puede ser desplegada desde este menú o bien pulsando la tecla de función [F1].

3.4.3.7. Trigger

El patrón de trigger es usado para sincronizar la captura de datos con una porción particular del flujo de datos capturados provenientes del circuito de test. La ventana de trigger permite la configuración de las palabras de trigger, la secuencia de repetición de las palabras, el ancho de pulso de trigger, las opciones de lógica de trigger, y el contador de ocurrencias de trigger. Para cambiar entre las opciones de configuración se pulsa la tecla [TAB].

Para rellenar las palabras de trigger se utilizan los tres símbolos lógicos conocidos “1”, “0”, y ”X” Don‘t care. Las fechas de desplazamiento del teclado permiten avanzar rápidamente tanto entre las palabras de trigger como entre las posiciones de cada palabra.

La secuencia de trigger se rellena mediante los símbolos lógicos “1”, “0”, y ”X” Don‘t care. Para avanzar por la secuencia se pulsan las flechas izquierda y derecha.

El ancho de pulso de trigger se configurando utilizando las flechas arriba y abajo. Para utilizar esta característica debe combinarse con la configuración del contador de pulsos y del trigger. Las posibles configuraciones de ancho de pulso son: *n veces*, *<= n clock*, *>= n clock*.

La opción de contador de ocurrencias de trigger se utiliza para la última palabra de la secuencia de trigger. El contador permite cambiarse desde 1 hasta 250 ocurrencias. Para cambiar el valor de esta opción se usan las flechas del teclado. Se puede cambiar de dígito con las flechas izquierda y derecha. Las flechas arriba y abajo cambian el valor del dígito seleccionado.

La lógica del trigger puede ser configurada también como verdadera o falsa. Las flechas arriba y abajo permiten cambiar entre lógica *true* o lógica *false*.

Para una mayor información consultar la guía de usuario proporcionada por el fabricante del instrumento.

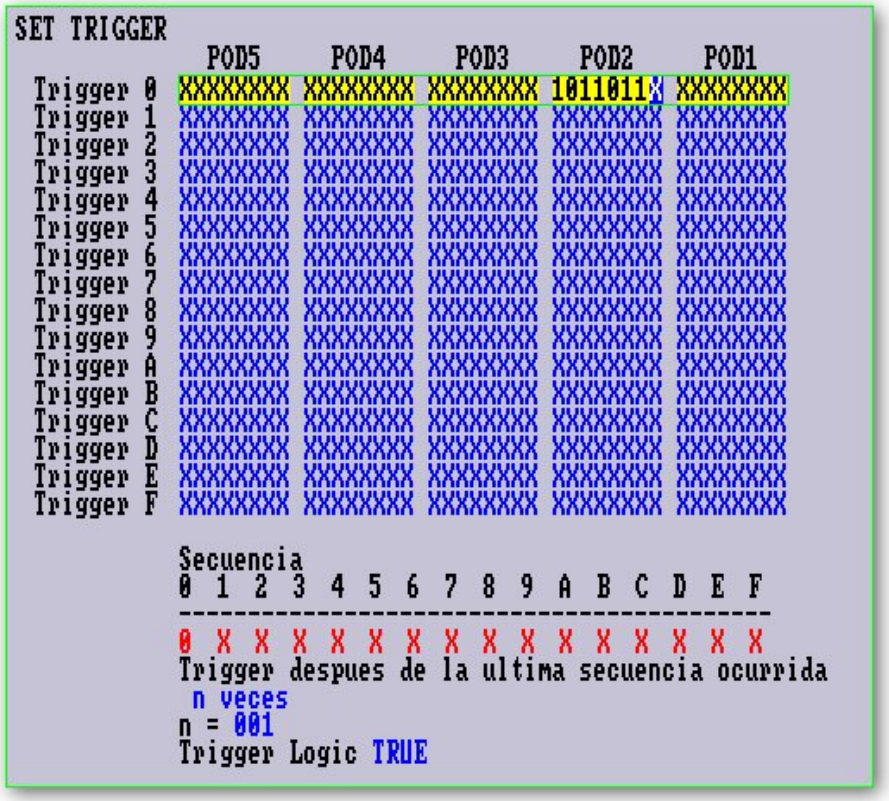


Figura 3.39: Ventana de Trigger

3.4.3.8. Formato de ficheros de programa

Fichero de datos

Los ficheros de datos son creado por la propia aplicación y salvaguardan tanto las opciones de configuración del programa como los datos almacenados en el buffer de programa actual.

Por facilidad del manejo los ficheros de datos se salvan en formato txt.

El fichero de datos se compone de una marca inicial de fichero que indica que el fichero es apropiado para ser cargado. La marca inicial con la que comienza el fichero debe ser la palabra *Analizador*.

Después aparecen los valores de opciones de configuración de programa precedidas por una cabecera que indica de qué variables se trata. Por ejemplo, la configuración de la frecuencia puede aparecer como *[Frecuencia] 13*, que indica que el valor de frecuencia de la última captura es 20 KHz.

Tras las cabeceras de opciones de programa aparecen las cabeceras de datos. En ellas se indica el pod y la memoria utilizada en la captura de la forma *[pod 1 8192 muestras por POD].* A continuación de la cabecera se almacenan los datos correspondientes a la última captura realizada a través del pod.

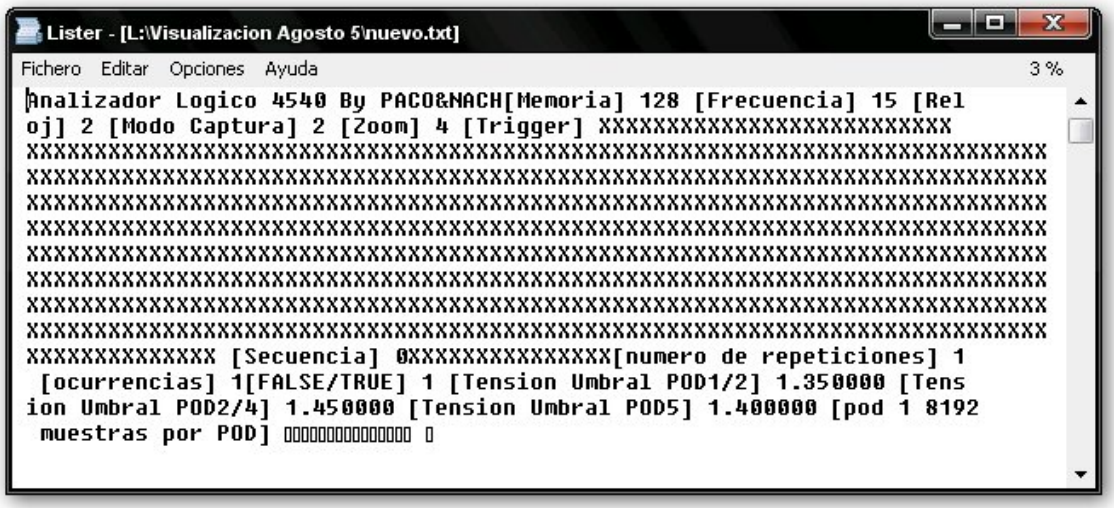


Figura 3.40: Fichero de datos

Durante la carga del fichero se comprueba que el fichero contiene la marca adecuada. Si es afirmativo se procede a cargar el fichero. Posteriormente se procede buscando la cabecera de la siguiente opción a cargar y se lee el valor. El valor se almacena en la variable adecuada de la aplicación y se repite el proceso para todas las cabeceras.

Fichero de patrones

El fichero de patrones debe ser editado de forma externa a la aplicación antes de ser cargado. Se ha escogido que su extensión sea txt por su facilidad de uso.

El archivo consta en la primera linea de la marca de fichero *PatronesLA*, que asegura que el contenido del fichero correspondrá a un fichero de patrones. Si la marca coincide se procede a la carga del fichero.

En la segunda línea le sucede la marca del primer pod *[POD1]*. Esta marca delimita el comienzo de los datos del patrón del primer pod. El patrón a cargar se almacena en pares de caracteres escritos en sistema hexadecimal, 8 bits correspondientes a los 8 canales disponibles en el primer pod. Para marcar el fin del patrón se introduce un asterisco '*' en lugar de un par de caracteres.

En una nueva línea se introduce la marca del segundo pod *[POD2]*, tras la que se almacena el contenido del segundo patrón de la forma descrita anteriormente.

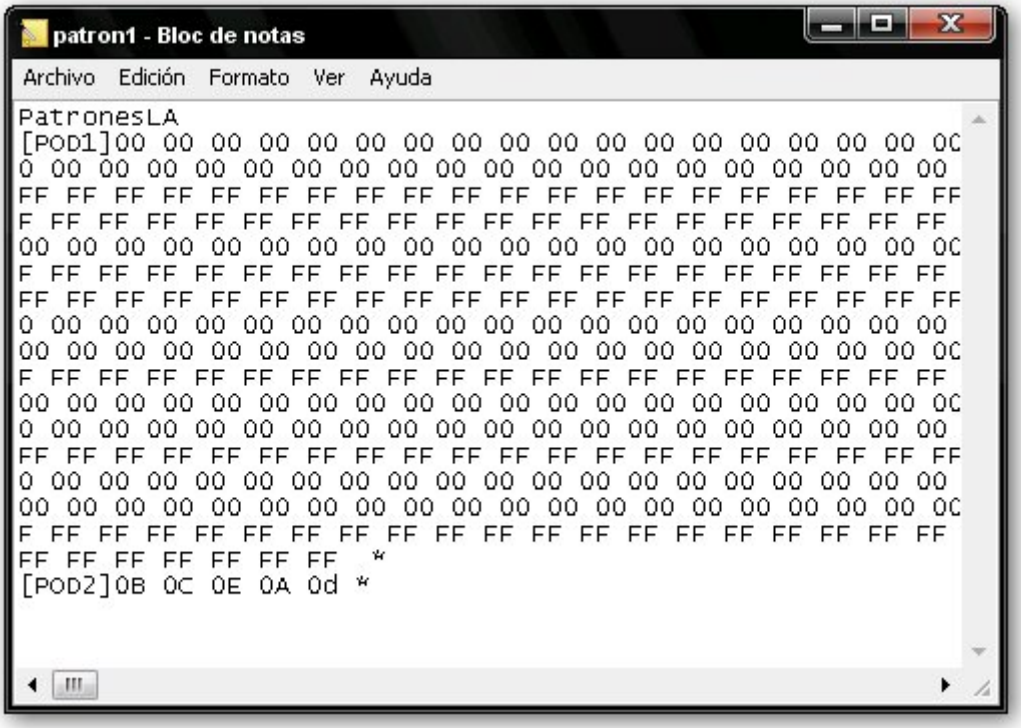


Figura 3.41: Fichero de patrones

Como se ha expuesto anteriormente el funcionamiento del analizador como generador de patrones se ha limitado para usar únicamente los pods 1 y 2, por lo que el fichero sólo contiene

los patrones para dichos pods.

La longitud del patrón será como máximo de 8K muestras. Siendo cada muestra compuesta por 8bits en formato hexadecimal como se muestra en la figura 3.41. Si el patrón a cargar tiene un tamaño inferior a la memoria que se usa en el programa principal se rellena con ceros por defecto hasta alcanzar la longitud del buffer de la memoria, el usuario debe indicar mediante la marca '*' el final de datos del pod. Es responsabilidad del usuario realizar un fichero de patrón con la cantidad de repeticiones necesarias dentro del tamaño del buffer de memoria.

4 Hardware

En la sección dedicada al hardware se aborda la tarea de crear el interface que una el PC con el analizador lógico LA-4540. Este interface tiene dos partes bien diferenciadas, por un lado una zona dedicada a potencia y por otro lado una zona dedicada a la transmisión de señales de datos.

En la parte de potencia se ha optado por usar un transformador con toma intermedia para pasar de 230v de alterna a 18v de continua. Y en la parte de transmisión de datos se han creado dos prototipos de interface, en primer lugar un interfaz paralelo y en segundo lugar un interfaz USB 2.0. Para el diseño de ambas PCB se ha utilizado el programa Proteus - Proteus ISIS y Proteus ARES - y en ambos diseños se ha utilizado la misma fuente de alimentación.

4.1. Diseño de la fuente de alimentación

4.1.1. Medida de consumos de potencia del LA-4540

Dado que el LA-4540 se alimentaba directamente a través de la tarjeta de expansión del bus ISA y ésta lo hace a través del puerto del PC, es necesario que el nuevo interfaz sea capaz de alimentar al analizador por si mismo.

El primer paso en este diseño es conocer que tensiones de alimentación son las que llegan desde la tarjeta de expansión hasta el analizador, en este caso son: 12v, 5'5v, 5v, -12v, y -5v - todas ellas tensiones continuas -.

Estas alimentaciones provienen del conector ISA del PC, y para poder ver que corriente suministra cada una de ellas se optó por hacer un corte en la pista de la tarjeta de expansión que lleva la alimentación y volver a unir ese corte poniendo entre medias un polímetro en modo amperímetro.

Siguiendo con este método para poder ver que potencia suministra el generador de 12v primero se saca la tarjeta de expansión del slot ISA, después se corta la pista B9, se sueldan dos cables - uno en cada extremo del corte - y se conectan esos cables al amperímetro. A continuación se vuelve a insertar la tarjeta de expansión en el slot ISA, se conecta el LA-4540 al conector SCSI de la tarjeta, para por último conectar el PC y ejecutar la aplicación `LA.exe`

de forma que en el amperímetro se puede observar el consumo de corriente para 12v.

Una vez visto el consumo de corriente para la fuente de 12v se pasaría a analizar el consumo de otra de las alimentaciones por ejemplo de la de -12v. Para ello una vez desconectado el PC se extrae la tarjeta de comunicaciones, se desueldan los cables que se habían soldado en la pista de 12v, se vuelve a soldar la pista perteneciente al pin B9 - comprobando que existe continuidad -, se hace un corte en la pista B7 que es la pista de -12v, se sueldan dos cables en cada extremos del corte y se prosigue como se ha comentado para el caso de 12v.

Una vez terminada la alimentación de -12v se seguiría con este mismo método para cada una de las alimentaciones.

El problema que presentó esta metodología fue al analizar el consumo de potencia de la fuente de 5v. El bus ISA proporciona la alimentación de 5v no sólo a través de un único pin sino a través de tres, a saber pines B3, B29 y D16. La tarjeta de expansión del analizador toma la tensión de los tres puntos.

En una primera medición se siguió el método antes mencionado, es decir, se cortó la pista B3 y después se midió el consumo de corriente y se volvió a soldar. En un siguiente paso se cortó la pista B29, se midió y se volvió a soldar. Y por último se hizo lo mismo con la pista D16. A continuación se sumaron las tres corrientes medidas en cada pista obteniéndose únicamente 110mA, con lo que al hacer el cálculo del transformador se optó por uno que como después ocurría era incapaz de suministrar la potencia correcta sin tener que calentarse demasiado, e incluso ocurrió la desgracia de que cuando el prototipo Interface Paralelo funcionaba correctamente y se optó por cerrar la caja que contenía la PCB, al no estar ventilada la temperatura dentro de ella aumentó de tal manera que el transformador superó sus márgenes de funcionamiento produciendo un funcionamiento anómalo que produjo la ruptura de un LA-4540.

Para solucionar el problema de la alimentación de 5v se buscó un punto en la tarjeta de expansión hacia el conector SCSI donde la corriente perteneciente a los 3 pines estuviese unida, obteniéndose una corriente de aproximadamente 580mA con lo que ahora los cálculos del transformador eran bastante diferentes, necesitando un transformador de proporcionase una mayor potencia

Los consumos presentados por el analizador lógico LA-4540 están descritos en la tabla 155:

Pin	tipo de alimentación	Corriente	Potencia
B13 B29 D16	5V	580mA	2.9W
B5	-5V	47mA	0.235W
B7	-12V	5mA	0'06W
B9	+12V	270mA	3'24W
CReductor	+5'5V	460mA	2'53W

Potencia total consumida LA	8'965W

Potencia consumida USB	0'5W

Cuadro 4.3: Consumos de corriente y potencia

A la vista de que la potencia máxima consumida por el analizador es de 8'965 W, se puede conocer la corriente máxima a la tensión de 12v siendo esta de 0'722 A y la corriente máxima a la tensión de -12V es de 0'069A ; lo que viene a decir, que si los reguladores se ponen en cascada en cada una de las ramas del rectificador y las las corrientes de retorno se suman y vuelven por la toma de masa del transformador esta suma de corrientes será la corriente máxima que debe suministrar el transformador. Concluyendo que el transformador debe suministrar como mínimo 0'747 A en total.

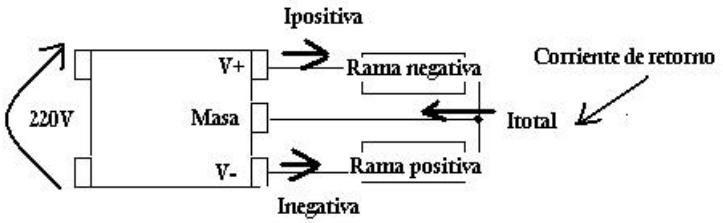


Figura 4.1: Corriente de retorno

Sin embargo, en este cálculo no se ha tenido en cuenta que en la PCB la fuente no sólo debe suministrar potencia al LA sino también a los drivers de transmisión de datos. Para el caso de transmisión paralelo los CI consumen tan poco que no hace falta su inclusión en los cálculos, pero para el caso de transmisión USB hay que sumar un consumo de potencia de 0'5W por lo que el transformador debe suministrar una corriente de 788mA, para poder suministrar

9,465W a 12v que necesita el LA.

4.1.2. Elección de los dispositivos electrónicos para la fuente de alimentación

4.1.2.1. Convertidor CA/CC

Para obtener la tensión continua desde la red alterna se optó por un transformador con toma intermedia más un puente rectificador que permitiese obtener tanto la tensión positiva como la tensión negativa.

Además, por sus características el uso de un transformador da unas ventajas e inconvenientes:

- ventajas

  Permite aislar el circuito primario del secundario - eléctricamente entre primario y secundario no existe ningún contacto eléctrico, únicamente están unidos por el flujo magnético; por ello las tierras de primario y secundario son distintas -

  Sirve como aislador o separador de la corriente alterna de la continua

- Desventajas

  Pérdidas de potencia - debido a las perdidas en vacío y pérdidas en carga - .

La configuración transformador más puente rectificador se muestra en la figura4.2.

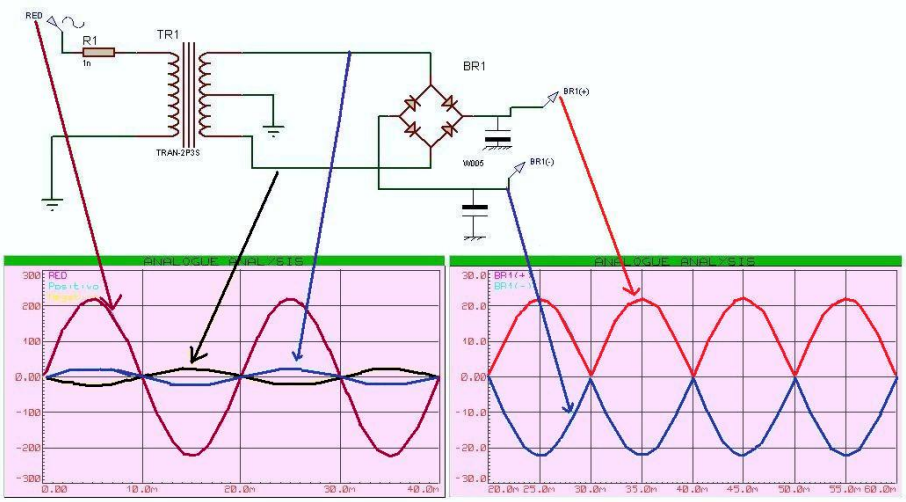


Figura 4.2: Transformador reductor

Dado que para conseguir las tensiones de funcionamiento del analizador se pensó usar reguladores lineales - y el de mayor tensión es el de 12V - la tensión en el secundario del transformador debería ser por lo menos unos tres voltios por encima del regulador de 12 voltios, más la caída de la rama del puente de diodos lo que suma unos 16'4 voltios. Al buscar un transformador que proporcionase esa tensión en el secundario se obtuvieron dos valores estándar cercanos o bien 15voltios o bien 18 voltios, finalmente se opto por el transformador de 18+18 voltios.

Una vez conocida la tensión en el secundario es necesario saber la potencia que debe suministrar dicho transformador para evitar que un mal diseño provoque un calentamiento excesivo de los devanados. Dado que la corriente máxima consumida por el analizador es de 0'788mA - incluido USB - se elige un modelo que por lo menos proporcione como mínimo esa corriente, en este caso, se eligió un transformador de 18v+18v con una corriente por el secundario de 800mA.

Una vez que ya se conoce la relación $\frac{N1}{N2} = \frac{V1}{v2} = \frac{230}{18*2}$ y la potencia a suministrar 30VA se procede a elegir un transformador comercial ,siendo elegido el modelo D2095 de la serie 30VA LOW PROFILE ENCAPSULATED TRANSFORMER de la marca DAGNALL ELECTRONICS LIMITED. En esta elección se tuvo en cuenta que el fabricante proporcionase un datasheet con las características técnicas y además que proporcionase las dimensiones de la huella del transformador para poder incluirla correctamente en la PCB mediante Proteus.

4.1.2.2. Reguladores Lineales

La solución más sencilla para poder conseguir las distintas tensiones de alimentación es la utilización de reguladores lineales. La limitación viene dada por la corriente máxima que pueden suministrar, en este caso 1A.

La disposición de los reguladores será en cascada, de forma que el de +12V suministra corriente a los reguladores de +5V y +5'5V, y el de -12V al regulador de -5V. Esta disposición se muestra en la figura 4.3. Al disponer los reguladores de esta forma hay que asegurarse de que cada regulador está suministrando la corriente dentro de sus margenes.

Regulador LM7812 Este regulador debe suministrar directamente al analizador 3'24W, e indirectamente 3'4W a través de la fuente de 5V y 2'53W a través de la fuente de 5'5V. En total 9'17W que a la tensión de 12V son 0'764 mA.

Regulador LM7805 En este caso el regulador suministra potencia al analizador, 2'9W, y al circuito encargado de la transmisión de datos - tanto en el caso Paralelo como en el caso

USB - contabilizado solo el caso del USB pues como anteriormente se ha comentado los CI necesarios para la transmisión paralelo consumen muy poco.

La potencia total a suministrar es de 3'4W que a 5V son 0'680mA.

Regulador LM713 Este integrado tiene la misma función que el convertidor reductor existente en la placa de expansión diseñada por el fabricante del analizador. El fabricante usaba como tensión de entrada 12V y como salida 5'5V, y en su diseño usó un CI que proporciona hasta 5A de corriente de salida, con posibilidad de variaciones de corriente rápidas.

Al analizar esta fuente de corriente se midió que en verdad sólo suministra 460mA lo cual está dentro de los margenes del CI elegido que en este caso es el LM713.

Regulador LM7912 Este regulador suministra la tensión negativa para el LA-4540; directamente suministra 0'06W al LA, e indirectamente 0'235W a través del LM7905. En total 0'295W que a 12v proporciona una corriente de 68mA.

En la figura 4.3 se ha optado por poner un diodo LED como indicador de funcionamiento que con la resistencia de 1K sólo consume 12mA a mayores.

Regulador LM7905 Es el encargado de suministrar otra de las tensiones negativas y la corriente a suministrar es de 47mA.

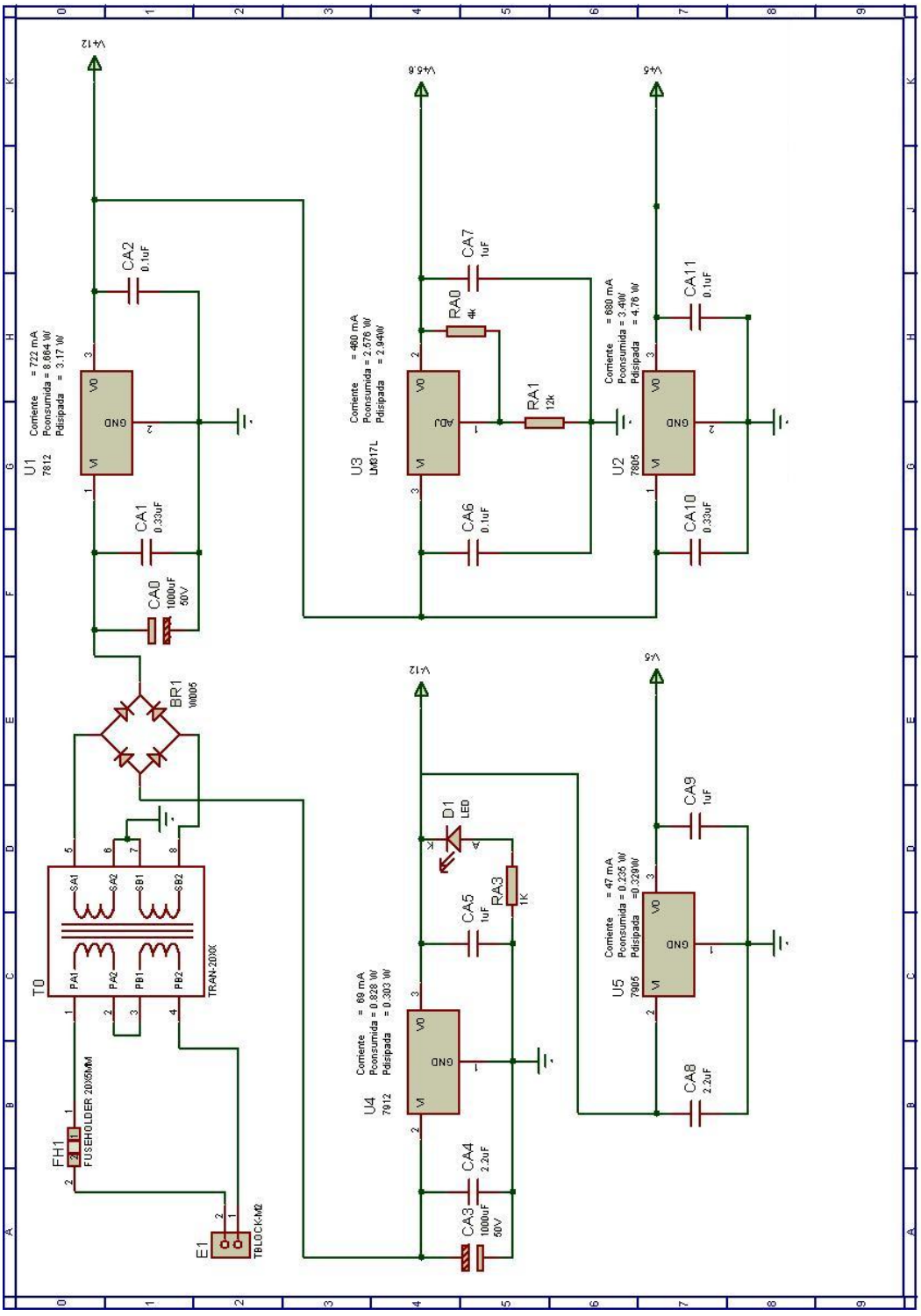


Figura 4.3: Esquema eléctrico fuente de alimentación

Tras analizar uno a uno cada regulador se comprueba que en ningún caso se va a superar la corriente de 1A por lo que con el uso de estos CI está solucionado satisfactoriamente el tema de la fuente de alimentación.

4.2. Encapsulado de los prototipos

Para el encapsulamiento de las PCB se utiliza una caja de plástico ABS con un tamaño de 193x113x62.5 mm de la marca Hammond Manufacturing en color negro.

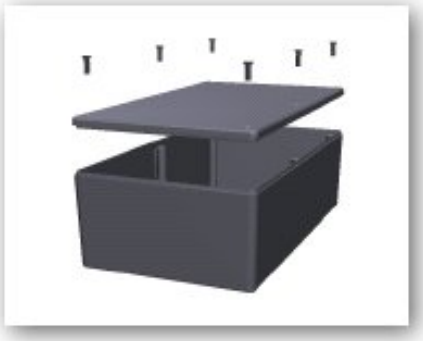


Figura 4.4: Caja de encapsulado

La elección de esta caja viene en primer lugar por sus dimensiones - su profundidad deja suficiente espacio para el transformador, hay espacio suficiente para la parte de potencia y la parte de transmisión - y por su datasheet en donde además de dar todas las especificaciones en cuanto al tamaño de la caja también se da información para incluir una PCB y la posibilidad de poder atornillarla.

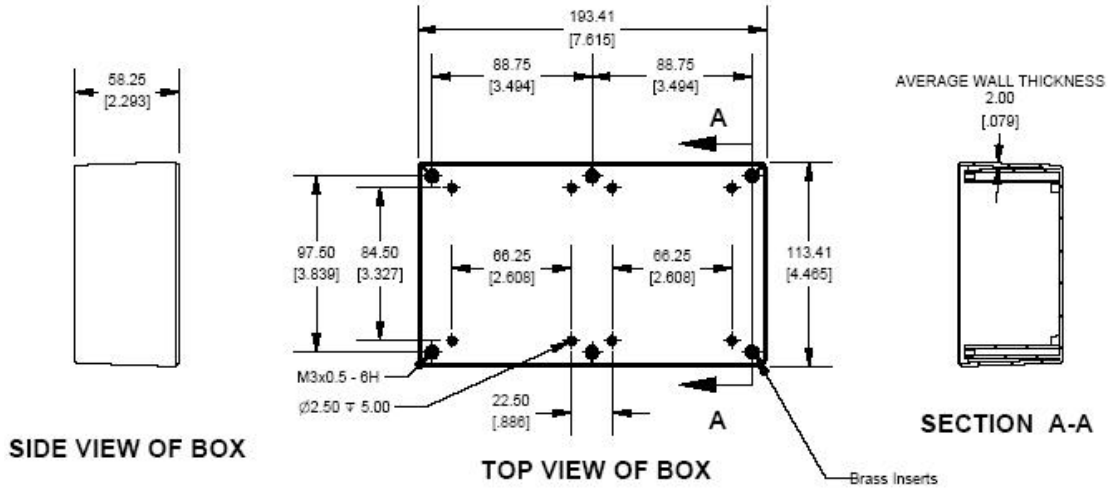


Figura 4.5: Dimensiones de la caja

Por otro lado el tener información tan detallada del tamaño de la PCB que se puede incluir dentro de la caja permitió hacer el diseño de la misma incluyendo los grids para atornillarla en el sitio exacto de forma que el proceso de crear la PCB se pudiese hacer de forma automatizada con la fresadora.

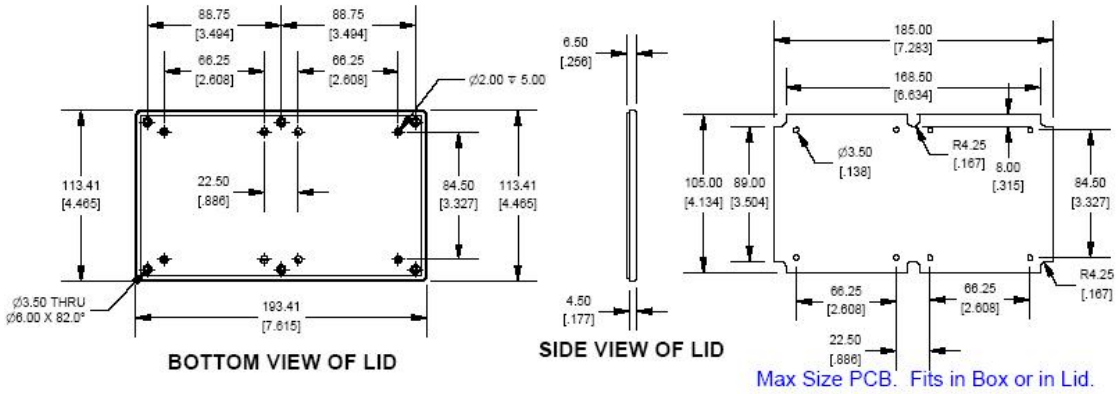


Figura 4.6: Dimensiones para la PCB dentro de la caja

En el acabado final de la caja se han introducido un interruptor de corriente y un conector IEC estándar para el cable de alimentación (similar al de un PC). La corriente eléctrica entra a la caja a través del conector IEC macho, que se secciona por medio del interruptor antes de realizar el conexionado final al "terminal block" montado sobre la PCB. Las dimensiones de los agujeros necesarios para introducir el interruptor y el conector en la caja son las que se

muestran en las siguientes figuras.

Recommended Panel Cut-Out	Thickness (mm)	Cut-out size (mm)
	1.0 (minimum)	19.3 x 13.0
	3.0 (maximum)	19.5 x 13.0

Figura 4.7: Interruptor

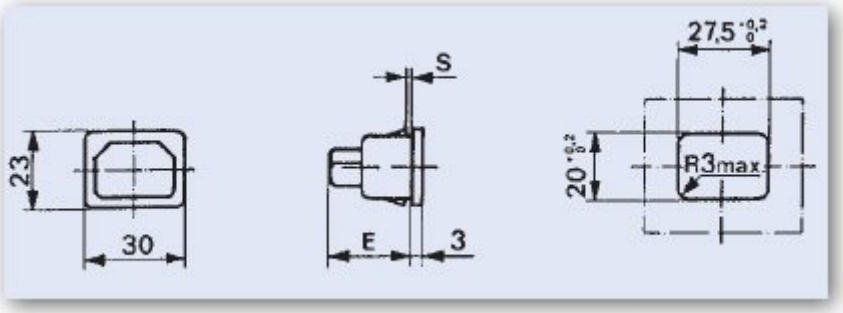


Figura 4.8: Conector caja IEC macho

Finalmente es necesario abrir en la caja los huecos necesarios para dejar libre el espacio de conexión al conector SCSI y al conector de cable paralelo o al conector USB según el prototipo de PCB. La presentación final de los prototipos se observa en la siguiente figura.

Figura 4.9: PCBs encapsuladas

4.3. Prototipo PCB para protocolo paralelo

4.3.1. Descripción del puerto paralelo

4.3.1.1. Introducción

Originalmente IBM diseñó el puerto paralelo para manejar impresoras desde su gama de microcomputadores PC/XT/AT. Un conector estándar macho de 25 pines aparecía en la parte trasera del PC con el solo propósito de servir de interfaz con la impresora. IBM desarrolló el puerto paralelo en combinación con el fabricante de impresoras Centronics, líder del mercado en el momento. Las especificaciones originales eran unidireccionales, lo cual significa que los datos solo podían ir en una dirección para cada pin con una velocidad de transferencia de entre 50 y 100 KB por segundo.

En 1987, IBM ofrece un diseño nuevo de puerto paralelo bidireccional, conocido como SPP (Standard Parallel Port), reemplazando por completo el diseño original. Las comunicaciones bidireccionales permiten a cada dispositivo recibir y transmitir datos por igual. Muchos dispositivos usan los pines del 2 al 9, originalmente diseñados para el envío de datos. Pero los pines del 18 al 25, utilizados para tierra, pueden ser usados también para datos. Esto permite una comunicación full-dúplex (ambas direcciones a la vez).

Los puertos paralelos mejorados EPP (Enhanced Parallel Port), fueron creados en 1991 por Intel, Xircom y Zenith, y permiten la transferencia de muchos más datos por segundo, entre 500 KB y 2 Mb de datos cada segundo. Fueron diseñados específicamente para dispositivos que no fueran impresoras que querían ser conectados al puerto paralelo, usualmente equipos de almacenamiento que necesitaban una mayor tasa de transferencia de datos.

Casi al mismo tiempo de la introducción de los puertos EPP, Microsoft y Hewlett Packard anuncian en conjunto una nueva especificación en 1992, llamada ECP (Extended Capabilities Port). Mientras que EPP estaba orientado a otros dispositivos, ECP fue diseñado para proveer una mejor funcionalidad y velocidad a las impresoras.

En 1994, el estándar IEEE 1284 es sacado a la luz. Incluye las especificaciones EPP y ECP. Para que ambos funcionaran correctamente, tanto el sistema operativo como el dispositivo, deben soportar estos requerimientos. Hoy en día esto no suele ser un problema ya que casi todos los ordenadores soportan todos los tipos de puertos paralelos, y detectará el modo a ser usado, dependiendo el dispositivo que este conectado. Si se desea elegir un modo de forma manual se puede hacer por medio de la BIOS.

El puerto paralelo se ha utilizado principalmente para conectar impresoras, pero también ha sido usado para programadores EPROM, escáneres, interfaces de red Ethernet a 10 MB, unidades ZIP y SuperDisk y para comunicación entre dos PCs (MS-DOS trajo en las versiones

5.0 ROM a 6.22 un programa para soportar esas transferencias). Actualmente y gracias a los nuevos modos de funcionamiento el puerto paralelo puede ser controlado por un usuario para ofrecer conectividad con otros propósitos diferentes a los originales.

El sistema operativo DOS cargado en dichos PC soporta hasta tres puertos paralelos asignados a los identificadores LPT1, LPT2 y LPT3, y cada puerto requiere tres direcciones consecutivas del espacio de E/S (entrada-salida) del procesador para seleccionar todas sus posibilidades. En la actualidad los PCs han reducido el número de puertos paralelos integrados en la placa base y la dirección base de acceso puede ser configurada en el menú de la BIOS.

Desde el punto de vista del hardware, el puerto consta de un conector hembra DB25 con doce salidas latch (poseen memoria/buffer intermedio) y cinco entradas, con ocho líneas de tierra. latch que mantiene el último valor que fue escrito hasta que se escribe un nuevo dato en el puerto. Sus características eléctricas son:

- Tensión de nivel alto: 3.3 o 5 V.
- Tensión de nivel bajo: 0 V.
- Intensidad de salida máxima: 2.6 mA.
- Intensidad de entrada máxima: 24 mA.

Desde el punto de vista del software, el puerto paralelo de las computadoras consta de tres registros (datos, estado y control) de 8 bits cada uno, que ocupan tres direcciones de E/S (I/O) consecutivas de la arquitectura x86. El registro de datos pude ser configurado como un bus de comunicación bidireccional de 8 bits de datos, mientras que los registros de control no utilizan la totalidad de los 8 bits asignados. El registro de control es bidireccional de 4 bits, con un bit de configuración que no tiene conexión al exterior. El registro de estado se trata de un registro de entrada de información de 5 bits.

La función normal del puerto consiste en transferir datos a una impresora mediante 8 líneas de salida de datos, usando las señales restantes como control de flujo. Sin embrago, puede ser usado como un puerto E/S de propósito general por cualquier dispositivo o aplicación que se ajuste a sus posibilidades de entrada/salida.

Un puerto paralelo ISA normal toma un ciclo-ISA para leer o escribir. En un sistema cuya velocidad de bus sea 1,3 Mhz, se puede decir que la lectura se puede hacer cada 1 ms (idealmente, ya que siempre existen otras instrucciones software, etc; En la práctica pueden

ser desde 1.2 ms a 2 ms). Algunos puertos soportan un modo "turbo" que elimina los 3 estados de espera de la CPU, con lo que la velocidad de lectura/escritura del puerto se duplica (2,7 MHz).

4.3.1.2. Descripción del conector físico

La conexión del puerto paralelo desde al PC a otros dispositivos se realiza mediante un conector hembra DB25. Observando el conector de frente y con la parte que tiene mayor número de pines hacia arriba, se numera de derecha a izquierda y de arriba a abajo, del 1 al 13 (arriba) y del 14 al 25 (abajo).

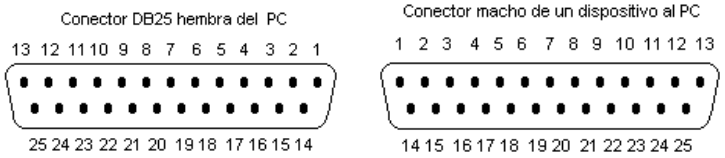


Figura 4.10: Conectores de puerto paralelo

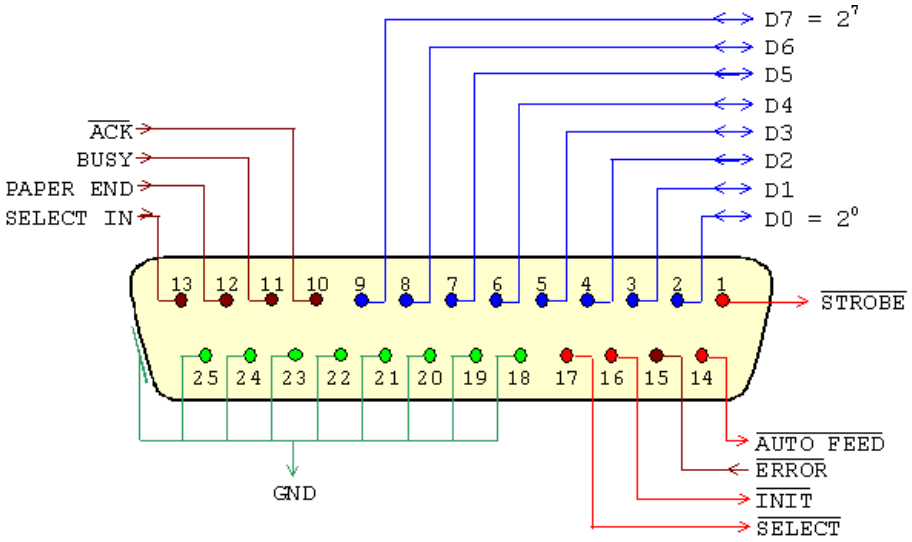


Figura 4.11: Señales en el puerto paralelo

En el conector paralelo los pines se corresponden con las siguientes líneas de señales:

- **8 líneas (pines)** son para salida de datos (bits de DATOS). Sus valores son únicamente modificables a través de software, y van del pin 2 (dato 0, D0) al pin 9 (dato 7, D7).

- **5 líneas** son de entrada de datos (bits de ESTADO), únicamente modificables a través del hardware externo. Estos pines son: 11, 10, 12, 13 y 15, del más al menos significativo.
- **4 líneas** son de control (bits de CONTROL), numerados del más significativo al menos: 17, 16, 14 y 1. Habitualmente son salidas, aunque se pueden utilizar también como entradas y, por tanto, se pueden modificar tanto por software como por hardware.
- **8 líneas** son masa, numeradas desde el pin 18 al 25.

En la tabla 4.5 se detallan la nomenclatura y descripción de cada línea. La columna "Centronics pin" se refiere a las líneas del conector tipo Centronics usado en las impresoras. La columna E/S se refiere al dato visto desde el lado del PC.

DB25	Centronics pin	Tipo (E/S)	Señal	Descripción
1	1	S	$\overline{STROBE}$	Si está bajo más de 0.5 µs, habilita a la impresora para que reciba los datos enviados.
2	2	S	D0	Bit 0 de datos, bit menos significativo (LSB)
3	3	S	D1	Bit 1 de datos
4	4	S	D2	Bit 2 de datos
5	5	S	D3	Bit 3 de datos
6	6	S	D4	Bit 4 de datos
7	7	S	D5	Bit 5 de datos
8	8	S	D6	Bit 6 de datos
9	9	S	D7	Bit 7 de datos, bit más significativo (MSB)
10	10	E	$\overline{ACK}$	Un pulso bajo de ~11µs indica que se han recibido datos en la impresora y que la misma está preparada para recibir más datos.
11	11	E	BUSY	En alto indica que la impresora está ocupada.
12	12	E	PAPER END	En alto indica que no hay papel.
13	13	E	SELECT IN	En alto para impresora seleccionada.
14	14	S	$\overline{AUTOFEED}$	Si está bajo, el papel se mueve una línea tras la impresión.
15	32	E	$\overline{ERROR}$	En bajo indica error (no hay papel, está fuera de línea, error no det.).

DB25	Centronics pin	Tipo (E/S)	Señal	Descripción
16	31	S	$\overline{INIT}$	Si se envía un pulso en bajo >50 µs la impresora se reinicia.
17	36	S	$\overline{SELECT}$	En bajo selecciona impresora (en gral. no se usa, ya que SelectIn se fija a alto).
18 - 25	19 -30, 33		GND	Masa retorno del par trenzado.
18 - 25	16			Masa lógica.
18 - 25	17			Masa chasis.

Cuadro 4.5: Nomenclatura del puerto paralelo

El nombre de cada señal corresponde a la misión que cumple cada línea con relación a la impresora, el periférico para el que fue diseñado el puerto paralelo. Las señales activas a nivel bajo aparecen con la barra de negación (por ejemplo, $\overline{Strobe}$). Cuando se indica alto o bajo se refiere a la tensión en el pin del conector. Alto equivale a ~5V en TTL y bajo a ~0V en TTL.

Centronics es el handshake estándar usado para la transferencia de datos desde un host a la mayoría las impresoras. Este handshake se implementa normalmente utilizando el modo de transferencia Standard Parallel Port (SPP) bajo el software de control control. La figura 4.12 es una simplificación del protocolo Centronics.

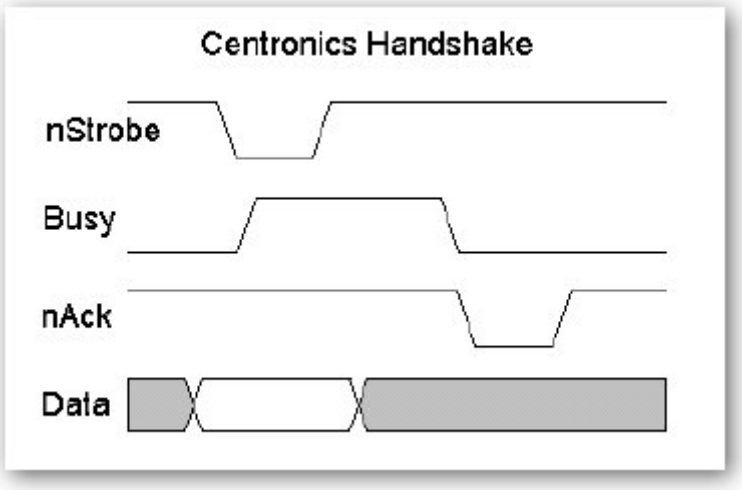


Figura 4.12: Protocolo Centronics

4.3.1.3. Acceso al puerto

El puerto paralelo se identifica por su dirección de E/S (entrada/salida, I/O) base y se reconoce en sistemas MS-DOS o Windows por el número LPT (lp en Unix/Linux). Cuando arranca la máquina, la BIOS chequea direcciones específicas de E/S en busca de puertos paralelos y construye una tabla de las direcciones halladas en la posición de memoria 40h:8h (o 0h:0408h).

Esta tabla contiene hasta tres palabras de 16 bits, cada palabra con el byte bajo primero seguido por el byte alto. Cada palabra es la dirección de E/S base del puerto paralelo. La primera corresponde a LPT1, la segunda a LPT2 y la tercera a LPT3. Las direcciones base estándar para los puertos paralelos son:

- 03BCh
- 0378h
- 0278h

La dirección base 3BCh era usada para puertos paralelos que estaban incorporados en tarjetas de vídeo. la dirección base 378h es la dirección típica del puerto LPT 1 mientras que la dirección base 278h es la dirección típica del puerto LPT 2.

La siguiente tabla muestra, como ejemplo, la memoria en un PC con dos puertos paralelo instalados en las direcciones hexadecimales 378 y 278.

Identificador DOS	Dirección	Byte alto	Byte bajo	Hexadecimal	Decimal
LPT1	0000:0408/9	03	78	378	888
LPT2	0000:040A/B	02	78	278	632

Cuadro 4.6: Asignación de memoria para puerto paralelo

4.3.1.4. Registros

El puerto paralelo estándar (SPP) consta, como se mencionó antes, de tres registros de 8 bits localizados en direcciones adyacentes del espacio de E/S del PC. Los registros se definen relativos a la dirección de E/S base (LPT_BASE) y son:

- **LPT_BASE + 0**: registro de **DATOS**
- **LPT_BASE + 1**: registro de **ESTADO**
- **LPT_BASE + 2**: registro de **CONTROL**

		REGISTROS			
		DATOS	ESTADO	CONTROL	Nombre habitual
Dirección E/S	Puerto	378h	379h	37Ah	LPT1
	Puerto	278h	279h	27Ah	LPT2
	Puerto	3BCh	3BDh	3BEh	MDA con puerto paralelo

Cuadro 4.7: Dirección de los registros del puerto paralelo

Se hará referencia a cada bit de los registros como una inicial que identifica el registro seguido de un número que identifica el número de bit, siendo 0 el LSB (bit menos significativo) y 7 el MSB (bit más significativo). Por ejemplo, D0 es el bit 0 del reg. de datos, S7 es el bit 7 del reg. de estado y C2 es el bit 2 del reg. de control.

Se indican con una barra de negación los bits que utilizan lógica negativa. En lógica positiva un 1 lógico equivale a alto (~5 V TTL) y un 0 lógico a bajo (~0 V TTL). En lógica negativa 1 equivale a bajo (~0 V) y 0 equivale a alto (~5 V).

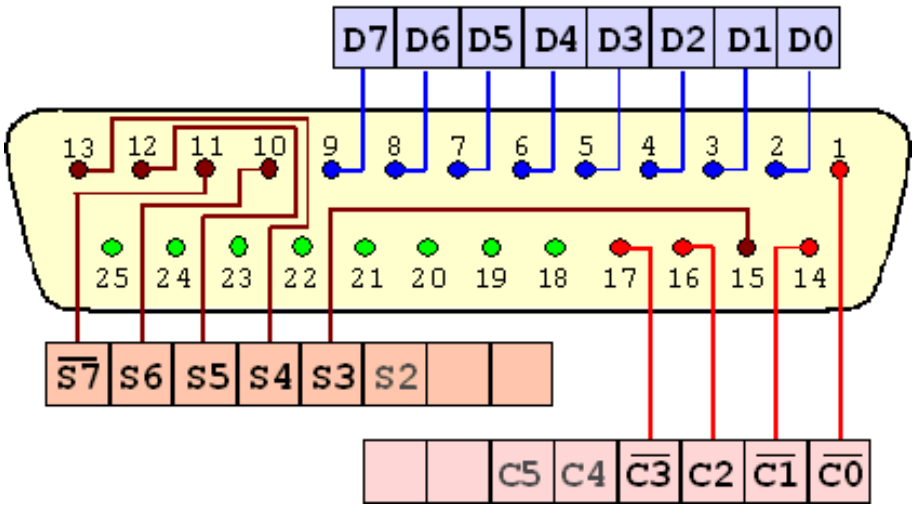


Figura 4.13: Registros del puerto paralelo

Es preciso no confundir la lógica que sigue el puerto con la lógica que mantiene la impresora. Por ejemplo, la impresora pone a alto Busy (pin 11) para indicar que está ocupada. Pero en realidad, al leer el registro de estado, Busy la interpretamos como 0 (puesto que el pin 11 se corresponde con S7). Es decir, es como si fuera activa en bajo (Busy).

Registro de datos (D)

El registro de estado se halla en LPT_BASE. Se puede leer y escribir. Escribir un dato en el registro causa que dicho dato aparezca en los pines 2 a 9 del conector del puerto. Al leer el registro, se lee el último dato escrito (NO lee el estado de los pines; para ello hay que usar un puerto bidireccional).

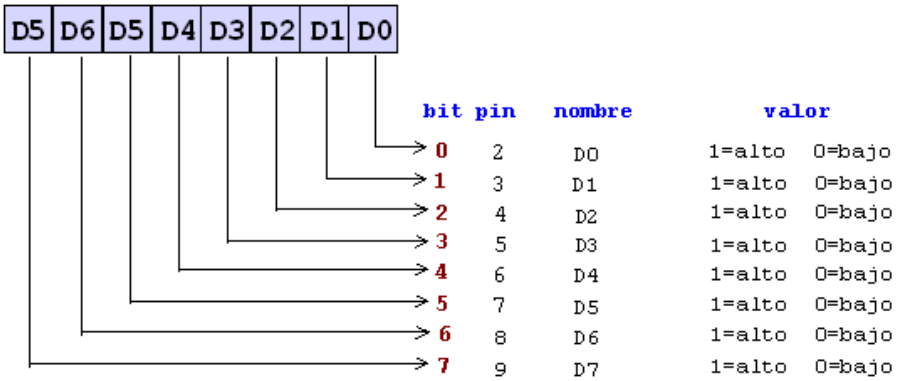


Figura 4.14: Registro de datos

El estándar es que las salidas sean LS TTL (low schottky TTL), aunque las hay que son de tipo OC (colector abierto). La corriente que pueden entregar (modo source) es de 2,6 mA máximo y pueden absorber (modo sink) un máximo de 24 mA. n el puerto original de IBM hay condensadores de 2,2 nF a masa. Las tensiones para el nivel bajo son entre 0 y 0,8V y el nivel alto entre 2,4V y 5V.

Registro de estado (S)

El registro de estado está en LPT_BASE+1. Es de sólo lectura (las escrituras serán ignoradas). La lectura da el estado de los cinco pines de entrada al momento de la lectura.

En la tabla siguiente los nombres de los pines se dejaron en inglés porque es como generalmente se identifican.

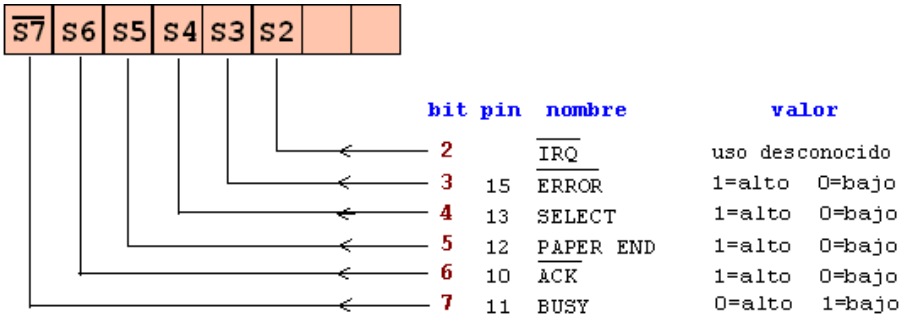


Figura 4.15: Registro de estado

La línea Busy tiene, generalmente, una resistencia de pull-up interna. El estándar es que sean entradas tipo LS TTL.

Registro de control (C)

El registro de control se encuentra en LPT_BASE+2. Es de lectura/escritura.

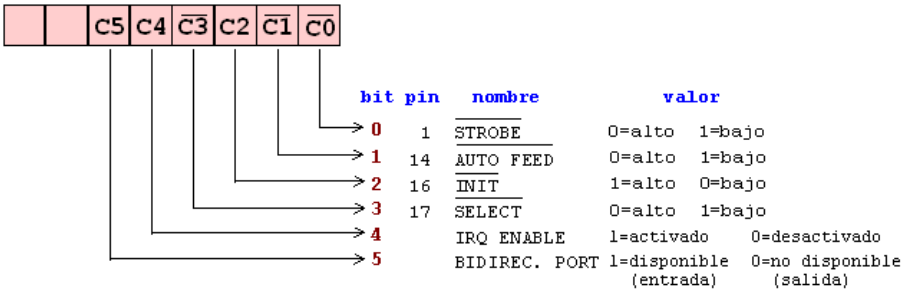


Figura 4.16: Registro de control

Los cuatro bits inferiores son salidas. La lectura devuelve lo último que se escribió a dichos bits. Son TTL a colector abierto con resistencias de pull-up de 4.7 kW, por lo que un dispositivo externo puede forzar el estado de los pines sin dañar el driver. Esto permite utilizar estas cuatro líneas como entradas. Para ello, se ponen en alto las cuatro salidas (escribiendo 0100b, es decir, 4h, en LPT_BASE+2) lo que hace que las salidas "floten". Ahora, un dispositivo externo puede forzar a bajo nivel alguna de las salidas con lo que, leyendo el puerto, sabemos si esto sucedió o no.

Es posible realizar esta técnica en salidas totem-pole (como D0-D7) pero no se recomienda su uso porque habría que tener un conocimiento preciso de la corriente, ya que se puede sobrecargar los transistores de salida y dañar el driver (especialmente en puertos integrados LSI).

Bit de puerto bidireccional (compatible PS/2) El bit C5, está disponible sólo si se trata de un puerto bidireccional; en los puertos comunes no se utiliza, al igual que los bits C6 y C7. Si C5=1, el buffer de los datos de salida se pone en alta impedancia, "desconectando" dicho buffer de los pines 2 a 9 del conector del puerto (D0 a D7). Si se escribe al registro de datos, se escribe al buffer pero no a la salida. Esto permite que al leer el puerto, se lea el estado de las entradas y no lo que hay en buffer. Cuando C5=0 el puerto retorna al modo salida, su estado por defecto.

En las computadoras IBM PS/2, para habilitar el puerto paralelo bidireccional, además de lo antes descrito, se debe poner a 1 el bit 7 del registro del puerto 102h (opciones de configuración).

En computadoras que no tengan puerto paralelo bidireccional compatible PS/2 hay que modificar uno o más bits de algún puerto específico correspondiente al chipset de la placa. A veces se habilita por el Setup o por jumper en la placa del puerto.

Bit de interrupción En trabajos normales de impresión ni la BIOS ni DOS hacen uso de la interrupción. El hecho de poseer una línea de interrupción que está conectada directamente al PIC (Programmable Interrupt Controller), lo hace muy útil para experimentación en data-loggers por ejemplo. El bit de interrupción está conectado al control de un buffer de tres estados. Cuando C4=1, se activa el buffer y su entrada, S6, se conecta a la línea IRQ (en general es IRQ7 o IRQ5). La lectura del bit, nos devuelve el estado del mismo (es decir si el buffer está en alta impedancia o no). Se producirá una interrupción, cuando haya un flanco descendente en el pin correspondiente a S6.

4.3.1.5. Modos del puerto paralelo en BIOS

Hoy en día muchos de los puerto paralelos son multimodo. Son configurables normalmente por software en una de sus modos desde la BIOS. Los modos son típicamente:

- Printer Mode (llamado modo por defecto o modo normal)
- Standard & Bi-directional (SPP) Mode
- EPP1.7 and SPP Mode
- EPP1.9 and SPP Mode
- ECP Mode
- ECP and EPP1.7 Mode

- ECP and EPP1.9 Mode

Printer Mode es el modo de funcionamiento más básico. Es el Standard Parallel Port en modo escritura desde el PC. No tiene la característica de bidireccional por lo que el bit 5 del registro de control no tiene efecto.

Standard & Bidirectional (SPP) Mode es el modo bidireccional. Usando este modo el bit 5 del registro de control hace que el puerto sea de lectura desde el PC, así que puede leerse un valor de la línea de datos.

EPP1.7 and SPP Mode es una combinación del EPP 1.7 (Enhanced Parallel Port) y el modo SPP. En este modo de operación se tiene acceso a los registros del modo SSP (Datos, Estado y Control) y a los registros del modo EPP. En este modo se es capaz de cambiar la dirección de comunicación del puerto usando el bit 5 del registro de control. EPP 1.7 es la versión más temprana del modo EPP. Esta versión 1.7 no permite disponer del bit de *time out.*

EPP1.9 y SPP Mode se comporta justo como el modo anterior, sólo que esta vez usa la versión EPP 1.9. Como en el otro modo, se tiene acceso a los registros del modo SSP, incluyendo el bit 5 del registro de control. Sin embargo, difiere del modo EPP1.7 y SPP Mode en que ahora sí se tiene acceso al bit de Timeout.

ECP Mode es el Extended Capabilities Port. El modo de este puerto puede ser configurado usando el registro de control extendido (ECR). Mediante dicho registro el puerto paralelo puede configurarse en cualquiera de los modos disponibles para puerto paralelo durante el funcionamiento normal del mismo. Sin embargo, en este modo de la BIOS el modo EPP no se encontrará disponible. Los registros del modo ECP están estandarizados bajo el *Microsoft's Extended Capabilities Port Protocol and ISA Interface Standard*, por lo que son compatibles entre fabricantes. Cuando se usa este modo una nueva configuración de registros aparece disponible partiendo de la dirección Base + 0x400h. Por tanto, el registro de control ECR estará mapeado en la dirección Base + 0x402h. Los modos seleccionables de funcionamiento del puerto paralelo aparecen en al tabla 4.8.

Bit	Descripción
7:5	Selecciona el modo de operación actual
	000 Standard Mode
	001 Byte Mode
	010 Parallel Port FIFO Mode
	011 ECP FIFO Mode
	100 EPP Mode
	101 Reserved
	110 FIFO Test Mode
	111 Configuration Mode
4	ECP Interrupt Bit
3	DMA Enable Bit
2	ECP Service Bit
1	FIFO Full
0	FIFO Empty

Cuadro 4.8: ECR - Extended Control Register

ECP y EPP1.7 Mode, y ECP y EPP1.9 Mode pueden proporcionar un puerto de capacidades extendidas justo como en el modo previo. Sin embargo, el modo EPP no estará disponible en el registro ECR. Si se está por tanto en el modo ECP y EPP1.7 Mode se tendrá un puerto EPP1.7, o si se está en el modo ECP y EPP1.9,el modo EPP1.9 Port estará a su disposición.

Los modos detallados arriba son configurables a través de la BIOS. Pueden reconfigurarse usando software propio pero esto no es recomendable ya que los registros software se encuentran típicamente en las direcciones 0x2FA, 0x3F0, 0x3F1, y se entiende que son solamente accedidos a través de la BIOS. No hay una configuración estándar de registros por lo que, si fuesen a ser utilizados esos registros, puede que el software no sea del todo portable a otras computadoras. Con los actuales S.O. multitarea no es recomendable cambiar el valor de esos registros si la BIOS conviene su valor por el usuario. De entre los modos multimodo se recomienda la configuración del modo ECP y EPP1.9 Mode.

4.3.2. Configuración del puerto paralelo

Como se expone en puntos anteriores el puerto paralelo puede funcionar en múltiples modos. Para el desarrollo del prototipo de la PCB con puerto paralelo se necesita de una comunicación bidireccional entre el PC y el analizador lógico. El PC debe tanto transmitir comandos al analizador como extraer del mismo los datos capturados.

De entre los modos de funcionamiento posibles del puerto se escoge el modo EPP ya que

permite la comunicación bidireccional utilizando la líneas del registro de datos D7-D0. Es el modo bidireccional más sencillo, y el cambio de sentido de la comunicación se realiza tan sólo controlando el bit 5 el registro de control. Si C5=1, el buffer de los datos de salida se pone en alta impedancia, permitiendo realizar lecturas sobre las líneas de datos a través del registro de datos. Cuando C5=0 el puerto retorna al modo salida, su estado por defecto, permitiendo escribir datos en las líneas correspondientes al registro de datos.

Para la utilización del modo de funcionamiento EPP éste debe configurarse previamente en el menú de la BIOS del PC. El acceso al menú de la BIOS se realiza pulsando una tecla durante los primeros instantes del arranque de la computadora, después de realizar el chequeo de los dispositivos conectados al bus del sistema y antes de proceder a la carga del S.O. A continuación se presenta la pantalla de testeo y cuenta de memoria, en la que aparece la tecla de acceso al menú de la BIOS. La tecla de acceso depende de la BIOS de cada fabricante, aunque puede decirse que típicamente es una tecla de función (F1,F2... F12) o la tecla [DEL] de suprimir.

Una vez accedido al menú debe navegarse hasta la sección de configuración avanzada de periféricos (PNP/PCI CONFIGURATION) donde se encuentra la opción de *Parallel Port Mode,* en la que se selecciona el modo EPP Mode. **Si se permite configurar la dirección base ésta debe ser 378h,** la de por defecto para el puerto LPT1. Esto se debe a que la dirección configurada en la aplicación para el manejo del analizador lógico ha sido la dirección LPT1 por defecto, que es la que aparece en la mayoría de las computadoras.

4.3.3. Desarrollo del hardware

La solución *hardware paralelo* se implementa partiendo de la fuente de alimentación diseñada, siendo la fuente de alimentación la encargada de proporcionar los niveles de tensión y voltaje necesarios por el analizador lógico.

Conocido el protocolo de comunicación con el AL es necesario implementar una simulación del protocolo ISA a través del puerto paralelo. Se trata de establecer una interfaz entre el puerto paralelo y el conector SCSI del cable del AL. Las señales que se deben transmitir se muestran en la figura 4.17 y son:

- Bus de datos de 8 bits bidireccional: D7:D0.
- Señales de direcciones: A11, A10, A3, A2, A1 A0.
- Señales de control: $\overline{SEN}$, $\overline{IOR}$,$\overline{IOW}$.

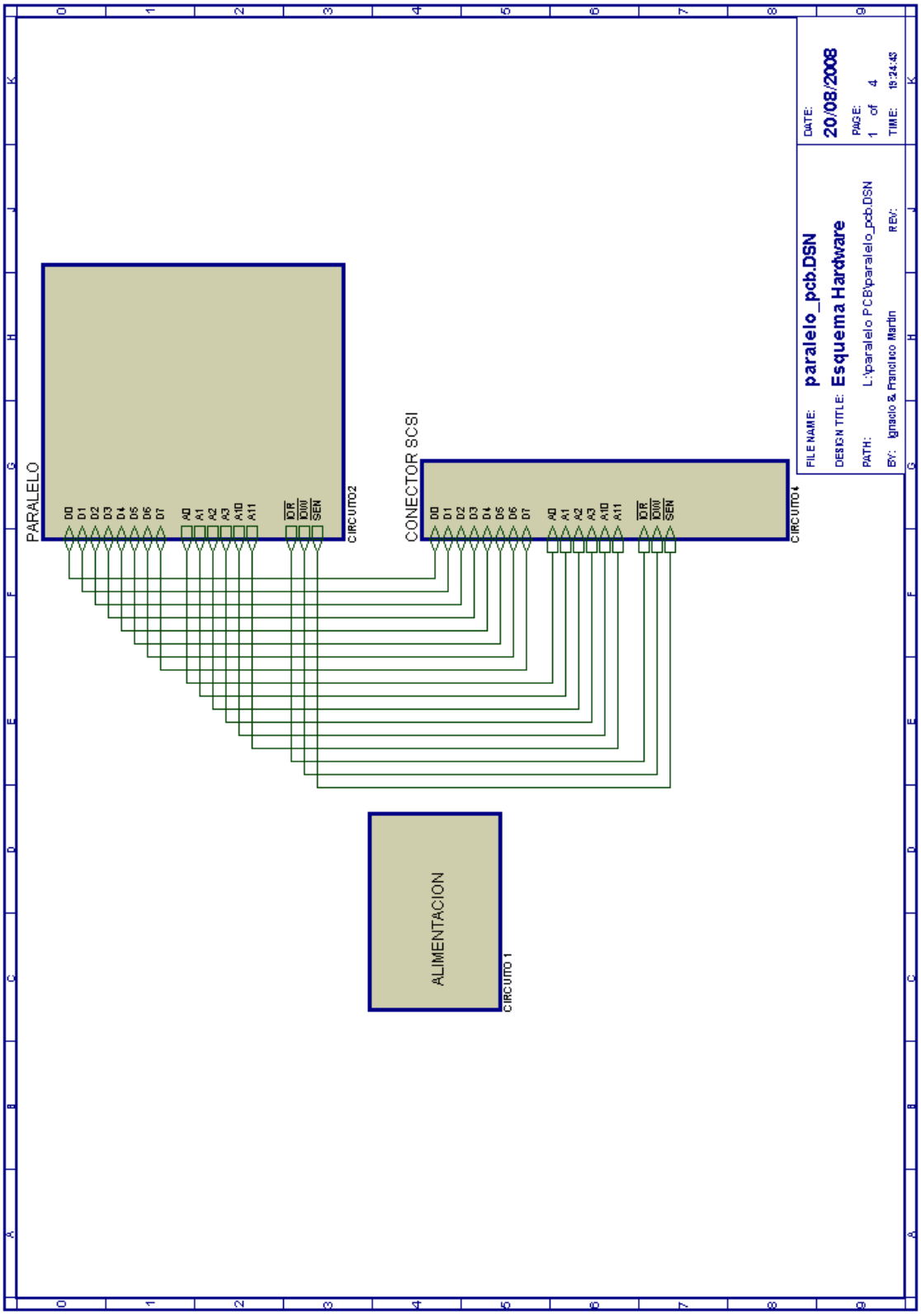


Figura 4.17: Diagrama de bloques de hardware paralelo

La solución a adoptar debe tener en cuenta que en el puerto paralelo en modo EPP tan sólo se dispone de un bus de 8 bits bidireccional para datos, así como de cuatro líneas de control correspondientes a los cuatro bits LSB del registro de control inicialmente destinados a la impresora. Estas líneas son:

- Bus de datos de 8 bits bidireccional: D7:D0.
- Líneas de control de la impresora: $\overline{STROBE}$, $\overline{AUTOFEED}$,$\overline{INIT}$,$\overline{SELECT}$.

Como no se dispone de líneas suficientes para cablear todas las necesarias en el conector SCSI se opta por utilizar el bus de datos del puerto paralelo como E/S de datos y Salida de direcciones simultaneamente. Este proceso requiere que la comunicación se realice utilizando varios ciclos de bus. En un primer paso se saca por el registro de datos el valor de las líneas de direcciones (A11, A10, A3, A2, A1 A0) y se retienen en un latch. Posteriormente se completa la simulación del ciclo ISA bien escribiendo un dato en el registro de datos (si se simula una escritura) o bien leyendo el estado del registro de datos (si se simula la escritura de una dato). Todas estas operaciones se apoyan en el manejo de las líneas de control de la impresora, convenientemente cableadas para realizar la simulación del protocolo.

Para poder explicar con más detalle el funcionamiento de la PCB paralelo es conveniente revisar las conexiones realizadas de la figura 4.18. En la tabla 4.9 se muestra la equivalencia de las líneas de control y del bus de datos del conector paralelo, el bit del registro donde se modifica su valor, así como su conexión con los drivers que posibilitan la comunicación.

Nombre de la señal	Registro	Equivalencia	Descripción
D0	$D0$	DATA0 - ADDR0	Bit 0 de datos y dirección
D1	$D1$	DATA1 - ADDR1	Bit 1 de datos y dirección
D2	$D2$	DATA2 - ADDR2	Bit 2 de datos y dirección
D3	$D3$	DATA3 - ADDR3	Bit 3 de datos y dirección
D4	$D4$	DATA4 - ADDR10	Bit 4 de datos y dirección
D5	$D5$	DATA5 - ADDR11	Bit 5 de datos y dirección
D6	$D6$	DATA6	Bit 6 de datos
D7	$D7$	DATA7	Bit 7 de datos
$\overline{STROBE}$	$\overline{C0}$	LA	Load Address: realiza la carga de las direcciones con un flanco de bajada
$\overline{AUTOFEED}$	$\overline{C0}$	$\overline{SEN}$	Señal de strobe del protocolo, habilita el driver de datos
$\overline{INIT}$	$C3$	$\overline{IOW}$	Señal de escritura
$\overline{SELECT}$	$\overline{C4}$	$\overline{IOR}$	Señal de lectura, cambia el sentido del driver de datos

Cuadro 4.9: Equivalencia de las líneas de control

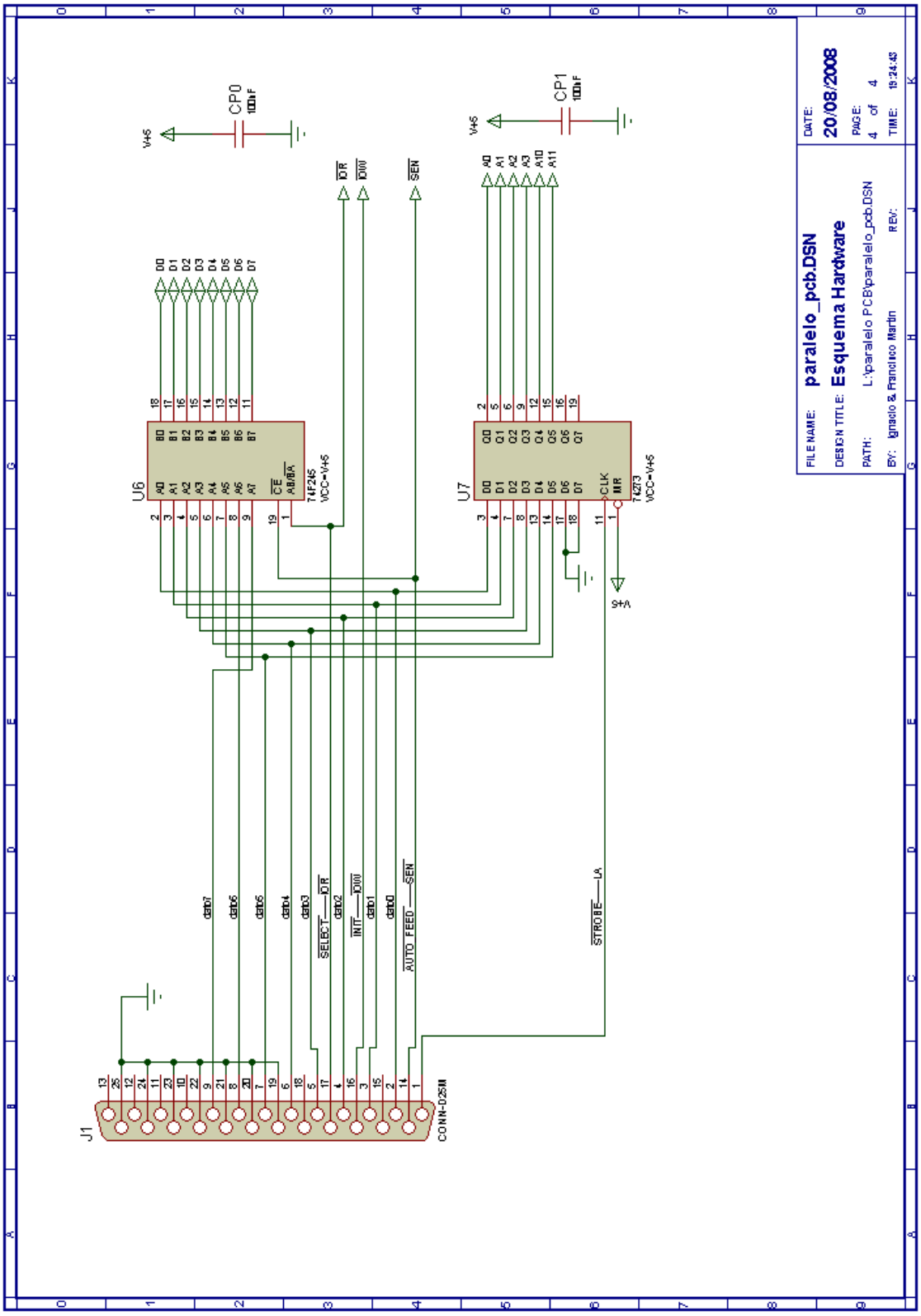


Figura 4.18: Detalle del conector paralelo

Los drivers que intervienen en la comunicación son el transceptor bidireccional de 8 bits 74F245 y el flip-flop tipo-D 74273. El transceptor 74F245 se utiliza para canalizar la transferencia de datos entre el PC y el analizador, mientras que el flip-flop 74273 se utiliza para realizar un latch de las direcciones sobre las que escribe o se lee del analizador.

El transceptor bidireccional 74F245 permite controlar el sentido del flujo de 8 bits de datos a través de su entrada Transmit/Receive ($T/\bar{R}$). Si en la señal $\overline{IOR}$ conectada a la entrada ($T/\bar{R}$) está a nivel alto, los datos se transmiten desde el puerto A hasta el B, por lo que transmite un dato del de el PC al analizador lógico. Si por el contrario ($T/\bar{R}$) está a nivel bajo, los datos pasan del puerto B al A, por lo que se lee un dato del analizador. Por último, si la entrada Output Enable ($\overline{CE}$) conectada a la señal $\overline{SEN}$ está a nivel alto se deshabilitan ambos puertos y sus salidas pasan a alta impedancia.

El flip flop 74243 permite realizar realizar un latch de las señales presentes en su entrada. Se compone de 8 flip-flops tipo-D con entradas D y salidas Q individuales. Las entradas de reloj CLK y master reset $\overline{MR}$ cargan y resetean todos los flip-flops simultaneamente. El estado de cada entrada D, un tiempo de set-up antes de una transición de reloj LOW-to-HIGH (flanco de subida), es transferida a la salida correspondiente (Qn) de cada flip-flop. Cuando la señal de reloj conectada a la línea LA realiza una transición de flanco de subida se produce la carga de las direcciones presentes en el bus de datos del puerto paralelo, cuyo valor se mantiene hasta la carga de la próxima dirección. Esto es así ya que no se permite utilizar la entrada de master reset, que se encuentra cableada a 5 voltios positivos.

En la figura 4.19 se muestra la conexión final del conector SCSI de 50 pines. Esta conexión reproduce fielmente las señales del protocolo ISA provenientes del bus de la computadora. Las señales de datos y direcciones que intervienen en el protocolo, una vez que atraviesan los drivers, son conducidas mediante pistas eléctricas a los correspondientes pines del conector SCSI. Las señales de control del protocolo ($\overline{SEN}$, $\overline{IOR}$,$\overline{IOW}$) son conducidas directamente al conector. En la figura también aparecen las líneas de alimentación necesarias para el correcto funcionamiento del analizador lógico.

En el diseño de las conexiones de las señales se han incluido unas resistencias en serie con las señales de datos, direcciones, y control. Son resistencias de bajo valor y su uso se justifica para evitar que se produzcan cortocircuitos en las líneas de transmisión.

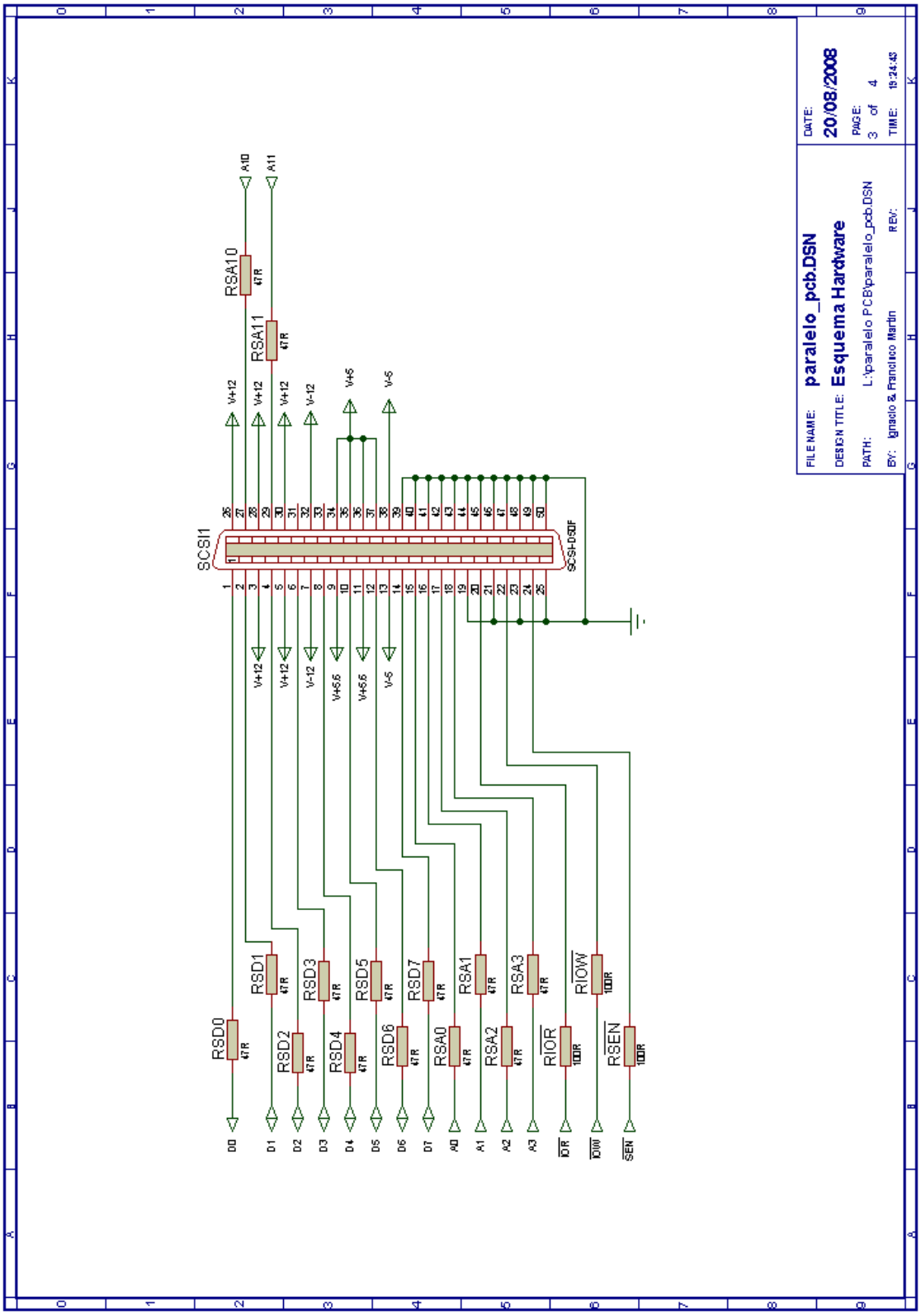


Figura 4.19: Detalle del conector SCSI

4.3.4. Programación del puerto paralelo

La solución software del puerto paralelo se realiza conforme a la solución hardware y se encuentra en el fichero `Tx.cpp` del proyecto. Como se ha dicho anteriormente el modo de acceso al puerto paralelo es EPP y se accede a la dirección 0x378.

Siendo la dirección base la 0x378, esta será la dirección del registro de DATOS, la 0x379 será la dirección del registro ESTADO, y la 0x37A la dirección del registro de CONTROL.

Al ir sacando los ficheros de funcionamiento del LA y su posterior paso a lenguaje C se usaron las instrucciones inportb y outportb donde las direcciones hacen referencia a la posición que indica la tarjeta de expansión por tanto es necesario crear una función por encima de los propios outportb e inportb que permita poder comunicarse con el LA a través de otros interface que no estén en esa dirección.

Para ello se cambian de nombre a las funciones inportb por inport e outportb por outport. Además se crea una variable tipo define llamada PUERTO que indica que protocolo se va ejecutar.

valor de PUERTO	Protocolo a ejecutar
0	ISA
1	Paralelo
2	USB

Cuadro 4.10: Tabla de protocolos

Entonces dependiendo del valor de PUERTO al compilar el ejecutable la función inport y outport se crearán de una forma u otra. Y de esta manera se permite crear una aplicación con distintos protocolos. Para poder comprender mejor estas palabras se puede consultar en el código de la aplicación el archivo `Tx.cpp`.

A la vista de la tabla 2.7 cuando se usa el bus ISA se necesita poner en el bus de direcciones cierto valores fijos - de A4 a A9 - y el resto - A0, A1, A2, A3, A10 y A11- sirven para direccionar el LA. Por tanto, una primera tarea es realizar una máscara sobre la información dirección, y así sólo pasar los 6 bits que interesan y olvidar el resto. En la figura 4.20se muestra como pasar de los 11 bits que se necesitan para poder utilizar la tarjeta de expansión a los 6 bits necesarios para utilizar los interfaces paralelo y USB.

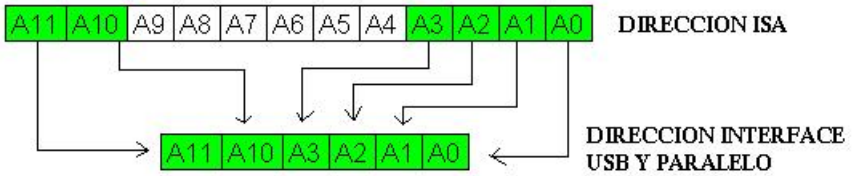


Figura 4.20: Máscara de dirección

Una vez explicado como realizar la máscara para pasar correctamente la dirección utilizando los interfaces paralelo y USB se pasa a explicar las dos posibles acciones a realizar con el interface paralelo. Se distinguen dos posibles tipos de acceso al periférico de salida, en este caso el LA-4540:

- escritura
- lectura

Escritura Para el acceso de escritura al LA se debe realizar una secuencia de eventos consistentes en en primero cargar la dirección a la que se quiere acceder del LA en el CI 74237, segundo cargar los datos a transmitir en el CI 74F245 y por último que las señales de control estén con los valores adecuados para que el LA cargue la información que se le quiere transmitir. Al final de este proceso se dejan las señales de control en alta impedancia.

Este proceso de escritura se consigue mediante 6 instrucciones, a saber:

1. outportb(LPT_CONTROL,0x05) : Dado que el 74273 realiza la carga en el flanco de subida de su pin CLK (señal $\overline{STROBE}$), lo primero es ponerla a nivel bajo para después al subirla a nivel alto conseguir un flanco de subida.

2. outportb(LPT_DATA, addr): El bus de datos del puerto paralelo se carga con la información del bus de direcciones del LA.

3. outportb(LPT_CONTROL,0x04): Ahora se sube la señal $\overline{STROBE}$ a nivel alto, con lo que la información que esta en la entrada del 74273 se carga y se mantiene en su salida. Se ha cargado la información relativa a la dirección a transmitir al LA.

4. outportb(LPT_DATA, data): Se vuelve a poner información en el bus de datos paralelo, en este caso esta información hace referencia al bus de datos del LA.

5. outportb(LPT_CONTROL, 0x02): Se dan valores a las señales de control de tal manera que reproduzcan un outport tal como lo haría el software del fabricante para conseguir un out:

   - $\overline{AUTOFEED} \equiv \overline{SEN}$ = '0'. Se está indicando al 74F245 que cargue los datos de su entrada en su salida. Y dado que $\overline{SEN}$ = '0' se el LA sabe que se le está direccionando
   - $\overline{SELECT} \equiv \overline{IOR}$ = '1'. Se está indicando al 74F245 cual es la dirección de carga.
   - $\overline{INT} \equiv \overline{IOW}$ = '0'. El LA reconoce que se le está pidiendo un proceso de escritura, $\overline{SEN} = \overline{IOW}$ = '0'.

6. outportb(LPT_CONTROL, 0x04): Se vuelven a poner las líneas de control en alta impedancia con lo que el LA deja de estar seleccionado.

En este proceso de escritura se diferencian dos etapas: en primer lugar una etapa para cargar el bus de direcciones en el 74273 y una segunda etapa para cargar y transmitir el bus de datos a través del 74F245. A la vista de la secuencia de instrucciones para hacer una escritura se puede observar que aunque se ponen en alta impedancia las señales de control el 74243 continua poniendo en su salida el bus de direcciones, y esto se puede usar para la siguiente escritura/lectura; si en la siguiente escritura/lectura se accede a la misma dirección no hace falta realizar las 3 primeras instrucciones, de esta forma se consigue que el interface Paralelo sea un poco mas eficiente en cuanto a tiempos de acceso al periférico LA-4540.

Al añadir esta mejora se pueden dar dos casos al realizar un out:

- 2 out consecutivos a direcciones distintas: 13'0us
- 2 out consecutivos a la misma dirección: 6'6us.

Estos dos casos se pueden apreciar en la figura 186 las dos primeras capturas hacen referencia al caso de hacer dos out a direcciones distintas - se puede apreciar como hay un proceso de carga de dirección en el CI 74273 - y la tercera captura hace referencia a dos out consecutivos donde la dirección no ha cambiado - con lo que se ahorra el tiempo de precarga -.

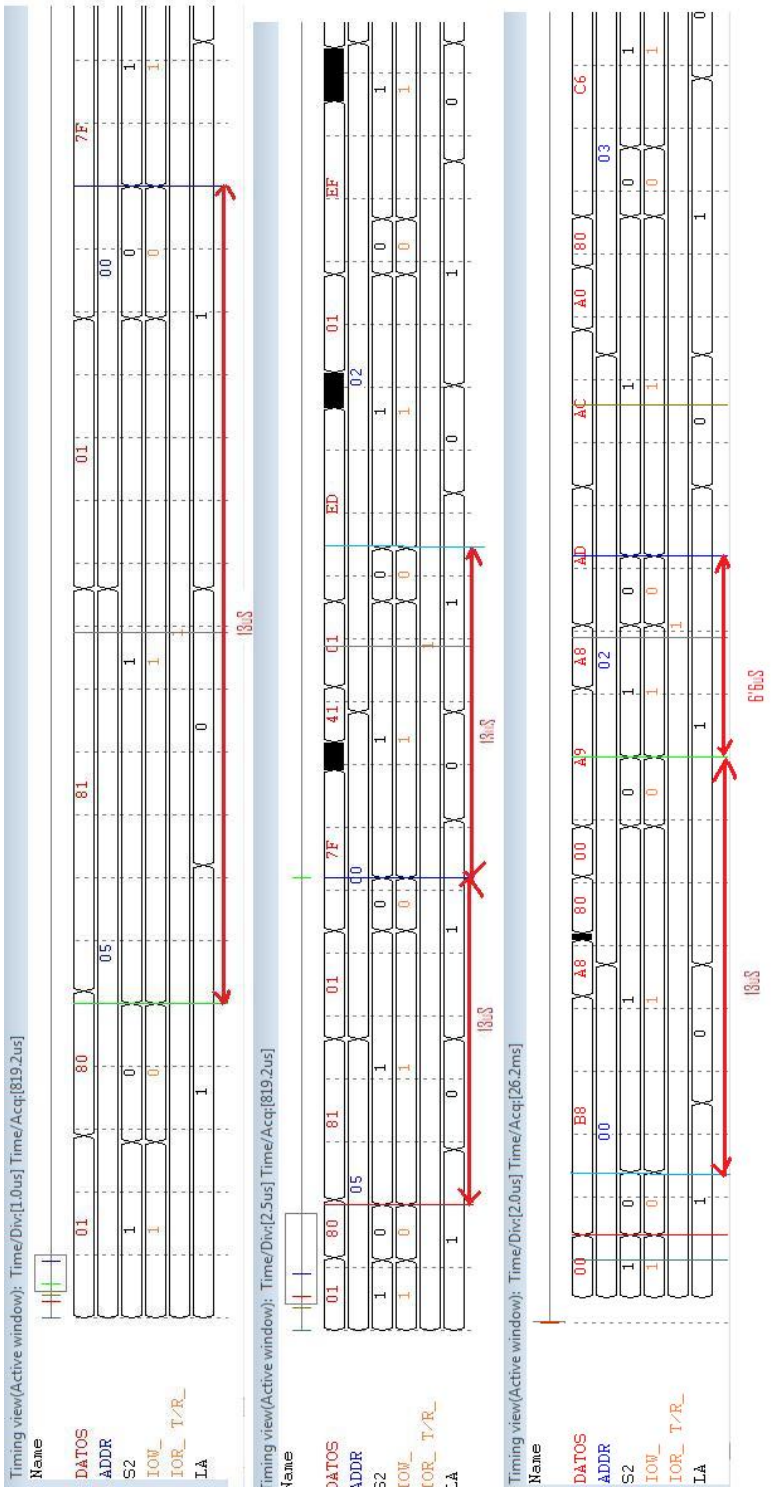


Figura 4.21: Cronograma del interface Paralelo. Caso OUT

Si se comparan estos tiempos con los necesarios al utilizar la tarjeta de expansión ISA vemos que el interfaz Paralelo es más lento que el ISA.

OUT ISA	OUT Paralelo
1'2 us	6´6 us

Lectura El proceso de lectura del periférico LA-4540 utilizando el puerto paralelo es similar al proceso de escritura. También se pueden diferenciar dos etapas - carga del bus de direcciones y carga del bus de datos - y como en el caso anterior se puede hacer la mejora de comprobar si la siguiente instrucción hace referencia a la misma dirección y así no realizar la etapa de carga de dirección en el 74273.

Como en el caso anterior la lectura está compuesta de 6 instrucciones; las tres primeras son como en la escritura, y consiste en cargar la dirección en el 74273. Lo diferente viene en la 4^a instrucción donde se indica al puerto paralelo que debe leer en vez de escribir.

1. outportb(LPT_CONTROL,0x05)
2. outportb(LPT_DATA, addr)
3. outportb(LPT_CONTROL,0x04)
4. outportb(LPT_CONTROL, 0x2E): Se cambia el sentido de transmisión del puerto paralelo, ahora recibe datos.
   - Se ponen las señales de control de tal forma que el 74F245 pasa los datos del LA al puerto paralelo.
   - El LA está direccionado y sabe que debe realizar un IN; $\overline{SEN} = \overline{IOR} =$ '0'. Por lo que pone información en el bus de datos.
5. var = inportb(LPT_DATA): Se le indica al puerto paralelo que lea sus entradas.
6. outportb(LPT_CONTROL, 0x04): Líneas de control en alta impedancia.

En este caso no interesa hablar de IN's a direcciones distintas pues no es un caso que se suela dar sino que la situación que importa analizar es cuando se piden los datos del analizador.

- El tiempo de lecturas consecutivas es de 6'6us:

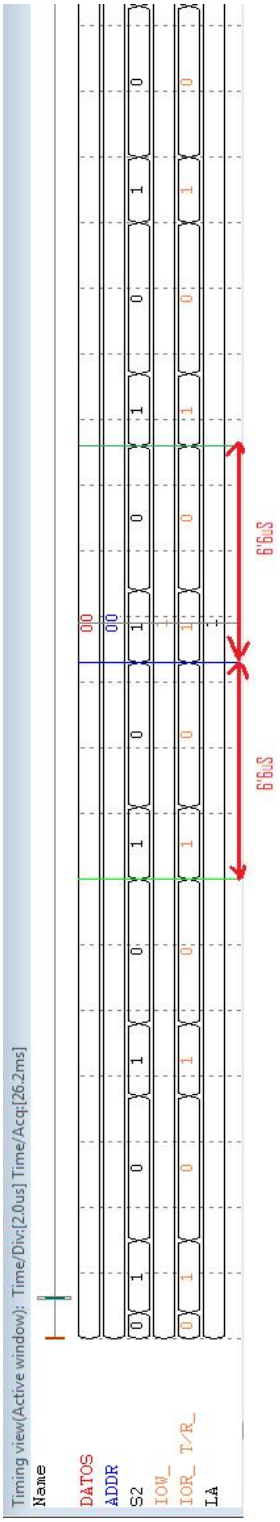


Figura 4.22: Cronograma del interface Paralelo. Caso IN

Si se comparan los interfaces ISA vs Paralelo en cuanto a velocidad de recogida de datos:

IN ISA	IN Paralelo
1'1 us	6´6 us

Tras explicar como el interface paralelo realiza las escrituras y lecturas el lector de este proyecto podría preguntarse si es posible realizar las mismas acciones sin utilizar el CI 74F245, la respuesta es afirmativa. No sería necesario la inclusión de este integrado pero su uso permite mejorar la transmisión/recepción entre LA y puerto ya que adaptan niveles de señal.

4.3.5. Layout de la PCB paralelo

Finalmente, tras el diseño del software y del hardware necesario se realiza el diseño de la PCB. Para ello se emplea el programa Ares (Advanced Routing and Editing Software) asociado al paquete de programas de la familia Proteus. El diseño de la solución final contiene pasos específicos debido a que la fabricación de la PCB se realiza mediante una microfresadora y que la placa se introduce en una caja como acabado final.

En primer lugar se dibuja el borde de la placa en el programa, estableciendo así el tamaño de placa mínimo necesario para la fabricación de la PCB. A continuación se plasma el diseño de la caja en la placa. Se dibuja el corte que la placa precisa para poder ser introducida dentro de la caja en una capa de mecanizado. Posteriormente se dibujan los agujeros necesarios para los tornillos de sujeción a la caja.

El siguiente paso es el rutado de las pistas y la colocación de los componentes. Los componentes se distribuyen en la cara de componentes y se unen por medio de pistas según indican la líneas de conectividad. El rutado de la placa se realiza buscando la distribución más favorable de las distintas posibilidades de ubicación de los componentes, minimizando en lo posible el uso de vías y la longitud y ángulos de las pistas.

La utilización de vías se hace necesaria para que la PCB sea un diseño a una sola cara, bien por la imposibilidad de rutado o bien para salvar pistas gruesas de alimentación en el conector SCSI. Pese a la utilización de vías en el diseño, el hecho de realizar una placa a una sola cara simplifica y facilita el proceso de fabricación.

Acabado el rutado de las pistas se procede a engrosar las pistas de alimentación conforme a la corriente que se espera que circule por ellas.

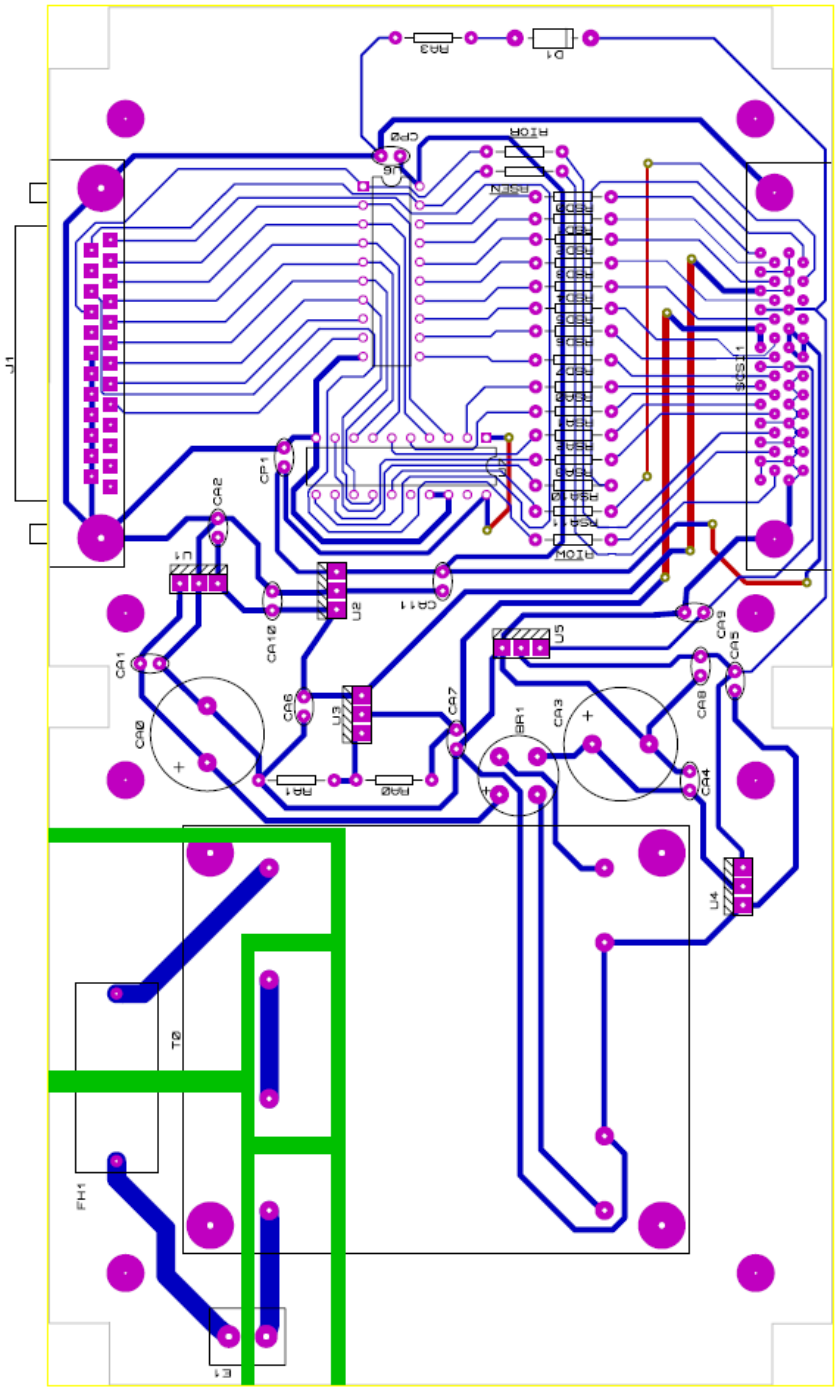


Figura 4.23: Layout PCB paralelo

4.4. Prototipo PCB para protocolo USB 2.0

4.4.1. Descripción del protocolo USB 2.0

4.4.1.1. Introducción

El Universal Serial Bus (bus universal en serie) o Conductor Universal en Serie es un puerto que sirve para conectar periféricos a una computadora. Fue creado en 1996 por siete empresas: IBM, Intel, Northern Telecom, Compaq, Microsoft, Digital Equipment Corporation y NEC.

El estándar incluye la transmisión de energía eléctrica al dispositivo conectado. Algunos dispositivos requieren una potencia mínima, así que se pueden conectar varios sin necesitar fuentes de alimentación extra. La gran mayoría de los concentradores incluyen fuentes de alimentación que brindan energía a los dispositivos conectados a ellos, pero algunos dispositivos consumen tanta energía que necesitan su propia fuente de alimentación. Los concentradores con fuente de alimentación pueden proporcionarle corriente eléctrica a otros dispositivos sin quitarle corriente al resto de la conexión (dentro de ciertos límites).

El diseño del USB tenía en mente eliminar la necesidad de adquirir tarjetas separadas para poner en los puertos bus ISA o PCI, y mejorar las capacidades plug-and-play permitiendo a esos dispositivos ser conectados o desconectados al sistema sin necesidad de reiniciar. Cuando se conecta un nuevo dispositivo, el servidor lo enumera y agrega el software necesario para que pueda funcionar.

El USB puede conectar los periféricos como ratones, teclados, escáneres, cámaras digitales, teléfonos móviles, reproductores multimedia, impresoras, discos duros externos, tarjetas de sonido, sistemas de adquisición de datos y componentes de red. El USB no ha remplazado completamente a los teclados y ratones PS/2, pero virtualmente todas las placas base de PC traen uno o más puertos USB. Sin embargo, el USB tiene una importante ventaja en su habilidad de poder instalar y desinstalar dispositivos sin tener que abrir el sistema, lo cual es útil para dispositivos de almacenamiento externo.

Un sistema USB tiene un diseño asimétrico consistente en un host y un número múltiple de puertos USB en cascada, con múltiples dispositivos periféricos conectados en una topología de estrella. Pueden añadirse hubs USB en los diferentes niveles de la topología, permitiendo una topología en árbol, y con hasta un máximo de 5 niveles. Un host USB puede tener múltiples controladores de host, y cada controlador de host puede proveer uno o mas puertos USB. Hasta 127 dispositivos incluyendo los dispositivos hub pueden ser conectados a un solo controlador de host.

Los dispositivos USB son conectados en serie a través de hubs. Siempre existe un hub

conocido como el hub raíz, el cual está construido dentro del controlador de host.

Un dispositivo físico de USB único debe consistir en varios sub-dispositivos lógicos que se refieren a funciones de dispositivos, porque cada dispositivo individual debe proporcionar varias funciones, tales como webcam (función de dispositivo de vídeo) con un micrófono construido en su interior (función de dispositivo de audio). Los dispositivos finales de la cadena de conexión USB son normalmente los dispositivos conectados.

La comunicación USB está basada en canales lógicos. Los canales son conexiones entre el controlador de host y una entidad lógica en el dispositivo final de la cadena USB. Un dispositivo USB puede tener hasta 32 canales activos, 16 dentro del controlador de host y 16 fuera del controlador. Cada dispositivo final puede transferir datos en una sola dirección, dentro y fuera del dispositivo, por lo que es unidireccional. Los dispositivos finales son agrupados en interfaces y cada interfaz es asociada con una sola función del dispositivo.

Cuando un dispositivo USB nuevo es conectado a un host USB, comienza el proceso de enumeración del dispositivo. El proceso de enumeración envía primero una señal de reset al dispositivo USB. La velocidad del dispositivo USB es determinada durante la señalización del reset. Después del reset, la información del dispositivo USB es leída del dispositivo por el host y el dispositivo es asignado a una única dirección de 7 bits de un controlador host específico. Si el dispositivo es reconocido por el host, los drivers necesarios para la comunicación con el dispositivo son cargados y el dispositivo se configura en un proceso de configuración. Si el USB es reiniciado, el proceso de enumeración es repetido para todos los dispositivos conectados.

El controlador de host sondea el bus para conocer el tráfico, normalmente en un algoritmo Round Robin, de forma que el dispositivo USB no puede transferir ningún dato en el bus sin una petición explícita del controlador de host.

4.4.1.2. Características de transmisión

Los dispositivos USB se clasifican en cuatro tipos según su velocidad de transferencia de datos:

1. **Low Speed** (1.0): Tasa de transferencia de hasta 1.5Mbit/s (192KB/s). Utilizado en su mayor parte por Dispositivos de Interfaz Humana (Human Interface Device) como los teclados, los ratones y los joysticks.

2. **Full Speed** (1.1): Tasa de transferencia de hasta 12Mbit/s (1.5MB/s). Esta fue la más rápida antes de la especificación USB 2.0 y muchos dispositivos fabricados en la actualidad trabajan a esta velocidad. Estos dispositivos, dividen el ancho de banda de la conexión USB entre ellos basados en un algoritmo de búferes FIFO.

3. **High-Speed** (2.0): Tasa de transferencia de hasta 480Mbit/s (60MB/s).

4. **Super-Speed** (3.0): Actualmente en fase experimental y con tasa de transferencia de hasta 4.8Gbit/s (600MB/s). Esta especificación no es definitiva y se preve que sea lanzada en 2008 por la compañía Intel. La velocidad del bus será diez veces más rápida que la de USB 2.0 debido a la sustitución del enlace tradicional por uno de fibra óptica que trabaja con conectores tradicionales de cobre para hacerlo compatible con los estándares anteriores.

Pin	**Nombre**	**Color del cable**	**Conexión**
1	VCC	Rojo	+5V
2	D-	Blanco	DATA -
3	D+	Verde	DATA +
4	GND	Negro	0V

Cuadro 4.11: Conexión USB

Las señales del USB son transmitidas en un cable de par trenzado con impedancia de 90Ω ±15 %, cuyos pares son llamados D+ y D-. Éstos, colectivamente utilizan señalización diferencial en half dúplex para combatir los efectos del ruido electromagnético en enlaces largos. D+ y D- usualmente operan en conjunto y no son conexiones simples. Los niveles de transmisión de la señal varían de 0 a 0.3V para niveles bajos (ceros) y de 2.8 a 3.6V para niveles altos (unos) en las versiones 1.0 y 1.1, y en ±400mV en Alta Velocidad (2.0).

Un bus autoalimentado de un periférico requiere un regulador de tensión de 3,3 V para alimentar la lógica del bus y para alimentar con el voltaje correcto a una resistencia de 1500Ω que puede ser conectada a cada uno de los pines USB D- y D+. Esta resistencia de pull-up señaliza al host que existe un dispositivo conectado, e indica la velocidad de operación del dispositivo. Pull-up sobre D+ indica Full Speed; Pull-up sobre D- indica Low Speed.

El otro lado de la conexión (host o hub) contiene resistencias de 15kΩ en pull-down en las patillas D+ y D- para poder detectar las resistencias de pull-up.

Pin de Pull-up	Modo de operación
D-	Low Speed
D+	Full Speed

Cuadro 4.12: Selección de velocidad para el dispositivo USB

Este puerto sólo admite la conexión de dispositivos de bajo consumo, es decir, que tengan un consumo máximo de 100mA por cada puerto, pero en caso que estuviese conectado un dispositivo que permite 4 puertos por cada salida USB (extensiones de máximo 4 puertos) entonces la energía del USB se asigna en unidades de 100mA hasta un máximo de 500mA por puerto.

USB utiliza un protocolo especial llamado "chirping" para negociar el modo High-Speed. En términos simples, un dispositivo que tiene capacidad HS siempre es conectado en primer lugar como un dispositivo FS, pero después de recibir un USB RESET (ambas D+ y D- son puestas a nivel bajo por el host) el dispositivo intenta elevar el nivel de la patilla D- a nivel alto. Si el host (o el hub) tiene capacidad HS, él retorna alternativamente las señales D- y D+ permitiendo al dispositivo saber que puede operar en High Speed.

La tolerancia del reloj es de 480.00 Mbit/s ±500ppm, 12.000 Mbit/s ±2500ppm, 1.50 Mbit/s ±15000ppm.

El estándar USB usa el sistema NRZI para codificar los datos, y utiliza el método conocido como "bit stuffing" por el que inyecta un “cero” artificial en el flujo de datos si contiene seis “unos” consecutivos antes de convertir el flujo de bits a NRZI.

Normalmente es pensado que los dispositivos Hi-Speed se refieren a "USB 2.0" y pueden llegar hasta una tasa de transferencia de 480 Mbit/s, pero no todos los dispositivos USB 2.0 son Hi-Speed. El USB-IF (USB Implementers Forum) certifica los dispositivos y proporciona licencias de comercio de logos especiales para cada uno de los dispositivos "Basic-Speed" (baja y alta velocidad) o Hi-Speed después de pasar un test de conformidad y pagar unos derechos de licencia.Todos los dispositivos son probados de acuerdo con las últimas especificaciones, así que los dispositivos recientes denominados Low-Speed son también dispositivos 2.0.

4.4.1.3. Cables USB

La longitud máxima del cable de par trenzado USB estándar es de 5 metros. La principal razón de este límite es el retraso máximo permitido de la señal de cerca de 1500 ns. Si un dispositivo USB no contesta a los comandos del host dentro del tiempo permitido, el host considerará que el comando se ha perdido. Cuando el dispositivo USB responde a tiempo, los retrasos utilizando el mayor número de hubs y los retrasos de los cables conectados a los hubs, sumandos los retrasos del host y los dispositivos, el máximo retraso provocado por un solo cable resulta ser de 26 ns. La especificación de USB 2.0 indica que el retraso del cable debe ser menor de 5,2 ns por metro, lo que significa que la longitud máxima de un cable USB es de 5 metros. Sin embargo, esto está muy cerca de la máxima longitud posible cuando se usa

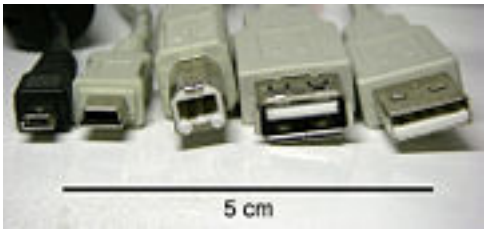
un cable estándar de cobre.

Aunque un solo cable esta limitado a 5 metros, las especificación del USB permite cinco hubs USB en una larga cadena de cables y hubs. Consecuentemente la distancia máxima de señal del cable es de 30 metros, utilizando seis cables de 5 metros y cinco hubs. En los usos actuales, el último hub es el punto final de la conexión más conveniente para varios dispositivos USB, por lo que la distancia final útil se establece en 25 metros.

4.4.1.4. Conectores

El estándar USB especifica tolerancias de impedancia y de especificaciones mecánicas relativamente bajas para sus conectores, intentando minimizar la incompatibilidad entre los conectores fabricados por distintas compañías. Una meta a la que se ha logrado llegar. El estándar USB, a diferencia de otros estándares también define tamaños para el área alrededor del conector de un dispositivo, para evitar el bloqueo de un puerto adyacente por el dispositivo en cuestión.

Las especificaciones USB 1.0, 1.1 y 2.0 definen dos tipos de conectores para conectar dispositivos al servidor: A y B. Sin embargo, la capa mecánica ha cambiado en algunos conectores. Por ejemplo, el IBM UltraPort es un conector USB privado localizado en la parte superior del LCD de los computadoras portátiles de IBM. Utiliza un conector mecánico diferente mientras mantiene las señales y protocolos característicos del USB. Otros fabricantes de artículos pequeños han desarrollado también sus medios de conexión pequeños, y ha aparecido una gran variedad de ellos, algunos de baja calidad.



Diferentes tipos de conectores de izquierda a derecha: Micro USB, Mini USB, Tipo B, Hembra tipo A, Macho Tipo A.

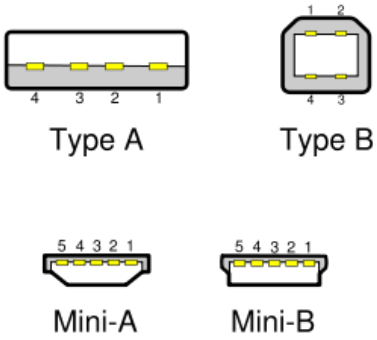


Figura 4.24: Conectores USB

Una extensión del USB llamada "USB-On-The-Go" (sobre la marcha) permite a un puerto actuar como servidor o como dispositivo - esto se determina por qué lado del cable está conectado al aparato. Incluso después de que el cable está conectado y las unidades se están comunicando, las 2 unidades pueden "cambiar de papel" bajo el control de un programa. Esta facilidad está específicamente diseñada para dispositivos como PDA, donde el enlace USB podría conectarse a un PC como un dispositivo, y conectarse como servidor a un teclado o ratón. El "USB-On-The-Go" también ha diseñado 2 conectores pequeños, el mini-A y el mini-B, así que esto debería detener la proliferación de conectores miniaturizados de entrada.

4.4.1.5. Alimentación

La especificación USB proporciona una alimentación de de 5 voltios en un solo cable desde el cual los dispositivos USB conectados pueden obtener alimentación. La especificación proporciona no más de 5,25 V y no menos de 4,75 V (5 V±5 %) entre las líneas del bus de alimentación positivo y negativo. Inicialmente, solo se permite un consumo de 100 mA por dispositivo. El dispositivo debe preguntar para obtener más corriente a los dispositivos que se

encuentran jerárquicamente por encima en tramos de 2 mA hasta un máximo de 500mA.

Nuevas especificaciones para carga de baterías (Battery Charging Specification) permite a los dispositivos USB consumir una corriente de hasta 1,5 A (modos Low y Full-Speed, o 900 mA en modo Hi-Speed) desde hubs y hosts, o hasta 1,8 A para cargadores dedicados que siguen la especificación Battery Charging Specification. Los cargadores específicos cortocircuitan los pines D- y D+ juntos de forma que no se puede enviar o recibir información en esas líneas, permitiendo la fabricación simple de cargadores de alta corriente. El incremento de corriente se produce cuando los hubs/hosts soportan esta nueva especificación.

4.4.2. Circuito Integrado FT2232D Dual USB UART/FIFO

El chip escogido para realizar el interfaz entre el protocolo USB y el protocolo de bus utilizado por el analizador es el FT2232 de FTDI. Se escoge por su sencillez de uso ya que posee un modo de funcionamiento específico que simula un host de bus estándar 8048 / 8051 por medio de una MCU. Su encapsulado es LQFP de 48 patillas.

El 8051 de Intel es un microcontrolador integrado en un solo chip desarrollado inicialmente por Intel en 1980 pare ser utilizado en sistemas embebidos. Sus características ofrecen las operaciones necesarias para el control de los elementos de una computadora (CPU, RAM, ROM, I/O, interrupciones lógicas, temporizador, etc) por lo que se hizo muy popular en el diseño de las mismas. Su función de host realizaba el control del bus de sistema respetando el protocolo de funcionamiento ISA.

Además del MCU Host Bus Emulation Mode, las características del dispositivo FT2232 ofrecen dos controladores UART / FIFO mutipropósito, los cuales pueden ser configurados de forma individual en varios modos. Tanto como el interfaz UART, el interfaz FIFO y el modo de I/O Bit-Bang de segunda generación de FTDI, el FT2232D ofrece una variedad de nuevos modos de operación, incluyendo un motor serie síncrono multi-protocolo (MPSSE) especialmente diseñado para protocolos serie síncronos como JTAG, I2C, y bus SPI.

FTDI proporciona una licencia gratuita del driver Virtual Com Port (V.C.P) que hace que los puertos del periférico parezcan puertos COM estándar de un PC. La mayoría de las aplicaciones software existentes deberían ser capaces de interfaz con los nuevos puertos virtuales COM creados por el driver del dispositivo simplemente reconfigurándolos. Usando los drivers VCP un programador de aplicaciones puede comunicarse con el dispositivo exactamente del mismo modo que cuando se utiliza un puerto COM de un PC. El driver del FT2232 también incorpora las funciones definidas por los drivers D2XXX de FTDI, permitiendo a los programadores del aplicaciones realizar un interfaz software directamente sobre el dispositivo utilizando una DLL para Windows. Los detalles del driver y del interfaz de programación

pueden ser encontrados en el website de FTDI www.ftdichip.com.

Algunas características importantes del dispositivo FTDI FT2232 son:

- Dos canales I/O (A y B) individualmente configurables en los estilos de interface UART y FIFO. Además esos canales pueden ser configurados en modos de IO especiales.
- Circuito integrado Power-On-Reset (POR). La función interna POR está disponible a través del pin RESET# permitiendo resetear a la lógica externa cuando sea necesario. Aunque este pin puede ser simplemente cableado a Vcc. El pin RSTOUT# proporciona un reset estable a una MCU externa u otros dispositivos.
- Circuito integrado RCCLK se usa para asegurar que el oscilador y el PLL multiplicador de de frecuencia de reloj es estable antes de la enumeración del USB.
- Conversor de tensión integrado en interfaz UART / FIFO y señales de control. Cada canal tiene su propio e independiente pin de VCCIO que puede ser alimentado desde 3V a 5V.
- Control de administración de alimentación mejorado para dispositivos de alta alimentación USB alimentados por bus. El pin PWREN# es activado cuando el dispositivo es enumerado por el USB, y desactivado cuando el dispositivo se suspende. Es útil para controlar la alimentación de un circuito externo.
- Soporte para USB síncrono de Transferencias. Mientras que la transferencia masiva de USB suele ser la mejor opción para la transferencia de datos, la programación del tiempo de los datos no está garantizada. Para aplicaciones en las que la programación de latencia tiene prioridad sobre la integridad de los datos, tales como la transferencia de audio y vídeo de bajo ancho de banda de datos, el FT2232D ofrece la opción de USB síncrono de transferencia a través de la configuración de bits en la EEPROM.
- Baja corriente de suspensión. Típicamente inferior a 100 µA (excluyendo la resistencia de 1.5K de pull up en USBDP) en el modo de USB suspendido.
- Timeout de recepción de buffer programable. El timeout del buffer de TX es programable sobre el USB en incrementos de 1ms, desde 1ms hasta 255ms, lo que permite al dispositivo estar mejor optimizado para protocolos que requieran tiempos de respuesta rápidos para pequeños paquetes de datos.
- Divisores de pre-escala de velocidad de transmisión (UART mode).

- Soporte extendido para EEPROM. El FT2232D soporta las EEPROM's 93C46 (64 x 16 bit), 93C56 (128 x 16 bit), and 93C66 (256 x 16 bit).

- USB 2.0 (opción full speed). Una opción basada en una EEPROM permite al FT2232D retornar un descriptor de USB 2.0 en contra del modo USB 1.1.

De los modos de funcionamiento incorporados por el FT2232, el modo de funcionamiento MCU Emulation Bus Mode es el de interés. Este nuevo modo combina los puertos «A» y «B» del dispositivo para hacer al FT2232D emular un interfaz estándar 8048 / 8051 MCU estilo bus. Esto permite que los dispositivos periféricos para estas familias MCU sean directamente atacados por USB a través de las entradas y salidas del FT2232D, con la ayuda de la tecnología MPSSE de interfaz.

Los 8 bits mas bajos (AD7 to AD0) son una multiplexación de bus de Direcciones / Datos. Los bits de A8 hasta A15 proporcionan el direccionamiento alto (direccionamiento extendido).

Existen cuatro operaciones básicas:

1. Read (no cambian de A15 a A8).

2. Read Extended (cambian de A15 a A8).

3. Write (no cambian de A15 a A8).

4. Write Extended (cambian de A15 a A8).

Habilitación del modo MCU Host Bus Emulation

El modo de emulación de bus de host se habilita utilizando el comando del driver Set Big Bang Mode. El valor hexadecimal 0x08 habilita el modo, mientras que un valor de 0x00 resetea el dispositivo. Para más información leer la nota de aplicación AN2232L-02, "Bit Mode Functions for the FT2232D" para mayor detalles y ejemplos. Los comandos de MCU Host Bus Emulation Mode se describen de forma completa en la nota de aplicación AN2232L-01 - "Command Processor For MPSSE and MCU Host Bus Emulation Modes".

Pines comunes

Los pines de uso común del dispositivo funcionan de la misma forma sin tener en cuenta el modo de funcionamiento empleado y son los siguientes:

USB INTERFACE GROUP

Pin#	Signal	Type	Description
7	USBDP	I/O	USB Data Signal Plus (Requires 1.5K pull-up to 3V3OUT or RSTOUT#)
8	USBDM	I/O	USB Data Signal Minus

EEPROM INTERFACE GROUP

Pin#	Signal	Type	Description
48	EECS	I/O	EEPROM – Chip Select. Tri-State during device reset. **Note 1
1	EESK	OUTPUT	Clock signal to EEPROM. Tri-State during device reset, else drives out. **Note 1
2	EEDATA	I/O	EEPROM – Data I/O Connect directly to Data-In of the EEPROM and to Data-Out of the EEPROM via a 2.2K resistor. Also, pull Data-Out of the EEPROM to VCC via a 10K resistor for correct operation. Tri-State during device reset. **Note 1

MISCELLANEOUS SIGNAL GROUP

Pin#	Signal	Type	Description
4	RESET#	INPUT	Can be used by an external device to reset the FT2232D. If not required, tie to VCC. **Note 1
5	RSTOUT#	OUTPUT	Output of the internal Reset Generator. Drives low for 5.6 ms after VCC > 3.5V and the internal clock starts up, then clamps it's output to the 3.3V output of the internal regulator. Taking RESET# low will also force RSTOUT# to drive low. RSTOUT# is NOT affected by a USB Bus Reset.
47	TEST	INPUT	Puts device into I.C. test mode – must be tied to GND for normal operation.
41	PWREN#	OUTPUT	Goes Low after the device is configured via USB, then high during USB suspend. Can be used to control power to external logic using a P-Channel Logic Level MOSFET switch. Enable the Interface Pull-Down Option in EEPROM when using the PWREN# pin in this way.
43	XTIN	INPUT	Input to 6MHz Crystal Oscillator Cell. This pin can also be driven by an external 6MHz clock if required. Note : Switching threshold of this pin is VCC/2, so if driving from an external source, the source must be driving at 5V CMOS level or a.c. coupled to centre around VCC/2.
44	XTOUT	OUTPUT	Output from 6MHz Crystal Oscillator Cell. XTOUT stops oscillating during USB suspend, so take care if using this signal to clock external logic.

**Note 1 - *During device reset, these pins are tri-state but pulled up to VCC via internal 200K resistors.*

Figura 4.25: MCU Host Bus Emulation - Pines comunes

POWER AND GND GROUP

Pin#	Signal	Type	Description
6	3V3OUT	OUTPUT	3.3 volt Output from the integrated L.D.O. regulator This pin should be decoupled to GND using a 33nF ceramic capacitor in close proximity to the device pin. It's prime purpose is to provide the internal 3.3V supply to the USB transceiver cell and the RSTOUT# pin. A small amount of current (<= 5mA) can be drawn from this pin to power external 3.3V logic if required.
3, 42	VCC	PWR	+4.35 volt to +5.25 volt VCC to the device core, LDO and non-UART / FIFO controller interface pins.
14	VCCIOA	PWR	+3.0 volt to +5.25 volt VCC to the UART / FIFO A Channel interface pins 10..13, 15..17 and 19..24. When interfacing with 3.3V external logic in a bus powered design connect VCCIO to a 3.3V supply generated from the USB bus. When interfacing with 3.3V external logic in a self powered design connect VCCIO to the 3.3V supply of the external logic. Otherwise connect to VCC to drive out at 5V CMOS level.
31	VCCIOB	PWR	+3.0 volt to +5.25 volt VCC to the UART / FIFO B Channel interface pins 26..30, 32..33 and 35..40. When interfacing with 3.3V external logic in a bus powered design connect VCCIO to a 3.3V supply generated from the USB bus. When interfacing with 3.3V external logic in a self powered design connect VCCIO to the 3.3V supply of the external logic. Otherwise connect to VCC to drive out at 5V CMOS level.
9,18, 25, 34	GND	PWR	Device - Ground Supply Pins
46	AVCC	PWR	Device - Analog Power Supply for the internal x8 clock multiplier. A low pass filter consisting of a 470 Ohm series resistor and a 100 nF to GND should be used on the supply to this pin.
45	AGND	PWR	Device - Analog Ground Supply for the internal x8 clock multiplier

Figura 4.26: MCU Host Bus Emulation - Pines de alimentación

Cuando el MCU Host Bus Emulation es el modo habilitado las líneas de señal de IO en ambos canales trabajan juntas y los pines se configuran de la siguiente manera (figura 4.27):

Pin#	Signal	Type	Description
24	AD0	I/O	Address / Data Bus Bit 0 **Note 28
23	AD1	I/O	Address / Data Bus Bit 1 **Note 28
22	AD2	I/O	Address / Data Bus Bit 2 **Note 28
21	AD3	I/O	Address / Data Bus Bit 3 **Note 28
20	AD4	I/O	Address / Data Bus Bit 4 **Note 28
19	AD5	I/O	Address / Data Bus Bit 5 **Note 28
17	AD6	I/O	Address / Data Bus Bit 6 **Note 28
16	AD7	I/O	Address / Data Bus Bit 7 **Note 28
15	I/O0	I/O	MPSSE mode instructions to set / clear or read the high byte of data can be used with this pin. **Note 28, **Note 29
13	I/O1	I/O	MPSSE mode instructions to set / clear or read the high byte of data can be used with this pin. In addition this pin has instructions which will make the controller wait until it is high, or wait until it is low. This can be used to connect to an IRQ pin of a peripheral chip. The FT2232D will wait for the interrupt, and then read the device, and pass the answer back to the host PC. I/O1 must be held in input mode if this option is used. **Note 28, **Note 29
12	IORDY	INPUT	Extends the time taken to perform a Read or Write operation if pulled low. Pull up to Vcc if not being used.
11	OSC	OUTPUT	Shows the clock signal that the circuit is using.
40	A8	OUTPUT	Extended Address Bus Bit 8
39	A9	OUTPUT	Extended Address Bus Bit 9
38	A10	OUTPUT	Extended Address Bus Bit 10
37	A11	OUTPUT	Extended Address Bus Bit 12
36	A12	OUTPUT	Extended Address Bus Bit 13
35	A13	OUTPUT	Extended Address Bus Bit 14
33	A14	OUTPUT	Extended Address Bus Bit 15
32	A15	OUTPUT	Extended Address Bus Bit 16
30	CS#	OUTPUT	Negative pulse to select device during Read or Write.
29	ALE	OUTPUT	Positive pulse to latch the address.
28	RD#	OUTPUT	Negative Read Output.
27	WR#	OUTPUT	Negative Write Output. (Data is setup before WR# goes low, and is held after WR# goes high)

**Note 28 : *In Input Mode, these pins are pulled to VCCIO via internal 200K resistors. These can be programmed to gently pull low during USB suspend (PWREN# = "1") by setting this option in the EEPROM.*

**Note 29 : *These instrucions are fully described in the application note AN2232L-01 - "Command Processor For MPSSE and MCU Host Bus Emulation Modes".*

Figura 4.27: Configuración de los pines en el modo MCU Host Bus Emulation

Ciclo de escritura

El ciclo de escritura es el que se muestra en el cronograma de la figura 4.28.

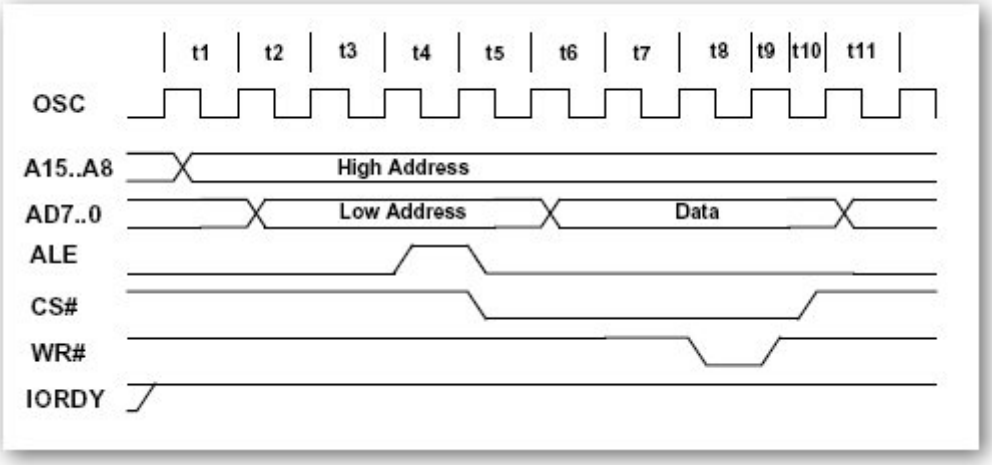


Figura 4.28: MCU Host Bus Emulation - Ciclo de escritura

Es de importancia recordar que la patilla 12 de señal IORDY permite alargar el ciclo de escritura o de lectura 5 periodos de reloj si se dispone en configuración pull-down.

Time	Description
t1	High address byte is placed on the bus if the extended write is used.
t2	Low address byte is put out.
t3	1 clock period for address is set up.
t4	ALE goes high to enable latch. This will extend to 2 clocks wide if IORDY is low.
t5	ALE goes low to latch address and CS# is set active low.
t6	Data driven onto the bus.
t7	1 clock period for data setup.
t8	WR# is driven active low. This will extend to 6 clocks wide if IORDY is low.
t9	WR# is driven inactive high.
t10	CS# is driven inactive, 1/2 a clock period after WR# goes inactive
t11	Data is held until this point, and may now change

Figura 4.29: MCU Host Bus Emulation - Secuencia de escritura

Ciclo de lectura

El ciclo de lectura es el que se muestra en el cronograma de la figura 4.30.

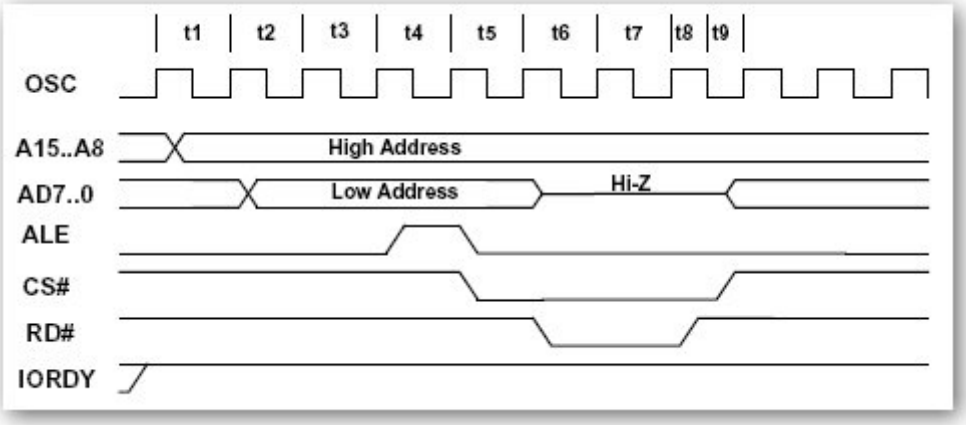


Figura 4.30: MCU Host Bus Emulation - Ciclo de escritura

Es de importancia recordar que la patilla 12 de señal IORDY permite alargar el ciclo de escritura o de lectura 5 periodos de reloj si se dispone en configuración pull-down.

Time	Description
t1	High address byte is placed on the bus if the extended read is used - otherwise t1 will not occur.
t2	Low address byte is put out.
t3	1 clock period for address set up.
t4	ALE goes high to enable address latch. This will extend to 2 clocks wide if IORDY is low.
t5	ALE goes low to latch address, and CS# is set active low. This will extend to 3 clocks if IORDY is sampled low. CS# will always drop 1 clock after ALE has gone high no matter the state of IORDY.
t6	Data is set as input (Hi-Z), and RD# is driven active low.
t7	1 clock period for data setup. This will extend to 5 clocks wide if IORDY# is sampled low.
t8	RD# is driven inactive high.
t9	CS# is driven inactive 1/2 a clock period after RD# goes inactive, and the data bus is set back to output.

Figura 4.31: MCU Host Bus Emulation - Secuencia de escritura

4.4.3. Descripción del hardware

La solución hardware USB se implementa partiendo de la fuente de alimentación diseñada en apartados anteriores, siendo la fuente de alimentación la encargada de proporcionar los niveles de tensión y voltaje necesarios para el analizador lógico y el chip FT2232. Dado que el chip se alimenta con 5 voltios se puede alimentar directamente desde la fuente diseñada.

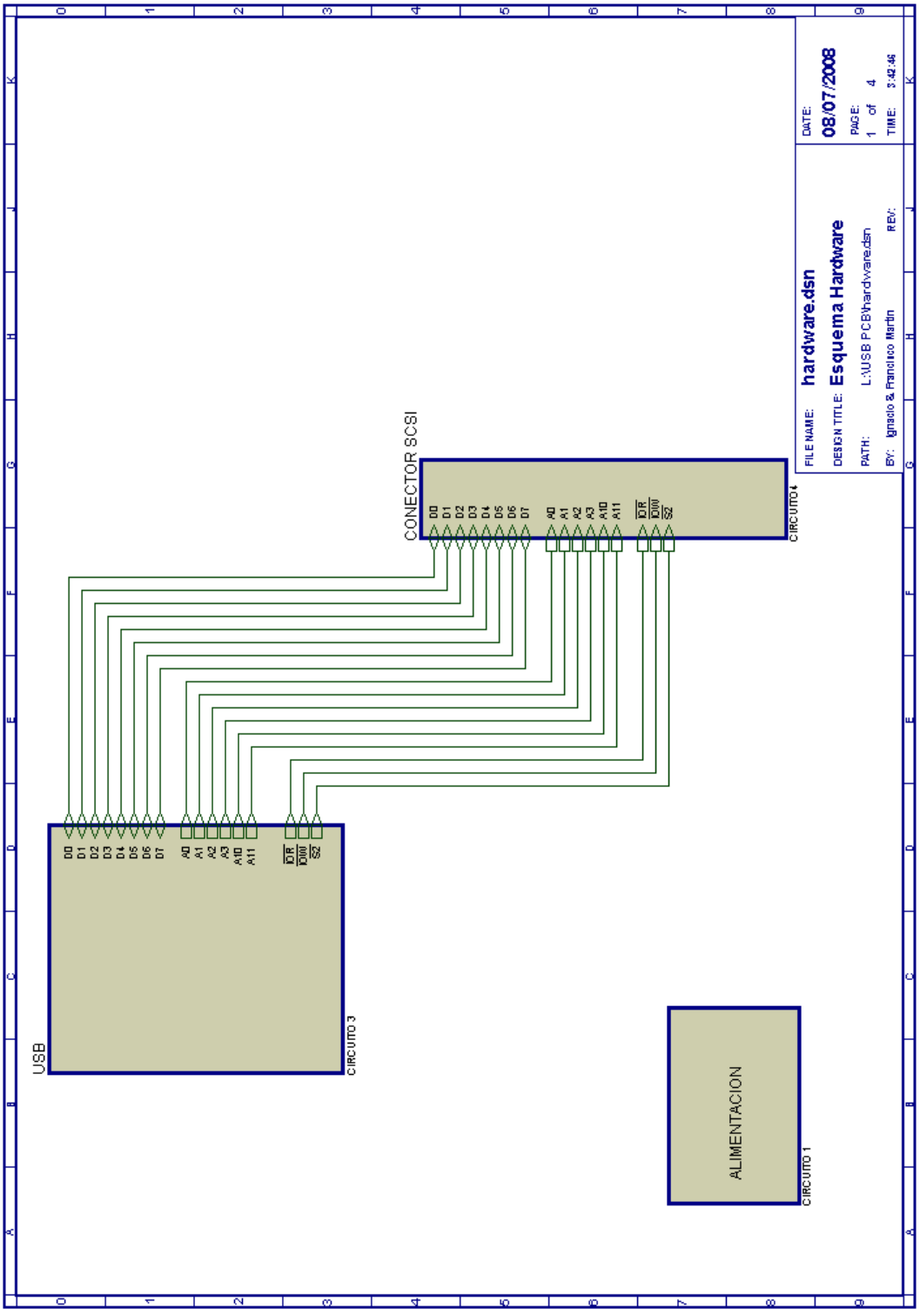


Figura 4.32: Diagrama de bloques de hardware USB

Como se ha expuesto en apartados anteriores, el chip FT2232 realiza la simulación de un host de bus MCU, por lo que no se precisa ningún hardware auxiliar a priori. Sin embargo, si se observan las señales de la tabla 4.27se puede concluir que los pines son los mismos tanto para del bus de datos como para las direcciones AD0:AD7 de menor peso. Para poder utilizar todo el direccionamiento de 16 bits (A15:A8 + AD7:AD0) el funcionamiento de este dispositivo debe de ir acompañado de un latch para las direcciones de menor peso AD7:AD0 activado con un flanco de subida de la señal ALE (como se muestra en las figuras de ciclo de escritura y ciclo de lectura). En un primer paso se realizaría un latch de las direcciones de menor peso, para posteriormente estabilizar las señales de datos y direcciones extendidas, y habilitar las señales de control del protocolo. No obstante, no se necesita todo el direccionamiento completo ya que como se conoce, las señales que se precisan son:

- Bus de datos de 8 bits bidireccional: D7:D0.
- Señales de direcciones: A11, A10, A3, A2, A1 A0.
- Señales de control: $\overline{SEN}$, $\overline{IOR}$,$\overline{IOW}$.

El número de señales de dirección necesarias es de tan solo 6 bits. Para no utilizar un un latch de direcciones para los 8 bits de menor peso se decide utilizar 6 de los 8 pines A15:A8 correspondientes al direccionamiento extendido como salida de las direcciones A11, A10, A3, A2, A1 A0. Los 8 pines de señales AD0:AD7 se utilizan exclusivamente como bus de datos bidireccional. Las señales de control son generadas automáticamente por el chip y tienen su correspondencia con las señales del protocolo precisado. La solución de conexionado de las señales de interés utilizadas junto con su nombre equivalente se especifica en la tabla 4.13 y puede observarse en la figura 4.33.

Cuadro 4.13: Conexión de los pines del FT2232

Pin	Señal	Tipo E/S	Equivalencia	Descripción
24	AD0	E/S	DATA0	Bit de datos 0
23	AD1	E/S	DATA1	Bit de datos 1
22	AD2	E/S	DATA2	Bit de datos 2
21	AD3	E/S	DATA3	Bit de datos 3
20	AD4	E/S	DATA4	Bit de datos 4
19	AD5	E/S	DATA5	Bit de datos 5
17	AD6	E/S	DATA6	Bit de datos 6

16	AD7	E/S	DATA7	Bit de datos 7
12	IORDY	Entrada	IORDY	Pull-down para extender el tiempo de los ciclos de lectura y escritura
40	A8	Salida	ADDR0	Bit de dirección 0
39	A9	Salida	ADDR1	Bit de dirección 1
38	A10	Salida	ADDR2	Bit de dirección 2
37	A11	Salida	ADDR3	Bit de dirección 3
36	A12	Salida	ADDR10	Bit de dirección 10
35	A13	Salida	ADDR11	Bit de dirección 11
33	A14	Salida	-	Sin uso
32	A15	Salida	-	Sin uso
30	$\overline{CS}$	Salida	$\overline{SEN}$	Señal de habilitación
28	$\overline{RD}$	Salida	$\overline{IOR}$	Señal de lectura
27	$\overline{WD}$	Salida	$\overline{IOW}$	Señal de escritura

Esta solución implica una programación específica sobre el dispositivo FT2232, utilizando como se verá posteriormente las instrucciones de escritura extendida y lectura extendida, necesarias para poder utilizar los pines A15:A8 de direccionamiento extendido.

Conforme a lo expuesto en el manual se utiliza la configuración de pull-down en la patilla IORDY para extender los ciclos de escritura y lectura, haciendo el dispositivo compatible con hosts de buses lentos. Esta opción se comprobó durante el montaje del prototipo y se estableció como indispensable para el correcto funcionamiento del analizador lógico.

Como se especifica en el manual, es posible realizar un montaje con la conexión de una memoria EEPROM que contenga la configuración inicial del chip FT2232. Esta información es el modo de funcionamiento de sus dos puertos, así como información VID relativa al vendedor e información PID relativa a la alimentación del chip. En nuestro diseño no se ha utilizado esta característica, aunque se ha incluido la huella de la memoria para la posibilidad de uso esté presente en futuras modificaciones del diseño. Al no utilizar la memoria EEPROM el usuario obtiene los valores del fabricante FTDI incluidos por defecto en el chip.

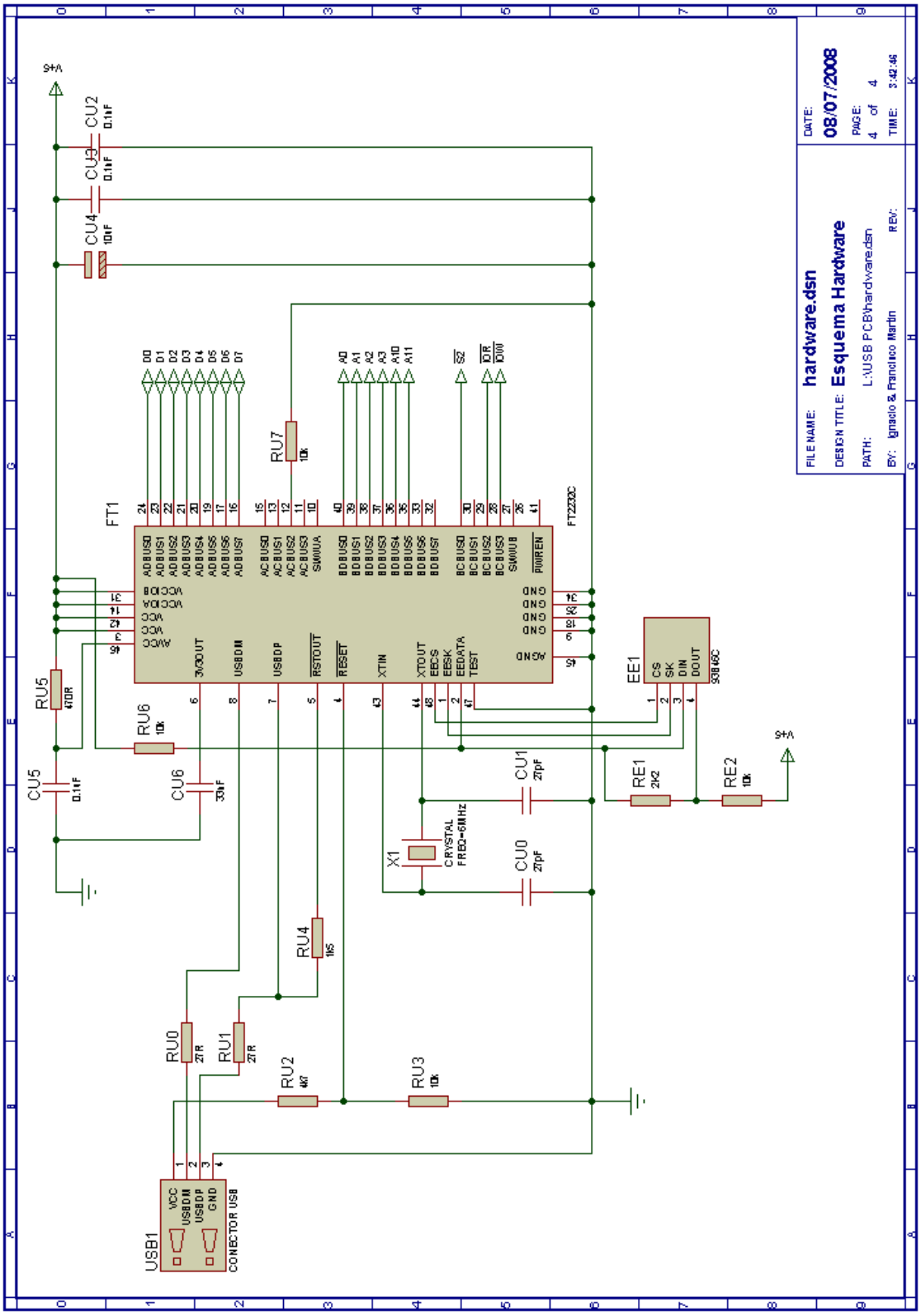


Figura 4.33: Detalle del conector USB

El resto del conexionado del FT2232 se realiza conforme a lo expuesto en el manual de usuario FT2232D Dual USB UART/FIFO I.C. Las opciones de conexionado particulares para este montaje se especifican en la tabla **??**, aunque existen algunas particularidades que se detallan a continuación:

- Se incluye la resistencia RU4 de 1.5 K ohmios entre las patillas USBDP (pin 7) y $\overline{RSTOUT}$ (pin 5) para indicar que el dispositivo se configura como USB 2.0 Full Speed. Durante el reset del dispositivo la patilla $\overline{RSTOUT}$ se encuentra a nivel bajo 5,6 ms, para posteriormente quedar a nivel alto gracias a su conexionado interno con el regulador de tensión de 3,3 V, quedando fijado el pull-up a través de la resistencia de 1,5K.

- La señal IORDY de la patilla 12 se configura en pull-down mediante una resistencia de 10K ohmios para hacer más lento el protocolo de comunicación con el analizador lógico, manteniendo la compatibilidad de tiempo de mantenimiento de las señal de lectura $\overline{RD}$ y la señal de escritura $\overline{WR}$ en 5 ciclos del reloj interno del FT2232.

  En la configuración de pull-up se ha comprobado experimentalmente que las señales de escritura y lectura están a nivel bajo durante un tiempo aproximado de 100 ns. Utilizando esta configuración de pull-down las señales se extienden en el tiempo hasta 500 ns, lo que hace que el protocolo de comunicación sea compatible con el protocolo del LA-4540.

- Las resistencias RE1 y RE2 son excluyentes respecto a la resistencia RU6. En el montaje propuesto se contempla la posibilidad de utilizar una memoria EEPROM, que precisa de las resistencias RE1 y RE2 conforme se detalla en el manual. Estas resistencias sólo deben montarse si se utiliza una memoria EEPROM. En caso contrario sólo debe montarse la resistencia RU6.

En la figura 4.34 se muestra la conexión final del conector SCSI de 50 pines. Esta conexión reproduce fielmente las señales del protocolo ISA provenientes del bus de la computadora, y es similar a la conexión utilizada para el conector SCSI en el desarrollo hardware paralelo. Las señales de datos, direcciones y control que intervienen en el protocolo provenientes del chip son conducidas mediante pistas eléctricas a los correspondientes pines del conector SCSI. En la figura también aparecen las líneas de alimentación necesarias para el correcto funcionamiento del analizador lógico.

En el diseño de las conexiones de las señales se han incluido unas resistencias en serie con las señales de datos, direcciones, y control. Son resistencias de bajo valor y su uso se justifica para evitar que se produzcan cortocircuitos en las líneas de transmisión.

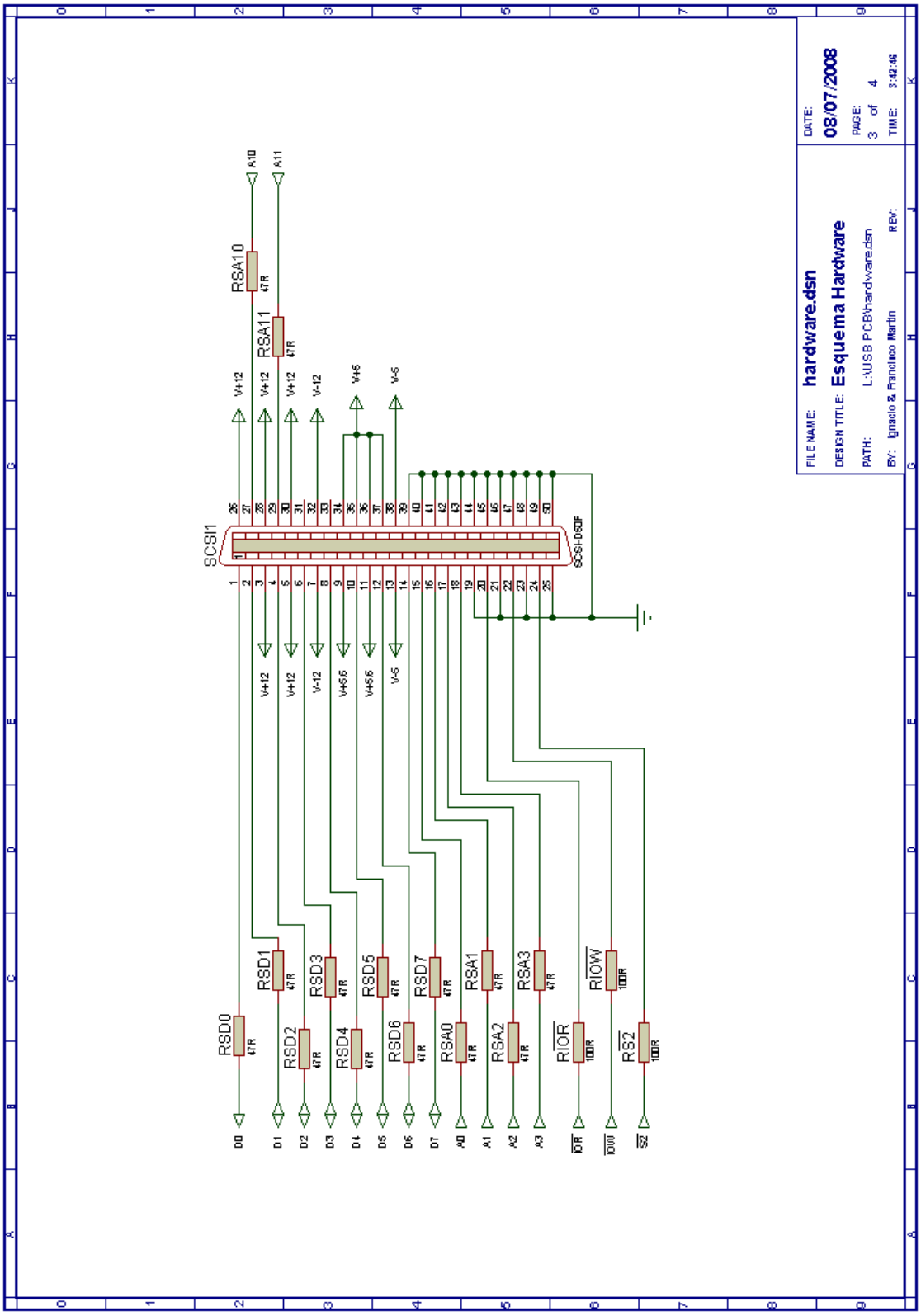


Figura 4.34: Detalle del conector SCSI

4.4.4. Programación del puerto USB

Para poder utilizar el interface USB es necesario que al compilar la aplicación la variable PUERTO valga 2. En ese caso la función inport y outport harán llamadas a los procedimientos y funciones que hacen funcionar el interface USB. Estas rutinas se explican detalladamente en AN2232-02 Bit Mode Functions for the FT2232.

En este manual están explicadas las instrucciones para controlar al FTDI2232, las principales son:

- FT_Open (int iDevice, FT_HANDLE *ftHandle)
- FT_SetBitMode (FT_HANDLE ftHandle, UCHAR ucMask, UCHAR ucMode)
- FT_Close (FT_HANDLE ftHandle)
- FT_Read (FT_HANDLE ftHandle, LPVOID lpBuffer, DWORD dwBytesToRead, LPDWORD lpdwBytesReturned)
- FT_Write (FT_HANDLE ftHandle, LPVOID lpBuffer, DWORD dwBytesToWrite, LPDWORD lpdwBytesWritten)

Con estas 5 instrucciones se puede controlar el chip FTDI2232C para que funcione de interfaz entre PC y el LA-4540.

Como en el caso del interfaz paralelo las operaciones principales son las de escritura y lectura en el periférico, sin embargo, en las aplicaciones basadas en USB es necesario primero realizar unos pasos previos.

Estos pasos previos consisten en:

- *Abrir el dispositivo USB*, obteniendo un manejador o handle - FT_Open -; sirve para poder direccionar el dispositivo USB. En cada instrucción que se refiera al USB se utiliza este manejador y en caso de obtener un fallo debido a desconexión del USB este manejador ya no sería válido y habría que volver a abrir el dispositivo para obtener otro manejador.
- *Resetear el dispositivo USB y ponerlo a funcionar en el modo que nos resulte mas conveniente* - FT_SetBitMode -. En este caso el chip que hace de interfaz entre el PC y el LA está por defecto en un modo de operación que no es válido para la aplicación. Por ello después de obtener el manejador de FTDI2232C se resetea el dispositivo y se le indica que el modo de funcionamiento es el modo MCU Host Bus Emulation Mode.

- *Cerrar el dispositivo USB* - FT_Close -; una vez que la aplicación del LA se finaliza es conveniente liberar el manejador del dispositivo USB.

Siguiendo con la programación de las dos operaciones principales, escritura y lectura sobre el periférico LA-4540 a través de un interfaz USB hay que advertir una sutil diferencia en cuanto a los interface ISA y Paralelo. En esos dos interfaces las instrucciones outport e inport se ejecutaban una a una inmediatamente. En el caso de usar el chip FT2232 ocurre que se mandan bloques de datos que se almacenan en buffers. El FTDI dispone de un *buffer de salida* - para las escrituras- y otro *buffer de entrada* - para las lecturas- cada uno con un tamaño que el programador puede configurar a su gusto. Si esos buffer se ponen con tamaño *uno*, el interface funciona pero los tiempos de transmisión de cada bloque de dato son extremadamente largos, en conclusión no se puede enviar/recibir bloques de un único dato. En el caso del buffer de salida - instrucciones OUT- se comprobó que cuanto mayor fuese el tamaño del bloque de datos más rápida era la aplicación, sin embargo para el caso del buffer de entrada - instrucciones IN - el tamaño óptimo era de 4.

Una vez explicado el término buffer del dispositivo FTDI2232C se puede entender las opciones que tiene la aplicación en modo USB.

Teniendo en cuenta como es la comunicación entre PC y analizador -o bien largas series de outport alternadas muy de vez en cuando con series de 2 a 6 inport, o bien series de inport de 8K a 128K para obtener los datos de los pods - hay que seguir una estrategia para poder utilizar el buffer del FTDI de forma óptima. En los diagramas de bloques de las figuras 4.35y 4.36 se muestra como se ha optado por transmitir los outport y los inport.

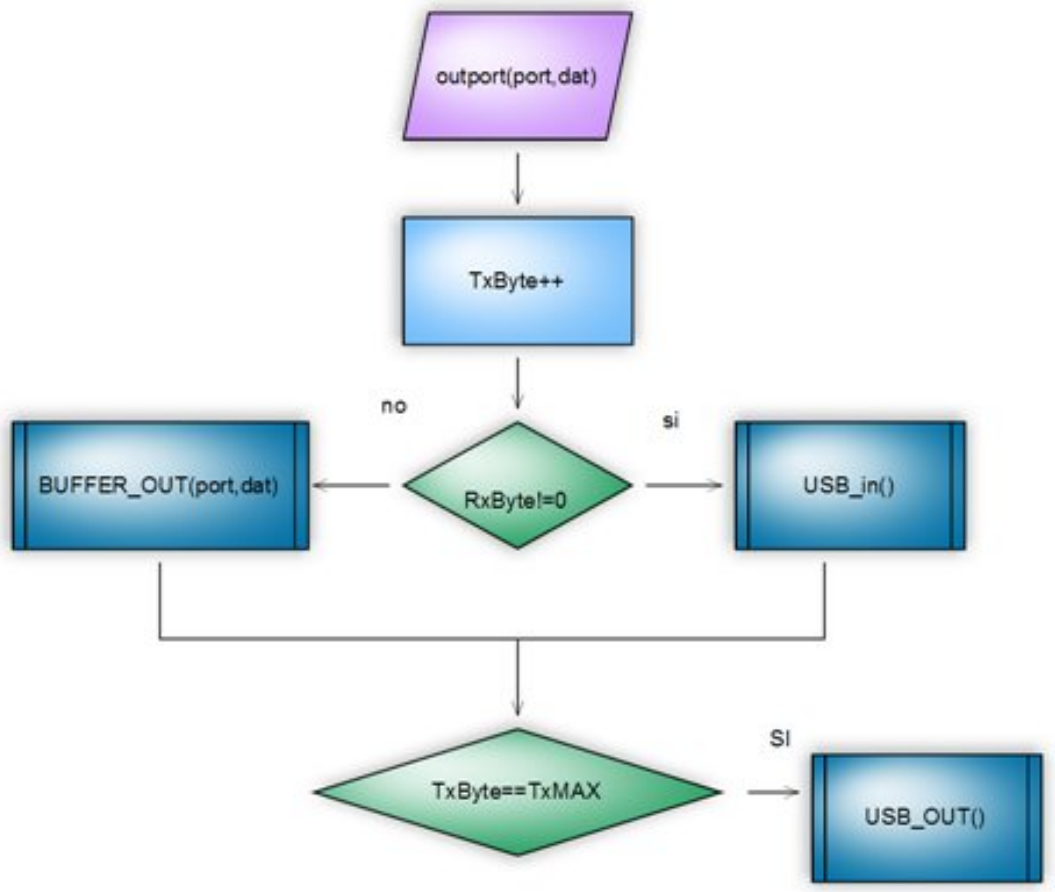


Figura 4.35: Diagrama de bloque para escritura en USB

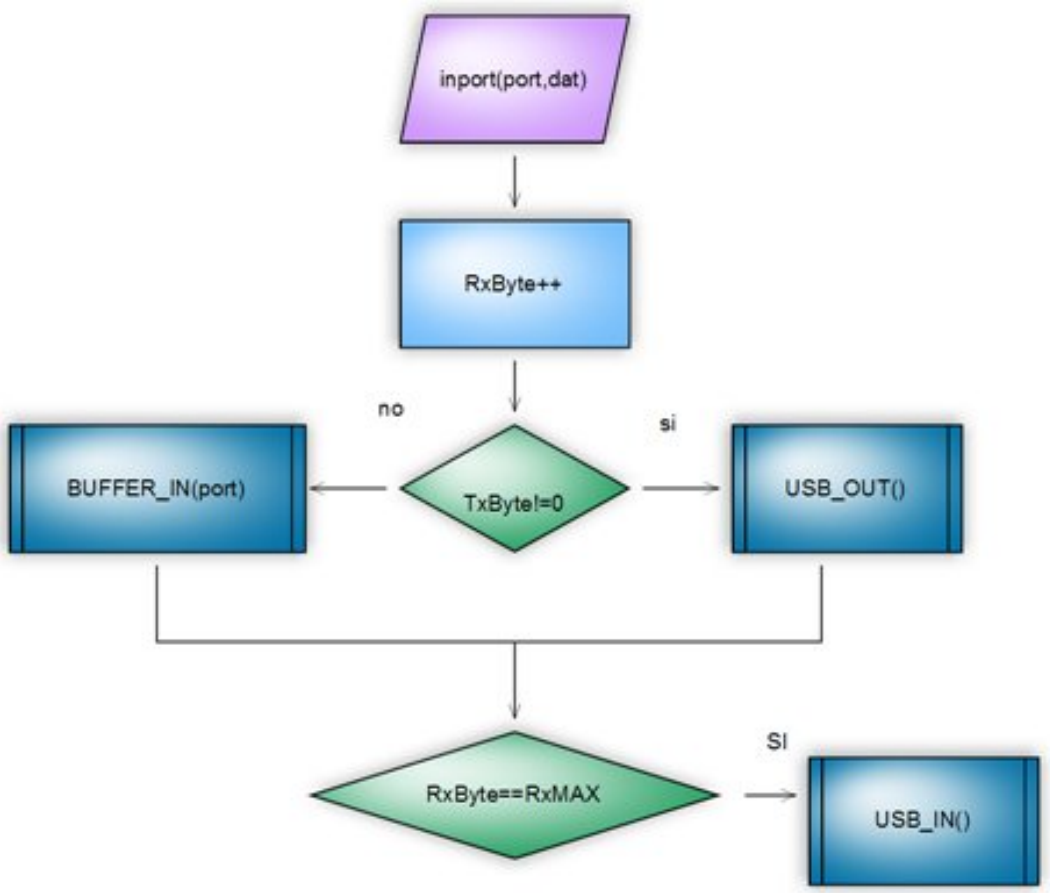


Figura 4.36: Diagrama de bloque para lectura USB

La idea consiste transmitir de forma ordenada los OUT's e IN's agrupados en bloques de transmisión de tamaño máximo. Existen dos buffers de datos donde se almacenan las instrucciones a transmitir - uno para OUT's y otro para IN's -, que cuando se llenan, se transmiten como un único bloque de instrucciones al puerto USB. El bloque de transmisión no puede empezar a trasmitir el buffer de OUT's sin haber vaciado el buffer de IN's y viceversa, para mantener el orden de transmisión. Por lo tanto, si no se ha llenado el buffer de OUT's pero se encuentra la instrucción IN, se transmite el contenido del buffer de OUT's y se almacena el IN en el buffer de IN's, y viceversa.

En el caso de ejecutar un outport lo primero es incrementar la variable llamada TxByte, que sirve para saber cuantos datos se van a introducir en el buffer antes de ser enviados, si antes de este outport se hizo un inport la variable RxByte tendrá un valor distinto de cero, lo cual indica que antes de poder llenar el buffer de salida hay que vaciar el buffer de entrada.

Proceso de escritura:

La escritura en el periférico FT2232 en el modo MCU Host Bus Emulation se realiza a través de la denominada *escritura extendida*. Para llevarla a cabo se utiliza la instrucción FT_Write con la particularidad de que no sólo hay que enviar la dirección y el dato sino que debido a la estructura hardware de esta configuración se distingue entre dirección alta - 8 bits - y dirección baja- 8bits -.

Para poder realizar la escritura extendida hay que crear una variable de tipo vector donde para enviar un sólo out hay que escribir la siguiente información:

0x93	dirección alta	dirección baja	dato

En esta aplicación la dirección baja no se utiliza, y su lugar solo se envía 0xFF.

Una vez creado ese vector, ya se puede enviar la información a través del FTDI mediante la instrucción FT_Write. En ella se indica el nombre del vector antes creado y el tamaño del mismo - en el caso de transmitir un OUT el tamaño es de igual 4 -.

Como se ha comentado antes el chip FTDI posee un buffer de salida en el cual se almacena la información mediante un FT_Write y se manda consecutivamente. Con esto se quiere indicar que no se emplea el mismo tiempo en transmitir OUT a OUT dedicando un FT_Write para cada uno, que utilizando un único FT_Write que los envíe consecutivamente al leer la información del vector como un único bloque de instrucciones. Si hubiese que enviar varios datos el vector tendría la forma:

vector:

0x93	dirección alta 1	0xFF	dato1	0x93	dirección alta 2	0xFF	dato2	...

Al ejecutar el FT_Write se le indica que debe transmitir la variable vector que tiene un tamaño de 'número de out·4'. En la figura 4.37 se puede comprobar como por el bus de datos van pasando los valores almacenados en el vector. El chip se encarga por sí mismo de ir poniendo cada valor en la línea correcta para que cuando las señales $\overline{SEN}$ y $\overline{IOW}$ se pongan a '0' los valores que llegan al LA sean los deseados.

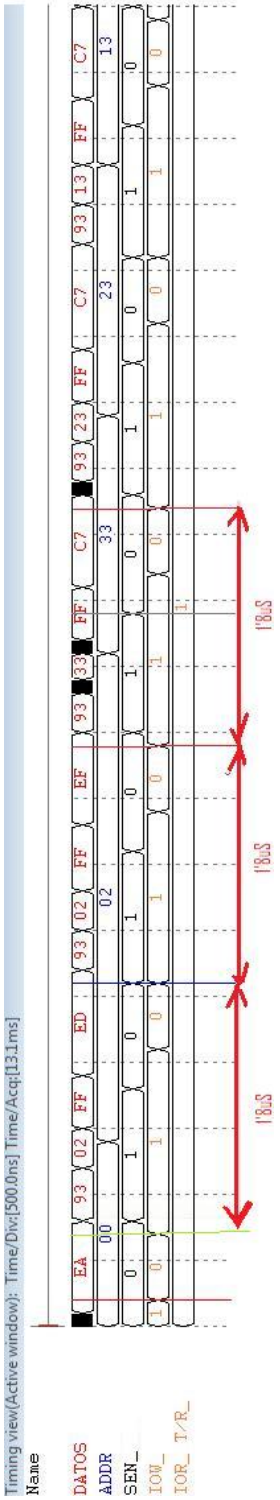


Figura 4.37: Cronograma del interface USB. Caso OUT

La rutina BUFFER_OUT(port,dato) se utiliza para ir metiendo la información en el vector de salida, de forma que cuando se llama a la rutina USB_OUT() se ejecuta un FT_Write donde se indica que debe transmitir el contenido de la variable vector con un tamaño conocido.

En el caso del interface USB se puede observar en la figura 4.37 que el tiempo necesario para realizar out consecutivos es de 1'8us, y viendo la tabla inferior se comprueba que es más lento que el interface ISA.

OUT ISA	OUT USB
1'2 us	1'8 us

Proceso de lectura:

Como ocurre en la escritura el caso de la lectura también es especial. Debido al modo en el que se trabaja la lectura se denomina *lectura extendida*. Y como ocurría antes se necesita transmitir información previa al chip antes de poder realizar un IN.

En el caso de lectura de un dato a través del FTDI primero hay que ejecutar la instrucción FT_Write leyendo un vector con la estructura siguiente:

0x91	dirección alta	dirección baja

Como se ha explicado antes dado que no se usa la dirección baja se envía el valor 0xFF.

Después de ejecutar el FT_Write es el momento de realizar un FT_Read indicando el número de lecturas , en este caso una, y donde se desea guardar.

Siguiendo con la explicación hecha para la escritura si ahora se desea leer más de un dato el vector a indicar en FT_Write sería:

vector_tx:

0x91	direccion1	0xFF	0x91	direccion2	0xFF	.....

Se ejecutaría la instrucción FT_Write indicando que se debe transmitir el vector_tx que tiene un tamaño 'n', y a continuación se ejecuta la instrucción FT_Read indicando que debe recibir 'n/3' datos y que debe guardarlos en una variable de tipo vector cuyo tamaño sea como mínimo igual a 'n/3'

En este caso la rutina BUFFER_IN(port) se encarga de ir llenando la variable de tipo vector vector_tx, para después cuando se llama a USB_IN() primero ejecutar un FT_Write

que transmita la información de vector_tx y a continuación se leen datos utilizando FT_Read.

Para el caso de leer la información relativa a los pods se ha creado un rutina distinta llamada USB_inPods en la que el tamaño del vector de entrada es de 8K de forma que al realizar una secuencia de IN de ese tamaño y consecutiva se consigue que el proceso de envío de datos desde el LA al PC sea más rápido.

En la figura 4.38 se puede apreciar el tiempo necesario para poder realizar una secuencia de INs utilizando el interfaz USB.

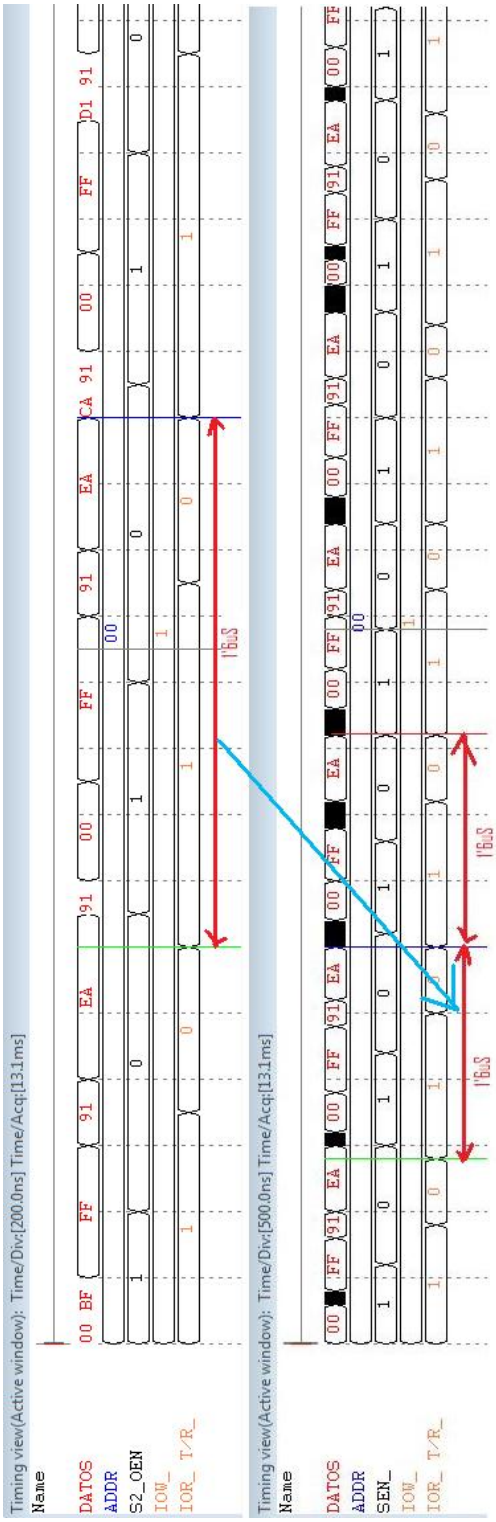


Figura 4.38: Cronograma del interface USB. Caso IN

OUT ISA	IN USB
1'1 us	1´6 us

Cuadro 4.14: Comparación tiempo de petición de datos ISA vs USB

Por otro lado al analizar el código se puede observar que en todos los ficheros que hacen referencia a la transmisión de outport e inport terminan con una llamada a la rutina USB_out_forzoso() - solo válida si PUERTO == 2 - cuya función es comprobar que se han transmitido todos los out que están almacenados en los vectores.

Para terminar con la programación del interface USB hay que analizar el posible caso en el cual el interface USB es desconectado del PC o desconectado de la alimentación, ambas situaciones provocan una pérdida del manejador del dispositivo USB. Dado que el tiempo utilizado para volver a obtener una conexión estable con el chip FTDI es superior a los 30 segundos se opto por cerrar la aplicación si ocurriese tales casos, viéndose obligado a reiniciar la misma.

4.4.5. Layout de la PCB USB

Finalmente, tras el diseño del software y del hardware necesario se realiza el diseño de la PCB. Para ello se emplea el programa Ares (Advanced Routing and Editing Software) asociado al paquete de programas de la familia Proteus. El diseño de la solución final contiene pasos específicos debido a que la fabricación de la PCB se realiza mediante una microfresadora y que la placa se introduce en una caja como acabado final.

En primer lugar se dibuja el borde de la placa en el programa, estableciendo así el tamaño de placa mínimo necesario para la fabricación de la PCB. A continuación se plasma el diseño de la caja en la placa. Se dibuja el corte que la placa precisa para poder ser introducida dentro de la caja en una capa de mecanizado. Posteriormente se dibujan los agujeros necesarios para los tornillos de sujeción a la caja.

El siguiente paso es el rutado de las pistas y la colocación de los componentes. Los componentes se distribuyen en la cara de componentes y se unen por medio de pistas según indican la líneas de conectividad. El rutado de la placa se realiza buscando la distribución más favorable de las distintas posibilidades de ubicación de los componentes, minimizando en lo posible el uso de vías y la longitud y ángulos de las pistas.

Todos los componentes se distribuyen en la cara de componentes excepto el chip FT2232, que se coloca en la cara inferior de la placa, donde se encuentran las pistas. El motivo es que el chip es de montaje superficial siendo su encapsulado de tipo LQFP-48. Es un encapsulado

muy pequeño por lo que la operación de estañado debe realizarse con extremo cuidado. Esta operación se realizó mediante un soldador de punta muy fina y con la ayuda de un microscopio de 50 aumentos.

También ha de tenerse especial cuidado en minimizar la longitud de las pistas de los componentes del oscilador para lograr una oscilación estable.

La utilización de vías se hace necesaria para que la PCB sea un diseño a una sola cara, bien por la imposibilidad de rutado o bien para salvar pistas gruesas de alimentación en el conector SCSI. Pese a la utilización de vías en el diseño, el hecho de realizar una placa a una sola cara simplifica y facilita el proceso de fabricación.

Acabado el rutado de las pistas se procede a engrosar las pistas de alimentación conforme a la corriente que se espera que circule por ellas.

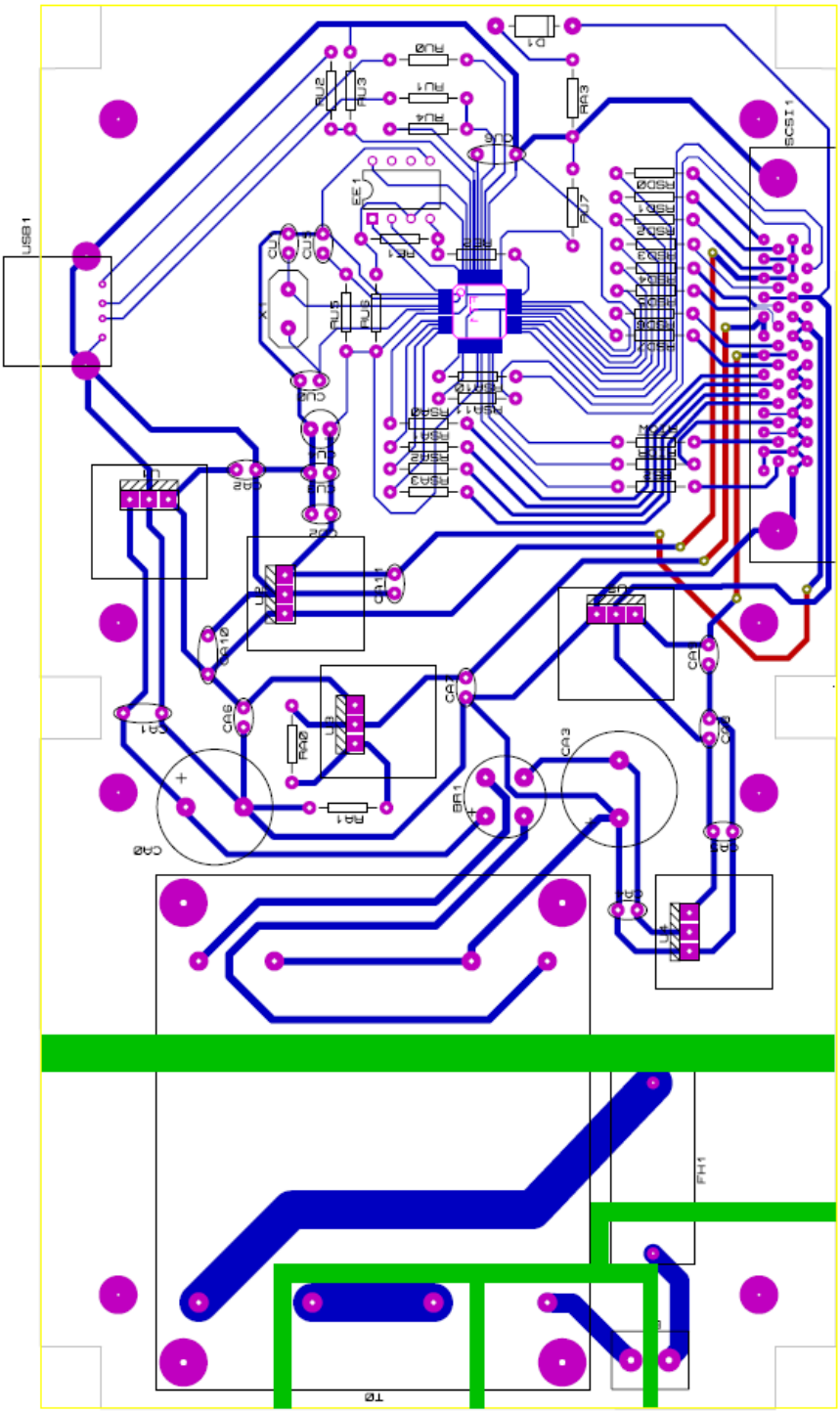


Figura 4.39: Layout PCB USB

5 Presupuesto

5.1. Coste de diseño

Durante el desarrollo del proyecto han participado dos ingenieros en régimen de jornada completa. El número de horas del proyecto se detalla para cada una de las tareas desarrolladas por los dos ingenieros.

Tarea	Descripción	Tiempo (h)	Precio (€)	Subtotal (€)
Estudio de la tarjeta de expansión del LA-4540	Desarrollo de un esquema eléctrico, estudio de los componentes de alimentación, estudio de los componentes del driver de transmisión de datos.	140	15	210
Estudio del software del fabricante	Depuración de programa `ATE.EXE` en lenguaje ensamblador, desensamblado y estudio de los programas `MAIN.EXE` y `LA.EXE`.	210	15	3150
Desarrollo de la metodología de trabajo	Análisis de las versiones de software MS-DOS y Windows, determinación de la disposición de los equipos, determinación del software de funcionamiento en los equipos y las opciones.	140	15	2100

Tarea	Descripción	Tiempo (h)	Precio (€)	Subtotal (€)
Análisis y obtención de los ficheros de protocolo	Desarrollo de una aplicación para análisis de los ficheros de protocolo, comparación de los listados obtenidos, comprensión de los comandos de código, y desarrollo de librerías para el manejo del analizador lógico.	910	15	13.650
Desarrollo de la aplicación software	Estudio de la librería de dibujo QuickCG basada en SDL, estudio del flujo de programa necesario, desarrollo de los entornos de menús, desarrollo del dibujo de las sub-ventanas, y programación de librerías de transmisión de datos.	490	15	7.350
Diseño de los prototipos PCB	Diseño de fuente de alimentación, búsqueda de componentes, diseño del prototipo PCB paralelo, diseño del prototipo PCB USB, fabricación de los prototipos, encapsulado de los prototipos.	420	15	6.300
	Subtotal horas de ingeniero	**2310**		
			TOTAL	**34.650 €**

El coste de diseño asciende a la cantidad total de **TREINTA Y CUATRO MIL SEISCIENTOS CINCUENTA EUROS**.

5.2. Costes de producción

5.2.1. Coste de materiales

En este apartado se incluyen los dispositivos electrónicos necesarios para la fuente de alimentación y para realizar cada interfaz. Se desglosa el precio para cada una de las placas de

circuito impreso por unidad de articulo. Dado que la mayoría de los componentes se compraron a través del distribuidor Farnell, los precios que se aplican son los del mismo distribuidor. Se incluye la referencia de los artículos para poder acceder a una descripción más detallada de los componentes y facilitar otros pedidos.

5.2.1.1. Presupuesto PCB paralelo

Articulo	Descripción	Referencia Farnell	Ud.	Precio	Subtotal
U1	FAIRCHILD SEMICONDUCTOR LM7812CT	1467365	1	0,29	0,29
U2	FAIRCHILD SEMICONDUCTOR LM7805ECT	1467364	1	0,29	0,29
U3	FAIRCHILD SEMICONDUCTOR LM317AHVT	1350590	1	0,44	0,44
U4	VISHAY GENERAL SEMICONDUCTOR MC7912CT	4532636	1	0,31	0,31
U5	FAIRCHILD SEMICONDUCTOR LM7905CT	1467368	1	0,34	0,34
U6	TEXAS INSTRUMENTS SN74F245N	1075920	1	0,47	0,47
U7	FAIRCHILD SEMICONDUCTOR MM74HCT273N	1014005	1	0,50	0,50
D1	LED TECHNOLOGY L02R3000F1- LED Standard	178304	1	0,31	0,31
BR1	VISHAY - W005G - Rectifier - Bridge	1336498	1	0,39	0,39
CA0, CA3	C. electrolítico 1000uF 50V	9693823	2	1,24	2,48
CA1, CA10	C. poliester 0.33uF 100V	1413787	2	0,22	0,44

Articulo	Descripción	Referencia Farnell	Ud.	Precio	Subtotal
CA2, CA6, CA11	C. electrolítico 0.1uF 63V	9452729	3	0,13	0,39
CA4, CA8	C. electrolítico 2.2uF 50V	3201650	2	0,05	0,10
CA5, CA7, CA9	C. electrolítico 1uF 50V	9693734	3	0,08	0,24
CP0, CP1	C. cerámico 100nF 50V	9411887	2	0,12	0,24
E1	Terminal Block PCB 10A	1357318	1	0,20	0,20
FH1	Portafusibles 20x5mm PCB 10A	1162741	1	0,30	0,30
Fusible	Fusible contra transitorios 200mA	1123197	1	0,26	0,26
J1	CONN-D25M	1084708	1	1,03	1,03
$R\overline{IOR}$, $R\overline{IOW}$, $R\overline{SEN}$	Resistencia 0,5W 100R	9339760	3	0,04	0,12
RA0, RA3	Resistencia 0,5W 1K	9339779	2	0,04	0,08
RA1	Resistencia 3K3	9340416	1	0,04	0,04
RSA0-RSA3, RSA10, RSA11, RSD0-RSD7	Resistencia 0,5W 47R	9340653	14	0,04	0,56
SCSI1	SCSI-D50F	1142820	1	8,79	8,79
T0	Transformador 30VA 2 x 18V - D2095	1166248	1	20,02	20,02
INTERRUPTOR	Interruptor oscilante negro	9562265	1	0,71	0,71
CONECTOR	Conector IEC Macho 10A	9523855	1	0,92	0,92
CAJA	Caja ABS Negro, 191 x 110 x 61 mm	1426575	1	8,90	8,90
CABLE IEC	Cable IEC negro, 10A, 2,5 m	1318719	1	5,15	5,15
CABE DE DATOS	Cable Paralelo, Conector A: DB25 Macho, Conector B: DB25 Hembra	1189830	1	3,70	3,70

Articulo	Descripción	Referencia Farnell	Ud.	Precio	Subtotal
MATERIAL BASE	Placa de cobre FR4, espesor 1.6mm, 233 x 160 mm	149060	1	8,52	8,52
				TOTAL	**66,53 €**

El precio total por unidad de placa de circuito impreso asciende a la cantidad de **SESENTA Y SEIS EUROS CON CINCUENTA Y TRES CÉNTIMOS DE EURO**[1].

5.2.1.2. Presupuesto PCB USB

Articulo	Descripción	Referencia Farnell	Ud.	Precio	Subtotal
U1	FAIRCHILD SEMICONDUCTOR LM7812CT	1467365	1	0,29	0,29
U2	FAIRCHILD SEMICONDUCTOR LM7805ECT	1467364	1	0,29	0,29
U3	FAIRCHILD SEMICONDUCTOR LM317AHVT	1350590	1	0,44	0,44
U4	VISHAY GENERAL SEMICONDUCTOR MC7912CT	4532636	1	0,31	0,31
U5	FAIRCHILD SEMICONDUCTOR LM7905CT	1467368	1	0,34	0,34
FT1	FTDI 2232, UART/FIFO USB, SMD, LQFP48	1615843	1	8,39	8,39
D1	LED TECHNOLOGY L02R3000F1- LED Standard	178304	1	0,31	0,31

[1]El presupuesto no incluye 30 cm de conductor de 1 mm^2ni estaño para las soldaduras.

Articulo	Descripción	Referencia Farnell	Ud.	Precio	Subtotal
BR1	VISHAY - W005G -Rectifier - Bridge	1336498	1	0,39	0,39
CA0, CA3	C. electrolítico 1000uF 50V	9693823	2	1,24	2,48
CA1, CA10	C. poliester 0.33uF 100V	1413787	2	0,22	0,44
CA2, CA6, CA11, CU5	C. electrolítico 0.1uF 63V	9452729	4	0,13	0,52
CA4, CA8	C. electrolítico 2.2uF 50V	3201650	2	0,05	0,10
CA5, CA7, CA9	C. electrolítico 1uF 50V	9693734	3	0,08	0,24
CU0, CU1	C. cerámico 27pF 100V	1138846	2	0,22	0,44
CU2, CU3	C. poliester 0.1nF 100V	1198303	2	0,21	0,42
CU4	C. electrolítico 10uF 100V	3618894	1	0,01	0,01
CU6	C. poliester 33nF 400V	1200740	1	0,21	0,21
E1	Terminal Block PCB 10A	1357318	1	0,20	0,20
FH1	Portafusibles 20x5mm PCB 10A	1162741	1	0,30	0,30
Fusible	Fusible contra transitorios 200mA	1123197	1	0,26	0,26
$R\overline{IOR}$, $R\overline{IOW}$, $R\overline{SEN}$	Resistencia 0,5W 100R	9339760	3	0,04	0,12
RA0	Resistencia 3W 4K	1155248	1	0,58	0,58
RA1	Resistencia 0,5W 12K	9339906	1	0,04	0,04
RA3	Resistencia 0,5W 1K	544590	1	0,04	0,04
RU3, RU6, RU7	Resistencia 0,5W 10K	9339787	3	0,04	0,12
RSA0-RSA3, RSA10, RSA11, RSD0-RSD7	Resistencia 0,5W47R	9340653	14	0,04	0,56
RU0, RU1	Resistencia 0,5W 27R	9340343	2	0,04	0,08
RU2	Resistencia 0,5W 4k7	9340629	1	0,04	0,04

Articulo	Descripción	Referencia Farnell	Ud.	Precio	Subtotal
RU4	Resistencia 0,5W 1K5	9339990	1	0,04	0,04
RU5	Resistencia 0,5W 470R	9338810	1	0,02	0,02
USB1	Conector USB Hembra PCB tipo A	1177883	1	0,45	0,45
X1	CRYSTAL 6MHz	1368787	1	0,94	0,94
SCSI1	SCSI-D50F	1142820	1	8,79	8,79
T0	Transformador 30VA 2 x 18V - D2095	1166248	1	20,02	20,02
INTERRUPTOR	Interruptor oscilante negro	9562265	1	0,71	0,71
CONECTOR	Conector IEC Macho 10A	9523855	1	0,92	0,92
CAJA	Caja ABS Negro 191x110x61 mm	1426575	1	8,90	8,90
CABLE IEC	Cable IEC negro, 10A, 2,5 m	1318719	1	5,15	5,15
CABLE USB	Cable USB A-A, 2m	1494745	1	2,50	2,50
MATERIAL BASE	Placa de cobre FR4, espesor 1.6mm, 233 x 160 mm	149060	1	8,52	8,52
				TOTAL	**74,92 €**

El precio total por unidad de placa de circuito impreso asciende a la cantidad de **SETENTA Y CUATRO EUROS CON NOVENTA Y DOS CÉNTIMOS DE EURO**[2].

5.2.2. Costes de fabricación y montaje

La fabricación de las placas de circuito impreso son realizadas por un operador cualificado. Las tareas que el proceso comprende son:

- Control del proceso de fresado del material base mediante una microfresadora.
- Montaje de los elementos discretos y su estañado.
- Encapsulado de la PCB en una caja.
- Test de funcionamiento del circuito.

[2]El presupuesto no incluye ni la memoria EEPROM opcional en el montaje ni las resistencias correspondientes. No se incluye tampoco 30 cm de conductor de 1 mm^2ni estaño para las soldaduras.

Tarea	Descripción	Tiempo (h)	Precio (€)	Subtotal (€)
Control del proceso de fresado	Control de los parámetros del PC y cambio de las herramientas de fresado / taladrado.	0,40	8	3,20
Montaje y estañado	Ubicación de los componentes y soldado de los mimos.	0,65	8	5,20
Encapsulado	Taladrado de los huecos del interruptor, conector de alimentación IEC, y conectores de protocolo. Montaje y cableado de la PCB en la caja.	0,15	8	1,20
Test de funcionamiento	Comprobación de las tensiones de alimentación, tensiones en los pines de los conectores, test general de funcionamiento.	0,10	8	0,80
	Subtotal horas de operario	**1,30**		
			TOTAL	**10,40 €**

El coste total de fabricación y montaje por placa de circuito impreso atribuido a un operario es de **DIEZ EUROS CON CUARENTA CÉNTIMOS DE EURO.**

5.2.3. Amortización de los costes de diseño

Los cálculos de esta sección se realizan suponiendo que la producción de placas de circuito impreso se centra en el interfaz USB 2.0. El coste de los prototipos debe amortizar el coste de diseño, más el beneficio industrial que se quiere obtener.

Para este supuesto se considera un beneficio industrial del 40 % respecto al coste de producción de las unidades fabricadas. El coste de diseño se distribuye uniformemente sobre las unidades fabricadas. El precio final de venta al público se calcula en función de la previsión de unidades fabricadas.

UNIDADES	**1**	**10**	**100**	**1000**
MATERIAL	74,92	749,20	7.492	74.920
MONTAJE	10,40	104	1.040	10.400
COSTE DE DISEÑO	34.650	34.650	34.650	34.650
SUBTOTAL DE COSTE DE PRODUCCIÓN	34.735,32 €	35503,20 €	43.182 €	119.970 €

UNIDADES	**1**	**10**	**100**	**1000**
COSTE DE PRODUCCIÓN	34.735,32	35503,20	43.182	119.970
BENEFICIO INDUSTRIAL	13.894,13	14.201,28	17.272,80	47.988
SUBTOTAL DE COSTES + BENEFICIOS	48.629,45 €	49.704,48 €	60.454,80 €	167.958 €

El precio por unidad de placa de circuito impreso se calcula como el subtotal de la suma de todos los costes más el 40 % de beneficio industrial.

UNIDADES	**1**	**10**	**100**	**1000**
PRECIO / UNIDAD	48.629,45	4.970,45	604,55	167,96
I.V.A. 16 %	7.780,71	795,27	96,73	26,87
PRECIO DE VENTA AL PÚBLICO	**56.410,16 €**	**5.765,72 €**	**701,28 €**	**194,83 €**

La última columna muestra el precio de venta al público de placa de circuito impreso con el que se logra amortizar los costes de diseño y fabricación, obteniendo un beneficio industrial del 40 % sobre unidad vendida, e incluyendo el I.V.A. del 16 %.

6 Conclusión y visión de futuro

La realización del proyecto ha requerido una gran inversión de tiempo y una gran constancia para la consecución del objetivo final, dada su complejidad y su voluminosidad. Los objetivos del proyecto que inicialmente se establecieron fueron:

1. Estudio del protocolo de funcionamiento del analizador lógico para determinar completamente su funcionamiento.
2. Desarrollo de una aplicación software para el manejo del analizador lógico.
3. Posible diseño, fabricación, y test de un prototipo PCB paralelo para constatar los progresos realizados.
4. Diseño, fabricación, y test del prototipo PCB definitivo con protocolo USB 2.0.

Inicialmente se pensaba que se disponía de un código fuente mediante el que se podría manejar el analizador lógico. Este código reduciría el proyecto al estudio del código, el desarrollo del software de manejo del analizador lógico, y al desarrollo de un hardware USB 2.0 capaz de implementar el interfaz entre el PC y el analizador lógico.

Tras el análisis inicial del código en ensamblador disponible y constatar que no era válido para manejar el analizador lógico LA-4540, se abrió una etapa de incertidumbre en la que se buscaron nuevas líneas de trabajo para lograr obtener el funcionamiento del analizador lógico. La primera etapa del proyecto debía replantearse en su totalidad, por lo que se exploraron diferentes opciones para obtener el protocolo de comunicación, lo que retrasó el proyecto hasta encontrar una línea de trabajo válida y definitiva.

La línea de trabajo definitiva es un proceso largo de trabajo muy minucioso, pero se mostró como la única línea posible. Por tanto, detrás de la escueta división inicial de etapas se esconde una gran división de tareas y operaciones no contempladas en la definición original del proyecto. Una persona externa al proyecto debe darse cuenta de la complejidad de algunas tareas, así como de la voluminosidad del trabajo necesario para abordar la totalidad del proyecto. Algunas de las tareas más relevantes de las que ha constado el proyecto han sido:

- Documentación a cerca del funcionamiento del analizador lógico.

- Estudio de la tarjeta ISA de comunicaciones: desarrollo de un esquema eléctrico, estudio de los componentes de alimentación, estudio de los componentes del driver de transmisión de datos, y estudio de las corrientes de consumo del analizador lógico.
- Estudio del software proporcionado por el fabricante: depuración del programa `ATE.EXE` en lenguaje ensamblador, desensamblado y estudio de los programas `MAIN.EXE` y `LA.EXE`.
- Desarrollo de una metodología de trabajo: análisis de las versiones de software proporcionadas por el fabricante, determinación de la disposición de los equipos, determinación del software de funcionamiento en los equipos y opciones, obtención de los primeros ficheros de protocolo.
- Análisis de los ficheros de protocolo: desarrollo de una aplicación para análisis de los ficheros de protocolo, comparación de los listados obtenidos, comprensión de los comandos de código, y desarrollo de librerías para el manejo del analizador lógico.
- Desarrollo de la aplicación software: estudio de la librería de dibujo QuickCG basada en SDL, estudio del flujo de programa necesario, desarrollo de los entornos de menús, desarrollo del dibujo de las sub-ventanas, y programación de librerías de transmisión de datos.
- Desarrollo de una fuente de alimentación: estudio de las corrientes necesarias para el funcionamiento del analizador lógico, búsqueda de componentes, diseño del prototipo de placa de circuito impreso de alimentación, elección de una caja que sirva de encapsulado a los diseños.
- Desarrollo del prototipo PCB paralelo: estudio del protocolo paralelo, estudio de la programación del puerto paralelo, diseño de la solución hardware y diseño de la simulación de protocolo ISA a través del puerto paralelo, fabricación y test del diseño de la PCB, modificaciones y diseño de la PCB final.
- Desarrollo del prototipo PCB USB 2.0: estudio del protocolo USB, estudio del circuito integrado FT2232, estudio de la programación del chip, diseño de la solución hardware, programación del protocolo de comunicación a través del puerto USB, fabricación y test del diseño de la PCB, modificaciones y diseño de la PCB final.

De todas las tareas realizadas de ingeniería inversa destacan por su volumen de trabajo las tareas relativas a la generación de las librerías de manejo del analizador. La tarea de obtención y comparación de archivos, pese al desarrollo de una aplicación para el análisis de los

ficheros de captura, ha sido especialmente trabajosa. La necesidad de comparar archivos de hasta 1500 líneas para parametrizar algunas de sus líneas mediante variables da una idea de la arduosidad del trabajo. Si a esto se añade que esta tarea debe realizarse para cada una de las opciones de funcionamiento disponibles en el software del analizador, da una idea del volumen de trabajo realizado a lo largo del proyecto. Tras el análisis de cientos de ficheros se llega a crear una librería para el manejo de algunas de las opciones disponibles, que se comprueban para obtener una realimentación en el proceso y tener la certeza de que el análisis fue correcto. La tarea es repetitiva y pesada. Paralelamente se avanza en el resto de etapas del proyecto para lograr la consecución del resto de objetivos.

A partir de los resultados obtenidos se puede afirmar que se han abordado todas las etapas del proyecto hasta conseguir un estado de éxito en cada una de ellas. La necesidad de buscar una realimentación en el proceso de ingeniería inversa nos ha llevado a ir abordando etapas siguientes del proyecto sin haber podido finalizar etapas anteriores.

El gran volumen de archivos a comparar debido a la cantidad de opciones disponibles en el software original hace que el proyecto se replantee. Se decide realizar una especificación de las opciones de manejo del analizador que se desea que funcionen a través de los prototipos de PCB a desarrollar. Estas especificaciones incluirán los usos mas comunes del analizador lógico.

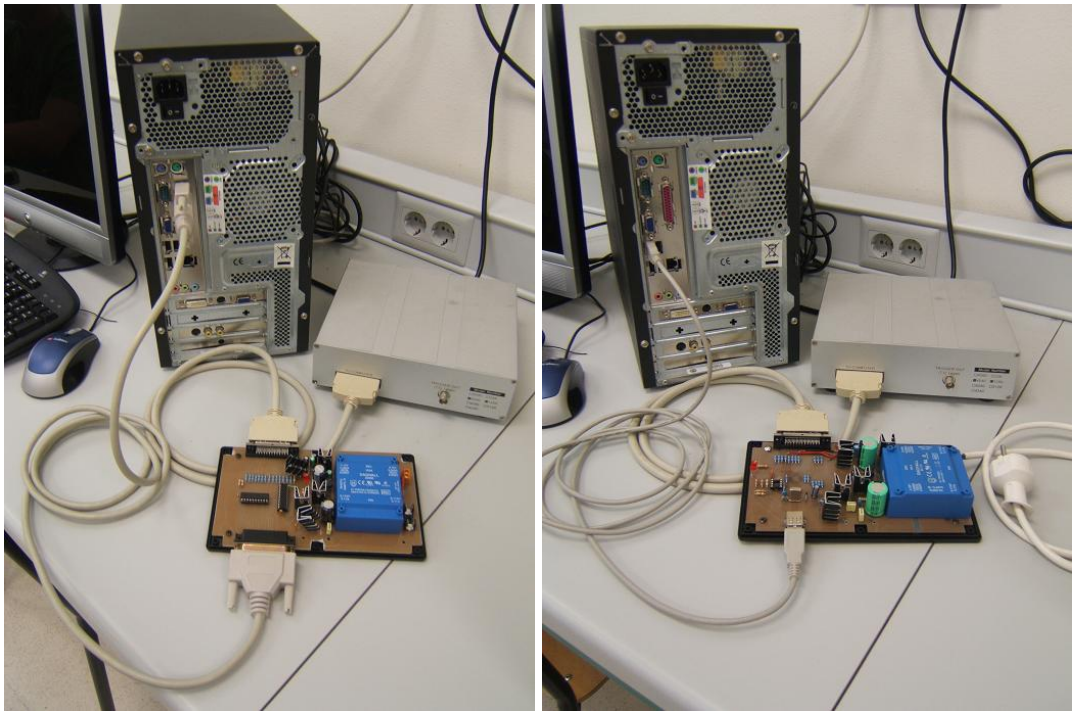
Las nuevas especificaciones del proyecto son los objetivos finales de manejo que se han logrado, y son finalmente:

1. Funcionamiento Analizador Lógico:
    - Velocidad de muestreo: 500MHz - 200Hz.
    - Modos de captura: ONCE, REPEAT ON TRIGGER, RANDOM.
    - Tamaño de captura: 128K, 8K.
    - Voltajes umbrales variables entre -6,40 V y +6,40V para los pods POD5, POD3-4, y POD1-2.
    - Trigger disponible con todas las opciones originales: trigger words, secuencia, trigger en ancho de pulso, contador de ocurrencias de trigger, y trigger true o false.
    - Función de búsqueda de un patrón de datos a través del buffer capturado.
    - Opciones de dibujo de grupos y canales.
2. Funcionamiento Generador de Patrones:

- El funcionamiento de este método esta restringido al POD2 y POD1, se muestrea por los pods POD5, POD3-4.
- Métodos de salida: CONTINUE, OUT UNTIL TRIGGER.
- Velocidad de muestreo: 100MHz - 200Hz.
- Memoria: 8K.
- Voltajes umbrales variables.

El funcionamiento como analizador lógico se ha logrado probar con éxito en las dos placas de circuito impreso desarrolladas, mientras que el funcionamiento del generador de patrones se ha comprobado mediante el protocolo ISA. Debido a su similitud con el protocolo paralelo se espera que funcione correctamente mediante la PCB paralelo. Sin embargo, el generador de patrones no funciona correctamente mediante la placa USB, lo cual podría deberse al retardo de tiempo que se provoca en el programador de tareas del S.O. al enviar pequeños paquetes de datos por el puerto USB.

El resultado de la fabricación de los prototipos se puede apreciar en la figura 6.1. Se muestra el modo de conexión de cada interfaz con un PC.



(a) Interface Paralelo (b) Interface USB

Figura 6.1: Conexión de los interfaces al PC

En la tabla 6.2se muestra una comparativa de las velocidades de transmisión logradas con cada interfaz en comparación al interfaz ISA.

Interface\Tipo de transmisión	OUT	IN
ISA	1'1 us	1'2 us
Paralelo	13 us	6'6 us
USB	1'8 us	1'6 us

Cuadro 6.2: Comparativa de velocidades de transmisión

Como conclusiones finales puede extraerse:

- Se han logrado diseñar e implementar dos placas de circuito impreso para protocolos de comunicación diferentes, puerto paralelo y USB 2.0.
- Se han sometido a tests de funcionamiento y se han mejorado los protocolos de transmisión de los datos para lograr la máxima eficiencia.
- Se ha logrado desarrollar una aplicación software que da soporte a las principales opciones de funcionamiento del software original ofrecido por el fabricante.
- El diseño PCB paralelo cumple con los requisitos de transmisión, mientras que el protocolo USB presenta algunas deficiencias debidas al programador de tareas del S.O.

Aunque el resultado del proyecto ha sido satisfactorio debe decirse que no ha llegado a su fin. Todavía sería posible continuar con su desarrollo para obtener el resto de funcionalidades no alcanzadas tras la nueva definición de especificaciones. Estas son tales como:

- El funcionamiento completo de los patrones: Uso de los 5 pods como generadores. Posibilidad de usar memoria de 128k.
- El uso de reloj externo como fuente de reloj en el manejo del analizador.
- La mejora del entorno gráfico y sus funciones.
- Posibilidad de uso del grupo de frecuencias bajas, desde 1Hz a 100Hz.

Sin embargo, cabe destacar que se han sentado las bases de trabajo para una posible mejora del funcionamiento del analizador con software propio.

A partir de lo expuesto en este documento es posible continuar en la línea de trabajo trazada para ampliar las librerías de funcionamiento de los analizadores lógicos, y siendo éste un ejemplo de como abordar un estudio de ingeniería inversa sobre cualquier dispositivo digital electrónico.

7 Anexo

En el CD adjunto al proyecto pueden encontrarse los documentos y archivos utilizados para la redacción del proyecto:

- Memoria en formato PDF.
- Carpeta de instalables:
  - port95nt.exe
  - LA_ISA.exe
  - LA_Paralelo.exe
  - LA_USB.exe
  - Drivers FTDI FT2232
- Carpeta con los datasheet de los CI utilizados para ambos diseños - paralelo y USB -.
- Carpeta con los diseños hardware.
- Carpeta con todos los archivos necesarios de la aplicación software.
- Carpeta con los manuales de ayuda de las librerías QuickCG de SDL.

Nomenclatura

AT Advanced Technology

ATE Automated Test Environment

BIOS Basic Input-Output System

BMP Windows bitmap

DLL Dynamic Linking Library

DMA Direct Memory Access

EEPROM Electrically Erasable Programmable Read Only Memory

FIFO First In First Out

ISA Industry Standard Architecture

LA Logic Analizer

LGPL Licencia Pública General Reducida de GNU

LPT Line Print Terminal

LQFP Low-profile Quad Flat Package

MCU Micro Computer Unit

MPSSE Multi-Protocol Synchronous Serial Engine

PC Personal Computer

SDL Simple DirectMedia Layer

TASM Turbo Assembler

TD Turbo Debugger

UART Universal Asynchronous Receiver-Transmitter

USB Universal Serial Bus

Bibliografía

[1] *Guide to LA4000 Series Logic Analyzer Cards Software Revision I.* Link Instruments

[2] *AN2232C-01 Command Processor for MPSSE and MCU Host Bus Emulation Modes*

[3] *AN2232-02 Bit Mode Functions for the FT2232*

[4] *FT2232C Dual USB UART / FIFO I.C.*

[5] *Windows XP Installation Guide FT2232*

[6] *Interfacing the Standard Parallel Port; http://www.senet.com.au/~cpeacock*

[7] www.linksinstrument.com

[8] www.bloodshed.net

[9] www.ftdichip.com

[10] WWW.techfest.com/hardware/bus/isa.htm

[11] www.wikipedia.org/wiki/BUS_ISA

[12] http://cfievalladolid2.net/tecno/cyr_01/control/index.htm

[13] http://www.ordenadores-y-portatiles.com/puerto-paralelo.html

[14] http://es.wikipedia.org/wiki/USB

[15] http://en.wikipedia.org/wiki/USB