

**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**  
**INGENIERÍA INFORMÁTICA**

**Diseño y construcción de un programador simple  
de memorias EEPROM y Flash de 5V**

**Realizado por**  
**Luis María Cruz Vázquez**

**Dirigido por**  
**Alberto Daza Márquez**

**Departamento**  
**Electrónica**

**UNIVERSIDAD DE MÁLAGA**

**Octubre de 2003**



**UNIVERSIDAD DE MÁLAGA**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**  
**INGENIERÍA INFORMÁTICA**

Reunido el tribunal examinador en el día de la fecha, constituido por:

Presidente Dº/Dª. \_\_\_\_\_

Secretario Dº/Dª. \_\_\_\_\_

Vocal Dº/Dª. \_\_\_\_\_

para juzgar el proyecto Fin de Carrera titulado:

\_\_\_\_\_

del alumno Dº/Dª. \_\_\_\_\_

dirigido por Dº/Dª. \_\_\_\_\_

ACORDÓ POR \_\_\_\_\_ OTORGAR LA CALIFICACIÓN DE \_\_\_\_\_

Y PARA QUE CONSTE, SE EXTIENDE FIRMADA POR LOS COMPARECIENTES DEL  
TRIBUNAL, LA PRESENTE DILIGENCIA.

Málaga, a \_\_\_\_\_ de \_\_\_\_\_ de 200\_\_\_\_

El Presidente

El Secretario

El Vocal

Fdo:

Fdo:

Fdo:



A todos aquellos que  
me rodean y me quieren.

Y especialmente a mis padres,  
sin los cuales no estaría aquí.



# Índice general

<b>Índice general</b>	<b>I</b>
<b>Índice de figuras</b>	<b>v</b>
<b>Índice de tablas</b>	<b>IX</b>
<b>I Introducción y objetivos</b>	<b>1</b>
<b>II Desarrollo</b>	<b>7</b>
<b>1. Punto de partida</b>	<b>9</b>
1.1. Programador basado en contadores . . . . .	10
1.2. Programador basado en registros de carga serie . . . . .	15
<b>2. Introducción a las memorias Flash y EPROM</b>	<b>19</b>
2.1. Tipos de memorias . . . . .	19
2.2. Memorias EPROM . . . . .	21
2.2.1. Patillaje . . . . .	21
2.2.2. Descripción funcional . . . . .	21
2.3. Memorias FLASH . . . . .	23
2.3.1. Patillaje . . . . .	23
2.3.2. Funcionamiento del bus . . . . .	25
2.3.3. Organización de las memorias . . . . .	27
2.3.4. Juegos de comandos . . . . .	28
2.3.5. Algoritmos de programación . . . . .	29

<b>3. El puerto paralelo</b>	<b>35</b>
3.1. Características físicas . . . . .	36
3.1.1. Conectores y asignación de pines . . . . .	36
3.1.2. Características eléctricas . . . . .	37
3.2. Programación del puerto paralelo. Registros hardware . . . . .	39
3.2.1. Registro de datos . . . . .	39
3.2.2. Registro de estado . . . . .	40
3.2.3. Registro de control . . . . .	40
3.2.4. Registro de control extendido . . . . .	41
3.3. Configuración del puerto paralelo en la BIOS . . . . .	41
3.4. El protocolo <i>Centronics</i> . . . . .	43
3.5. El puerto paralelo mejorado - EPP . . . . .	44
3.5.1. Señales de EPP . . . . .	44
3.5.2. El protocolo EPP . . . . .	45
<b>4. Diseño del hardware</b>	<b>51</b>
4.1. Diseño lógico del circuito . . . . .	51
4.1.1. Envío de direcciones . . . . .	51
4.1.2. Lectura de datos . . . . .	53
4.1.3. Escritura de datos . . . . .	53
4.1.4. Selección del tipo y tamaño de memoria . . . . .	55
4.1.5. Fuente de alimentación . . . . .	56
4.1.6. Cálculo del retardo . . . . .	57
4.1.7. Obtención del retardo . . . . .	62
4.1.8. Coste . . . . .	63
4.2. Diseño PCB del circuito . . . . .	65
4.2.1. Colocación de componentes . . . . .	66
4.2.2. Trazado de líneas . . . . .	66
<b>5. Fabricación y montaje de la placa</b>	<b>71</b>
<b>6. Análisis y diseño del software</b>	<b>73</b>
6.1. Análisis de requisitos. Casos de uso . . . . .	73
6.2. Análisis del software . . . . .	100
6.3. Diseño del software . . . . .	100

<b>7. Implementación del software</b>	<b>107</b>
7.1. Acceso a los puertos de E/S bajo Windows . . . . .	107
7.2. Tiempo real . . . . .	108
<b>8. Pruebas</b>	<b>111</b>
8.1. Ruido en la alimentación . . . . .	111
8.2. Comprobación del retardo . . . . .	111
8.3. Pruebas de lectura y escritura . . . . .	119
<b>9. Conclusiones y mejoras posibles</b>	<b>121</b>
9.1. Conclusiones . . . . .	121
9.2. Mejoras posibles . . . . .	122
<b>III Apéndices</b>	<b>125</b>
<b>A. Manual del usuario</b>	<b>127</b>
A.1. Conexión del programador . . . . .	127
A.2. Instalación del software . . . . .	127
A.3. Colocación de la memoria . . . . .	129
A.4. Manejo del programador . . . . .	129
A.4.1. Selección del puerto paralelo . . . . .	129
A.4.2. Selección del modelo de memoria (fig. A.4) . . . . .	129
A.4.3. Lectura de una memoria (fig. A.5) . . . . .	130
A.4.4. Grabación de una memoria (fig. A.6) . . . . .	131
A.4.5. Borrado de una memoria (fig. A.7) . . . . .	134
A.4.6. Ejecución de comandos (fig. A.8) . . . . .	134
A.5. Gestión de la base de datos de memorias . . . . .	135
A.5.1. Gestión de fabricantes (fig. A.9) . . . . .	135
A.5.2. Gestión de juegos de comandos (fig. A.10) . . . . .	136
A.5.3. Gestión de dispositivos (fig. A.11) . . . . .	137
<b>B. Diagramas del hardware</b>	<b>141</b>
B.1. Diseño esquemático final . . . . .	142
B.2. Trazado final placa PCB. Cara superior . . . . .	143
B.3. Trazado final placa PCB. Cara inferior . . . . .	144

<b>C. Diagramas de análisis del software</b>	<b>145</b>
C.1. Diagramas de casos de uso . . . . .	145
C.2. Diagramas de secuencia de casos de uso . . . . .	149
C.3. Modelo conceptual . . . . .	175
<b>D. Diagramas de diseño del software</b>	<b>177</b>
D.1. Diagrama de clases de control . . . . .	177
D.2. Diagrama de clases de interfaz . . . . .	179
D.3. Especificación del comportamiento del interfaz . . . . .	181
<b>Bibliografía</b>	<b>191</b>

# Índice de figuras

1. Fotografía de una placa base de PC donde se puede apreciar la memoria Flash . . . . .	3
2. Memoria Flash en formato PDIP28 . . . . .	4
1.1. Esquema lógico de un programador basado en contadores . . . . .	14
1.2. Esquema lógico de un programador basado en registros de carga serie . . . . .	17
2.1. Patillaje 27C16 y 27C32 . . . . .	21
2.2. Patillaje 27C64, 27C128, 27C256 y 27C512 . . . . .	22
2.3. Patillaje 27C010, 27C020 y 27C040 . . . . .	22
2.4. Cronograma de un ciclo de lectura de una EPROM . . . . .	23
2.5. Patillaje AT28C17, AT28C64, AT28C256 y AT29C256 . . . . .	25
2.6. Patillaje Am29F512, Am29F010, Am29F002, Am29F040 . . . . .	25
2.7. Cronograma de un ciclo de lectura de una Am29F040 de AMD . . . . .	26
2.8. Cronograma de una operación de grabación de una Am29F040 de AMD . . . . .	27
2.9. Juego de comandos de una Am29F040B de AMD . . . . .	29
2.10. Algoritmo de programación byte a byte . . . . .	30
2.11. Algoritmo de programación por páginas . . . . .	31
2.12. Algoritmo de detección de fin de grabación <i>data polling</i> . . . . .	32
2.13. Cronograma del algoritmo <i>data polling</i> . . . . .	32
2.14. Algoritmo de detección de fin de grabación <i>toggle bit</i> . . . . .	32
2.15. Cronograma del algoritmo <i>toggle bit</i> . . . . .	33
2.16. Algoritmo de detección de fin de grabación <i>data polling</i> para una Am29F040 . . . . .	33
3.1. Protocolo Centronics . . . . .	43
3.2. Protocolo EPP - Escritura de datos . . . . .	46
3.3. Protocolo EPP - Escritura de dirección . . . . .	46
3.4. Protocolo EPP - Lectura de datos . . . . .	47
3.5. Protocolo EPP - Lectura de dirección . . . . .	47

4.1.	Lógica de direccionamiento . . . . .	52
4.2.	Tabla de verdad y mapa de Karnaugh de <i>OE</i> . . . . .	53
4.3.	Lógica de control resultante para la lectura de un dato . . . . .	54
4.4.	Tabla de verdad y mapa de Karnaugh de <i>WE</i> . . . . .	54
4.5.	Lógica de control resultante para la escritura de un dato . . . . .	54
4.6.	<i>Glitch</i> en <i>OE</i> durante las escrituras de datos . . . . .	55
4.7.	Configuración de <i>jumpers</i> para memorias EPROM . . . . .	56
4.8.	Configuración de <i>jumpers</i> para memorias Flash . . . . .	57
4.9.	Versión inicial del programador . . . . .	58
4.10.	Programador con puertas NAND . . . . .	60
4.11.	Programador con retardo basado en inversores . . . . .	64
4.12.	Diseño final. Esquemático . . . . .	68
4.13.	Diseño final. Cara superior trazado PCB . . . . .	69
4.14.	Diseño final. Cara inferior trazado PCB . . . . .	69
6.1.	Modelo conceptual . . . . .	101
6.2.	Diagrama E-R de la base de datos . . . . .	102
6.3.	Modelo relacional de la base de datos . . . . .	102
6.4.	Clases de control . . . . .	104
6.5.	Clases de interfaz . . . . .	105
8.1.	Simetría del retardo . . . . .	114
8.2.	Retardo <i>nWait</i> respecto <i>nDataStrobe</i> . . . . .	115
8.3.	Retardo <i>OE</i> y <i>WE</i> respecto <i>nDataStrobe</i> . . . . .	116
8.4.	Efectos del retardo sobre la señal . . . . .	117
8.5.	Ciclo de escritura de dirección . . . . .	117
8.6.	Ciclos de lectura y escritura de datos . . . . .	118
8.7.	Captura ciclo de lectura de un byte . . . . .	119
8.8.	Captura ciclo de escritura de un byte . . . . .	120
9.1.	Retardo de <i>nWait</i> con monoestables . . . . .	122
A.1.	Ubicación de los <i>jumpers</i> de selección de memoria . . . . .	128
A.2.	Configuración de <i>jumpers</i> para memorias Flash . . . . .	128
A.3.	Configuración de <i>jumpers</i> para memorias EPROM . . . . .	128
A.4.	Ventana de información sobre memorias . . . . .	130
A.5.	Ventana de lectura de memorias . . . . .	131

A.6. Ventana de grabación de memorias . . . . .	132
A.7. Ventana de borrado de memorias . . . . .	134
A.8. Ventana de ejecución de comandos . . . . .	135
A.9. Ventana de edición de fabricantes . . . . .	136
A.10. Ventana de edición de juego de comandos . . . . .	137
A.11. Ventana de edición de memorias . . . . .	138



# Índice de tablas

2.1. Señales de una memoria EPROM . . . . .	21
2.2. Señales de una memoria Flash . . . . .	24
3.1. Asignación de señales en modo SPP . . . . .	37
3.2. Registro de datos . . . . .	40
3.3. Registro de estado . . . . .	40
3.4. Registro de control . . . . .	41
3.5. Registro de control extendido - ECR . . . . .	41
3.6. Modos de operación registro ECR . . . . .	42
3.7. Asignación de señales en modo EPP . . . . .	45
3.8. Registros EPP . . . . .	48
4.1. Tiempos de retardo de algunos elementos de la familia HCT . . . . .	57
4.2. Variaciones del retardo de un par de inversores producidas por un condensador	62
4.3. Combinaciones de inversores en cadena y sus retardos totales . . . . .	63
4.4. Coste de componentes . . . . .	65
8.1. Tiempos de retardo . . . . .	112
8.2. Tiempos de retardo entre <i>nDataStrobe</i> y <i>nWait</i> . . . . .	112
8.3. Tiempos de lectura y escritura en distintos equipos . . . . .	120

X

# **Parte I**

## **Introducción y objetivos**



Ordenadores (BIOS, firmware de discos duros, lectores de CD o DVD, tarjetas de red, etc), reproductores de vídeo, CD, MP3 o DVD, teléfonos, PDAs, cámaras fotográficas digitales, expendedores automáticos, paneles electrónicos de publicidad, sistemas de localización geográfica GPS, sistemas de optimización de la combustión en motores, climatizadores, electrodomésticos tales como lavadoras o lavavajillas... Existe en el mercado multitud de dispositivos gestionados por un microprocesador, cuyo software y/o datos se encuentran alojados en un chip de memoria no volátil. Si por cualquier motivo queremos reprogramar uno de estos chips (por ejemplo, para añadir nuevas funcionalidades o porque accidentalmente se haya borrado la memoria) generalmente debemos recurrir a programadores genéricos de un coste bastante elevado.

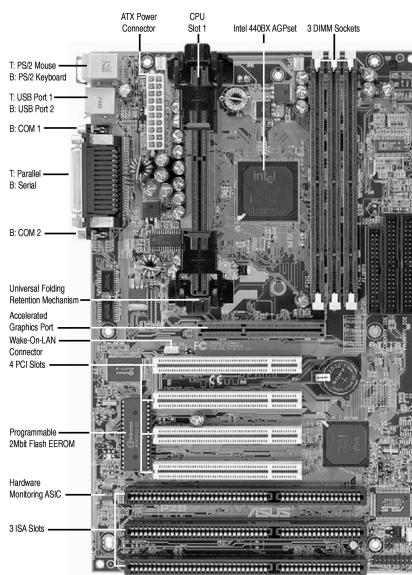


Figura 1: Fotografía de una placa base de PC donde se puede apreciar la memoria Flash

Uno de los tipos de integrados tradicionalmente más usados en este tipo de aplicaciones son las memorias Flash CMOS de 5 voltios, con encapsulado P-DIP de 24, 28 y 32 patillas, de hasta 4Mbits de capacidad y cumpliendo los estándares JEDEC. Casi todos son compatibles en modo lectura, de modo que son intercambiables entre sí. Además, el funcionamiento en modo escritura también suele ser bastante parecido, con más similitudes que diferencias, de modo que de forma relativamente simple y con el mismo hardware se puede programar una gran variedad de modelos. Asimismo puede encontrarse en el mercado bastantes memorias EPROM y EEPROM clásicas compatibles que podrían leerse con el mismo equipo.

Nos proponemos el diseño y la construcción de un programador simple de este tipo de memorias EEPROM y Flash a través del puerto paralelo de un PC, lo suficientemente genérico como para programar los integrados más usuales, manteniendo un coste reducido del hardware



Figura 2: Memoria Flash en formato PDIP28

que permita:

- Volcar el contenido actual de una memoria EPROM, EEPROM o Flash.
- Grabar y verificar una memoria Flash.
- Borrar completamente un chip de memoria.
- Hacer uso del control de bloques que ofrezca la memoria.
- Añadir soporte para nuevos modelos compatibles a nivel físico y de algoritmos de programación similares.

El software de control del programador y de gestión de la base de datos de memorias se ejecutará bajo el sistema operativo Windows en cualquiera de sus variantes (9x, ME, NT, 2000 y XP).

## Organización de esta memoria de proyecto

- En el primer capítulo se hará un estudio de algunos programadores similares al que nos proponemos realizar, cuyo diseño se puede encontrar en Internet y que usaremos como base para el desarrollo del nuestro.
- En el segundo capítulo se hará un breve repaso de la evolución de las memorias no volátiles, se presentarán las memorias EPROM y Flash y se detallarán las características físicas y el funcionamiento de éstas.
- En el tercer capítulo se verá el funcionamiento y características del puerto paralelo de los PC compatibles.
- En el cuarto capítulo se procederá al diseño del hardware de nuestro programador, tomando como base los estudiados en el primer capítulo.

## INTRODUCCIÓN Y OBJETIVOS

- El quinto capítulo detalla el proceso de fabricación y montaje de la placa del circuito del programador.
- El sexto capítulo se dedicará al análisis y diseño de la aplicación de control del programador y de gestión de la base de datos de memorias.
- En el séptimo capítulo se comentarán detalles de la implementación del software.
- En el octavo capítulo de detallarán las pruebas realizadas.
- En el noveno capítulo se plasmarán las conclusiones obtenidas del desarrollo de este proyecto y se dejará abierto el camino hacia nuevas mejoras.
- Finalmente se encuentran los apéndices, con el manual para el usuario final del programador y todos los diagramas del hardware y del análisis y diseño del software.

---

*INTRODUCCIÓN Y OBJETIVOS*

# **Parte II**

## **Desarrollo**



# Capítulo 1

## Punto de partida

Uno de los principios básicos de la ingeniería consiste en la reutilización del trabajo realizado previamente, así que en vez de embarcarnos en el diseño del programador de memorias Flash a partir de cero se hizo un estudio, con la ayuda de buscadores en Internet, de los diversos dispositivos similares que ya existieran, y que por sus características de sencillez, bajo coste y licencia de uso nos permitieran tomarlos como base para el nuestro.

Entre varias decenas de anuncios de programadores comerciales se encontraron los esquemas de un programador que Jeff Frohwein, su autor, había liberado bajo dominio público, lo que nos permitía basarnos en él y modificarlo a nuestro antojo. Este programador, que luego estudiaremos con más detalle, funciona a través del puerto paralelo de un PC y se basa en tres contadores binarios (74HC590) para generar las línea de dirección y un registro de desplazamiento (74HC595) y un multiplexor (74HC157) para las líneas de datos. Este programador, según su autor, tarda unos 15 minutos en grabar una memoria de 4Mbits (AMD Am29F040).

En su misma web, Jeff nos ofrece una versión de su programador optimizada por Sshua que, utilizando registros de desplazamiento también para las línea de dirección, consigue reducir el tiempo de programación de una flash de 4Mbits a 3 minutos. Adelantamos al lector que nuestro programador es capaz de grabar estas memorias en menos de 30 segundos en un Pentium 133.

Los esquemas lógicos, diseños de placas PCB y el software de control (para MSDOS) de ambos programadores se pueden encontrar en la página web de Jeff Frohwein[3, 4].

Veamos en detalle cada uno de estos programadores.

## 1.1. Programador basado en contadores

Este es el diseño original de Jeff Frohwein (figura 1.1). Se caracteriza por el uso de un grupo de tres integrados 74HC590 para la generación de direcciones de memoria. Éstos se componen internamente de un contador binario de 8 bits conectado a un registro también de 8 bits, con relojes independientes. Si se unen ambos relojes, como es el caso del circuito que estamos analizando, la salida del integrado va un paso por detrás del estado del contador. Cada uno de estos integrados está conectado a sendas señales de reloj *CLK0* a *CLK2*, y todos comparten una señal de *clear* común, *CLR*. Todas estas señales se vuelcan al circuito a través de las líneas de datos del puerto paralelo.

La lectura y escritura de datos de la memoria se realiza mediante el uso combinado de un registro de carga serie y un multiplexor de dos entradas de 4 bits cada una. Los datos se envían a la memoria a través del registro de carga serie, que está controlado por tres señales: *SER* (por la que se envían consecutivamente cada uno de los 8 bits de los que consta una palabra de memoria) y *CLK3* (reloj que controla la carga de estos bits), obtenidas del puerto de datos del puerto paralelo y *WOE* que permite controlar en qué momento se vuelca el dato al bus de datos de la memoria y se obtiene de las líneas de control del puerto. Los 74HC595, además del registro de carga serie, tienen un registro a la salida a modo de *latch*, controlado por la entrada *RCK*; en este circuito, esta entrada se encuentra también conectada a *CLK3*, por tanto para pasar los datos al registro de salida sólo hay que generar un pulso más de reloj (el registro de carga serie irá siempre un paso por delante que el registro de salida). La lectura de un byte de memoria se realiza en grupos de 4 bits a través de las entradas de estado del puerto paralelo gracias al multiplexor, que con la señal *U/L* (línea de datos) nos permite seleccionar la mitad más o menos significativa del byte.

Un bloque de *jumpers* permite cambiar la asignación de señales a pines concretos del zócalo PDIP32, adaptando el programador a los distintos tipos y tamaños de memorias. Una serie de *buffers* triestado pone en alta impedancia todos los pines de la memoria cuando el dispositivo está apagado para evitar dañar la memoria durante su colocación o extracción.

Para alimentar el circuito se puede usar cualquier fuente de 7.5 a 25 voltios, independientemente de si está rectificada o no, y en el caso de que lo esté, de su polaridad, gracias al regulador de tensión de 5V y el puente de diodos.

A continuación veremos paso a paso el funcionamiento del programador en las operaciones de lectura y escritura y estimaremos el número de operaciones de lectura o escritura que realiza en el puerto paralelo el programa original del autor.

#### **Lectura de una dirección de memoria**

- Se ponen a 0 los contadores. Para ello hay que generar un pulso negativo en *CLR* (2 operaciones de escritura en el puerto de datos: una para poner esta señal a 0 y otro para volver a ponerla a 1).
- Se incrementan cada uno de los contadores hasta alcanzar el valor deseado generando pulsos de las señales *CLK0* a *CLK2*. Los tres contadores se pueden ir incrementando simultáneamente, por tanto habrá que repetir este proceso un número de veces igual al mayor valor de los 3 bytes que componen una dirección de memoria. Esto supone, calculando por lo bajo, una media de 128 incrementos (256 operaciones de escritura en el registro de datos).
- Se generan nuevos pulsos en *CLK0*, *CLK1* y *CLK2* para que los valores de los contadores pasen a los *latches* de los 74HC590 (2 operaciones de escritura). Ahora la memoria ya tiene en su bus de direcciones el valor correcto.
- Se pone a 0 la señal *RD* (1 operación de escritura). A partir de este momento, la memoria localizará la dirección solicitada y la volcará al bus de datos.
- Se selecciona el *nibble* (medio byte) menos significativo del byte poniendo *U/L* a 0 (1 operación de escritura).
- Se lee ese *nibble* (1 operación de lectura).
- Se selecciona el *nibble* más significativo poniendo *U/L* a 1 (1 operación de escritura).
- Se lee el *nibble* (1 operación de lectura).
- Se vuelve a poner *RD* a 1 (1 operación de escritura). La memoria vuelve a poner su bus de datos en alta impedancia.

Esto haría una media de 266 operaciones en el puerto paralelo para leer un byte determinado de una memoria. Sin embargo, para leer posiciones consecutivas de memoria sólo debemos ir incrementando la dirección, lo que nos da una media de 7 operaciones por lectura (1 para incrementar la dirección y 6 para la lectura del byte).

#### **Escritura de una dirección de memoria<sup>1</sup>**

- Enviamos la dirección de la misma forma que antes (260 operaciones).
- Por cada uno de los 8 bits que componen el dato se establece *SER* con el valor del bit y se genera un pulso en la señal *CLK3* ( $8 \times 3 = 24$  operaciones de escritura).
- Se pasa el valor al *latch* del 74HC595 volviendo a generar otro pulso en *CLK3* (2 operaciones).
- Se activa la salida del 74HC595 poniendo *WOE* a 0 (1 operación).
- Se genera un pulso negativo en *WR* (2 operaciones).
- Se desactiva la salida del 74HC595 volviendo a poner *WOE* a 1 (1 operación).

Tenemos pues que, de media, cada escritura supone 290 operaciones. Si las escrituras fueran en direcciones consecutivas, las restantes escrituras sólo necesitarían 31 operaciones. Lamentablemente, sólo se podría aprovechar esta ventaja en la grabación de memorias que no requieran el envío de una secuencia de inicio de grabación o en memorias de grabación por páginas, que una vez enviada la secuencia, nos permiten enviar una serie de bytes consecutivos a grabar (generalmente 128 ó 256).

#### **Posibles optimizaciones a este programador**

A nivel hardware se podrían distribuir los bits de las líneas de dirección en dos grupos de 7 bits y uno de 5, en vez de los dos grupos de 8 bits y uno de 3 actuales. Esto nos permitiría reducir el número medio de incrementos necesarios para enviar una dirección a la mitad, y por tanto reduciendo el número medio de operaciones de 256 a 128.

A nivel software también se podrían realizar algunas optimizaciones. Una muy simple sería incrementar simultáneamente los contadores necesarios en la función *IncAddr*, aunque la mejora de rendimiento no sería excesivamente significativa. Otra posible mejoría sería el solapeamiento del envío de direcciones y datos que permitiría reducir en 26 ciclos las escrituras, aunque a costa de complicar bastante el código.

---

<sup>1</sup>En este caso nos referimos al envío de un par dirección-dato, no a la grabación en sí de un dato, que dependiendo del tipo de memoria requerirá el envío de varios pares dirección-dato para enviar los comandos de grabación seguidos de un último par con la dirección y dato a grabar y luego una serie de lecturas para comprobar el estado del proceso.

### *1.1. PROGRAMADOR BASADO EN CONTADORES*

---

Aplicando todas estas mejoras se reduciría el número medio de operaciones necesarias para una escritura a 136, que comparado a las 290 actuales supone una mejora del 53 %. Es decir, que de los 15 minutos necesarios para grabar una 29F040 se pasaría a unos 7 u 8.

## 1.1. PROGRAMADOR BASADO EN CONTADORES

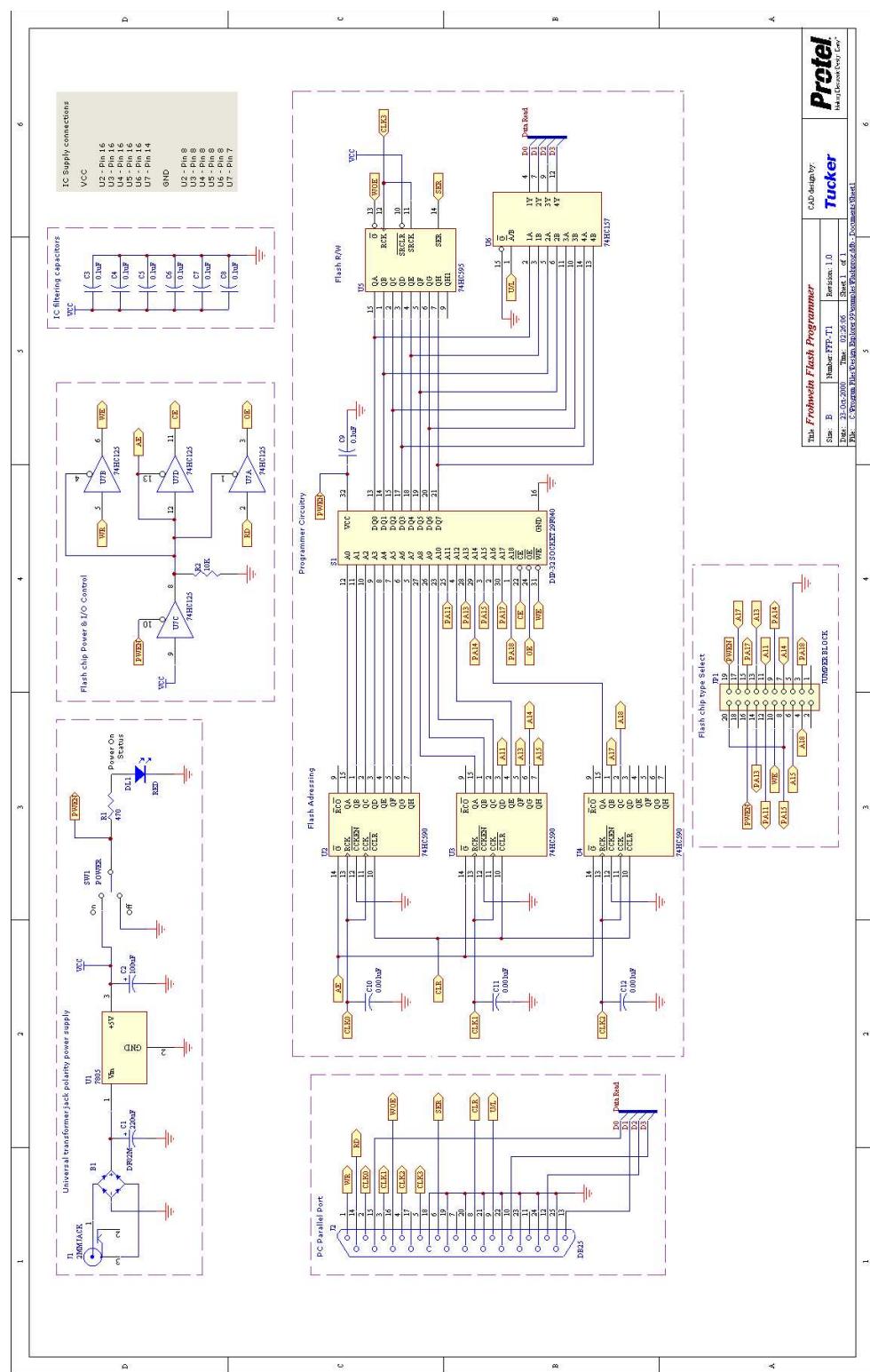


Figura 1.1: Esquema lógico de un programador basado en contadores

## 1.2. Programador basado en registros de carga serie

Este circuito es una optimización del anterior realizada por Sshua, quien pensó que el uso de contadores resulta muy interesante para el acceso secuencial a las memorias, pero que para el envío de valores dispersos, tal y como ocurre durante la programación de una memoria de grabación byte a byte que requiera el envío de secuencias de inicio de grabación, se pierde mucho tiempo poniendo a cero e incrementando los contadores hasta alcanzar los valores necesarios. Así que sustituyó los contadores por otros tres registros de carga serie 74HC595, que reciben los bits que forman los tres bytes de la dirección a través de las señales *SERO*, *SER1* y *SER2* y la señal de reloj *CLKA*. La figura 1.2 muestra el esquemático de esta versión del programador.

Veamos en detalle, como antes, las operaciones que realiza el programa ofrecido por el autor para la lectura y la escritura de una dirección de memoria.

### Lectura de una dirección de memoria

- Por cada uno de los 8 bits que forman los tres bytes que constituyen a su vez una dirección se establecen *SERO* a *SER2* con los valores de estos bits y se genera un pulso de reloj *CLKA*. (En total:  $8 \times 3 = 24$  operaciones).
- Se vuelve a generar un pulso en *CLKA* para pasar los valores a los *latches* de los 74HC595 (2 operaciones).
- Se pone a 0 la señal *RD* (1 operación de escritura). A partir de este momento, la memoria localizará la dirección solicitada y la volcará al bus de datos.
- Se selecciona el *nibble* (medio byte) menos significativo del byte poniendo *U/L* a 0 (1 operación de escritura).
- Se lee ese *nibble* (1 operación de lectura).
- Se selecciona el *nibble* más significativo poniendo *U/L* a 1 (1 operación de escritura).
- Se lee el *nibble* (1 operación de lectura).
- Se vuelve a poner *RD* a 1 (1 operación de escritura). La memoria vuelve a poner su bus de datos en alta impedancia.

Resulta ahora un total de 32 ciclos por cada lectura, independientemente del modo de acceso a la memoria (secuencial o aleatorio).

## Escritura de una dirección de memoria<sup>2</sup>

- Por cada uno de los 8 bits que forman los tres bytes de dirección y el de datos se establecen *SER0* a *SER3* con los valores de estos bits y se generan pulsos de reloj en *CLKA* y *CLKD*. (En total:  $8 \times 5 = 40$  operaciones).
- Se pasa la dirección a los *latches* generando un nuevo pulso de reloj en *CLKA* (2 operaciones).
- Se procede de igual forma con *CLKD* para pasar el dato al *latch* (2 operaciones).
- Se activa la salida del 74HC595 poniendo *WOE* a 0 (1 operación).
- Se genera un pulso negativo en *WR* (2 operaciones).
- Se desactiva la salida del 74HC595 volviendo a poner *WOE* a 1 (1 operación).

Por tanto, el programa original de los autores requiere 48 operaciones en el puerto paralelo para enviar un par dirección-dato. Vemos que se ha reducido considerablemente este número a costa, eso sí, de hacer un poco más lentas las lecturas.

## Posibles optimizaciones a este programador

A nivel hardware, una posible optimización sería repartir los bits de dirección en dos grupos de 7 bits y uno de 5, en vez de en dos grupos de 8 bits y uno de 3. Esto permitiría reducir en 3 ciclos una operación de lectura, al reducir a sólo 7 iteraciones el envío de una dirección. Sin embargo este cambio no reduciría el tiempo necesario para una escritura (porque de todas formas hay que realizar 8 iteraciones para enviar el dato) y no creemos que mereciera la pena.

El software sí que admite una optimización importante a la vez que sencilla. Solapando el envío de los relojes *CLKA* y *CLKD* durante las escrituras, se reduciría el número de operaciones necesarias en 18 ( $8 \times 2 = 16$  en el envío de la dirección y otras 2 en el paso del dato y dirección a los *latches*). Una reducción de aproximadamente un 37 % que seguramente reduciría el tiempo de grabación de una 29F040 de 3 minutos a unos 2, sin cambiar absolutamente nada en el circuito.

---

<sup>2</sup>En este caso nos referimos al envío de un par dirección-dato, no a la grabación en sí de un dato, que dependiendo del tipo de memoria requerirá el envío de varios pares dirección-dato para enviar los comandos de grabación seguidos de un último par con la dirección y dato a grabar y luego una serie de lecturas para comprobar el estado del proceso.

## 1.2. PROGRAMADOR BASADO EN REGISTROS DE CARGA SERIE

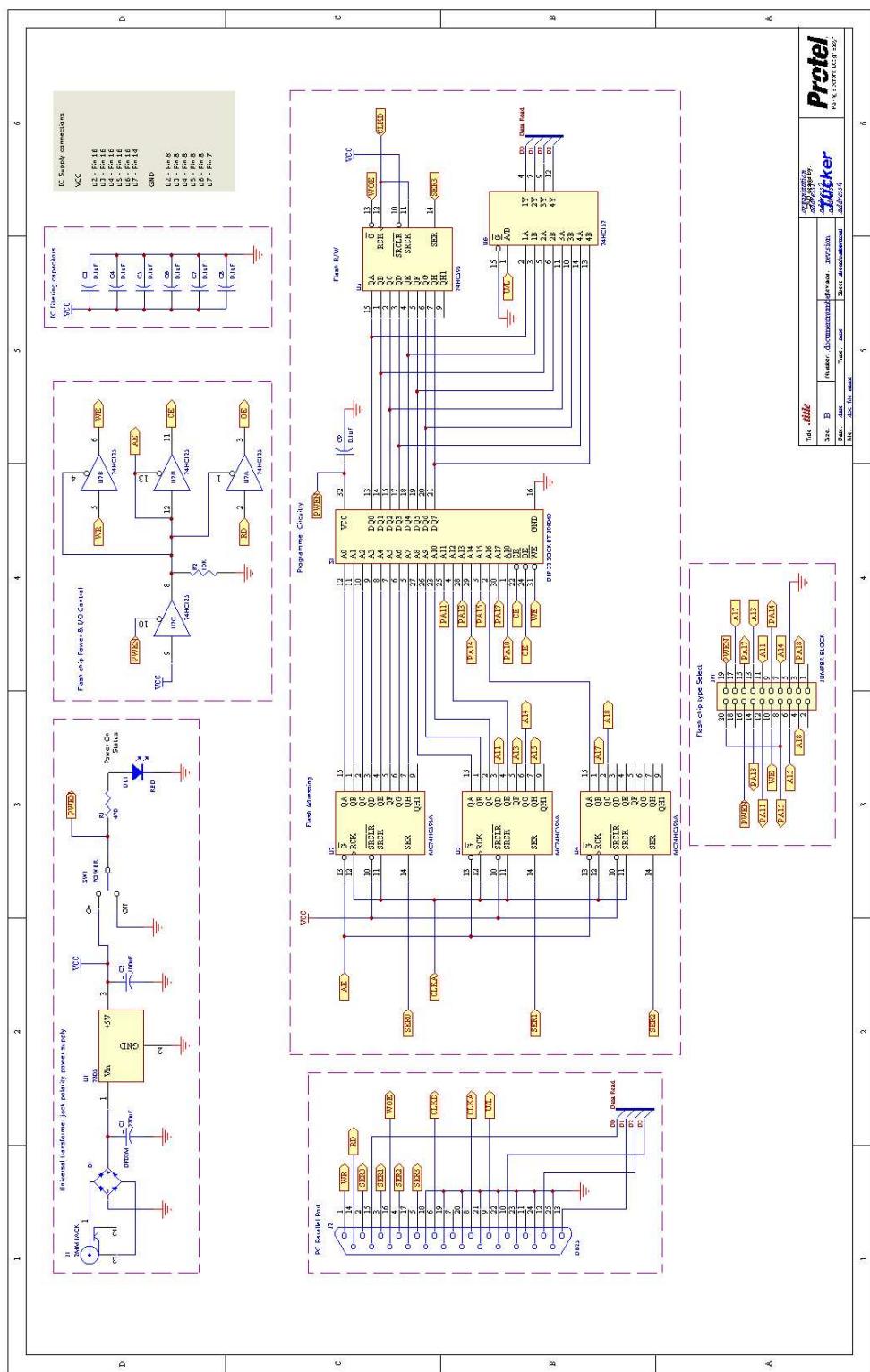


Figura 1.2: Esquema lógico de un programador basado en registros de carga serie

---

*1.2. PROGRAMADOR BASADO EN REGISTROS DE CARGA SERIE*

# Capítulo 2

## Introducción a las memorias Flash y EPROM

### 2.1. Tipos de memorias

Como vimos anteriormente, existe multitud de situaciones en las que se requiere un medio de almacenamiento no volátil (es decir, que no pierda la información al quedarse sin alimentación), rápido y que ocupe poco espacio (por lo que queda descartado el uso de unidades de disco, por ejemplo). Veamos la evolución de los principales tipos de memorias semiconductoras no volátiles a lo largo de la historia.

**ROM** Son memorias de sólo lectura. La información queda fijada durante su proceso de fabricación y no pueden modificarse nunca. Sólo resultan rentables si se va a fabricar gran cantidad de integrados con el mismo contenido, por lo que a pesar de su alta integración (gran capacidad de almacenamiento en poco espacio), han ido cayendo en desuso.

**PROM** Aparecen para cubrir la necesidad de integrados que puedan ser programados por el usuario. Sólo se podían programar una vez, utilizando para ello un aparato especial; en el dispositivo final se comportan como simples memorias ROM. Su estructura interna se basa en un transistor bipolar y una especie de fusible por celda, lo que permitía también una alta integración. Solían ser también bastante rápidas (tiempos de acceso en torno a los 50ns). La programación se realiza mediante la destrucción selectiva de los fusibles con la aplicación de voltajes de 12V o más.

**EPROM** Para la creación de prototipos y dispositivos que requieran actualización del software las memorias PROM no resultaban del todo adecuadas, ya que había que grabar un nuevo integrado y descartar el antiguo. Se hace deseable un tipo de integrado que pueda

ser reprogramado varias veces y aquí entran en escena las memorias EPROM. Basadas en tecnología CMOS, son grabadas mediante la acumulación de electrones con la aplicación de impulsos de “alto voltaje” (las primeras funcionaban a 25V, aunque luego fueron apareciendo de menor tensión, llegando a los 12V). Para borrarlas, son sometidas a radiaciones ultravioleta durante unos 10 a 30 minutos, a través de una ventanita transparente en el encapsulado. Este proceso de escritura y borrado se puede repetir unas 1000 veces. Las primeras memorias de este tipo eran bastante más lentas que las PROM bipolares, con tiempos de acceso en torno a los 200ns, aunque rápidamente la tecnología fue mejorando hasta acercarse a la velocidad de éstas, hasta el punto que las PROM bipolares desaparecieron del mercado (aún se encuentran memorias que sólo se pueden programar una vez, pero suelen ser EPROM CMOS que no traen la ventana transparente en el encapsulado, lo que permite que sean mucho más baratas).

**EEPROM** Tener que extraer la memoria del dispositivo para borrarla y reprogramarla suele ser muy tedioso, por lo que pronto aparecieron memorias que permitían ser borradas y reprogramadas eléctricamente en el mismo circuito, las EEPROM. Éstas se programan y borran aplicando impulsos eléctricos, de forma similar a las anteriores. Sin embargo resultaban caras, y la tecnología usada no permitía los niveles de integración de sus predecesoras, por lo que solían ser memorias de muy poca capacidad.

**FLASH** La evolución de la tecnología ha permitido la aparición de memorias no volátiles regrabables cada vez más rápidas, con mayor capacidad y menor consumo: las memorias FLASH. Se trata de una tecnología relativamente joven, que aún sigue evolucionando, impulsada especialmente por la aparición de gran cantidad de dispositivos móviles que requieren cada vez una mayor capacidad de almacenamiento y un mínimo consumo. Entre sus ventajas podemos destacar el funcionamiento a menor voltaje (nosotros haremos uso de las de 5V, pero han ido apareciendo memorias flash de 3V e incluso 1.8V), alta capacidad de integración (generalmente un solo transistor por celda, integrados de hasta 128Mbits), bajo coste, facilidad de programación (el proceso de programación es controlado internamente; ya no hay que andar generando pulsos hasta que la memoria quede completamente grabada), velocidad de acceso y larga vida útil (pueden reprogramarse hasta 1 millón de veces).

## 2.2. Memorias EPROM

### 2.2.1. Patillaje

Todas las memorias EPROM de la familia 27 comparten las mismas señales (a excepción de la señal  $\overline{PGM}$ , sólo presente en algunas), aunque su disposición varía dependiendo del tamaño de la memoria. La tabla 2.1 muestra una lista de estas señales y su descripción. En las figuras 2.1, 2.2 y 2.3 se muestra la distribución de estas señales para cada uno de los tamaños de memoria.

Señal	Descripción
$GND$ o $Vss$	Masa o tierra
$Vcc$ o $Vdd$	Alimentación
$Vpp$	Alimentación de programación
$DQ0-7$ ó $I/O0-7$	Líneas de datos
$A0-N$	Líneas de dirección
$\overline{CE}$ o $\overline{E}$	Chip enable. Activa o desactiva la memoria
$\overline{OE}$ o $\overline{G}$	Output enable. Activa o desactiva la salida del bus de datos
$\overline{PGM}$	Program enable. Habilita o inhabilita la escritura

Tabla 2.1: Señales de una memoria EPROM

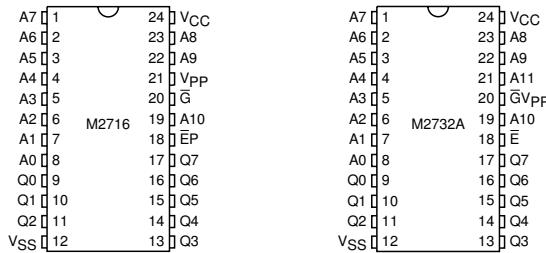


Figura 2.1: Patillaje 27C16 y 27C32

### 2.2.2. Descripción funcional

#### Modo de reposo, espera o *standby*

Cuando la entrada  $\overline{CE}$  se encuentra a nivel alto la memoria pasa a un estado bajo consumo. En este estado se ignora el resto de entradas y las salidas de datos se encuentran en alta impedancia. Cuando la entrada  $\overline{CE}$  pasa a nivel bajo, la memoria abandona el estado de bajo consumo, pasando al modo activo.

## 2.2. MEMORIAS EPROM

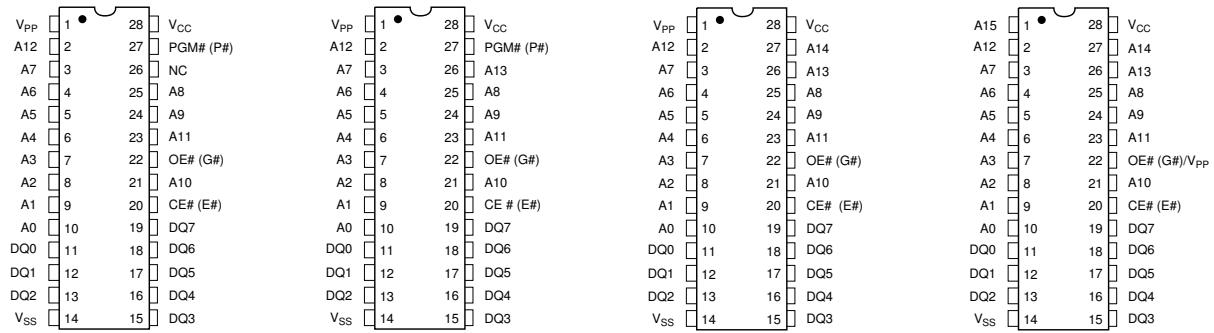


Figura 2.2: Patillaje 27C64, 27C128, 27C256 y 27C512

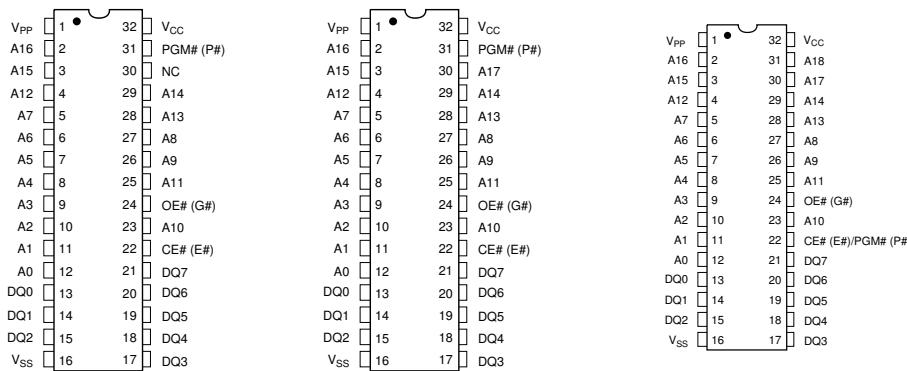


Figura 2.3: Patillaje 27C010, 27C020 y 27C040

### Modo de lectura

Estando la memoria en modo activo, una bajada del nivel lógico de  $\overline{OE}$  produce el volcado por el bus de datos del contenido de la dirección de memoria indicada en las líneas de dirección. Si esta señal se encuentra a nivel alto el bus de datos estará en alta impedancia.

La figura 2.4 muestra el cronograma de un ciclo de lectura de una memoria EPROM, extraído de la documentación técnica de una SGS-Thomson M2732A.

### Modo de escritura

Los mecanismos de escritura varían ligeramente entre modelos y fabricantes (por ejemplo, algunas requieren la generación de pulsos de escritura hasta que el dato se haya grabado correctamente, mientras que otras se graban con un solo pulso; algunas tienen una señal de control  $\overline{PGM}$  y otras no).

Además, para la escritura de estas memorias se requieren voltajes superiores a 5V (generalmente 12V, aunque algunos modelos más antiguos pueden necesitar voltajes de hasta 25V). Por motivos de sencillez, nuestro programador no va a dar soporte de escritura de memorias

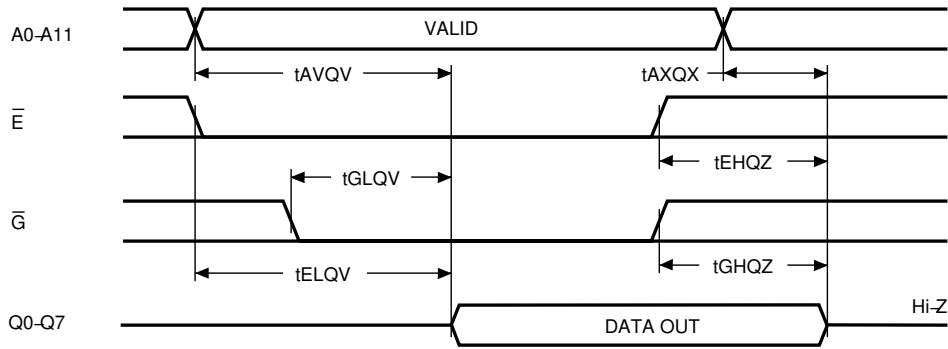


Figura 2.4: Cronograma de un ciclo de lectura de una EPROM

EPROM. Por tanto, no merece la pena entrar en más detalles en cuanto a la programación de este tipo de memorias.

## 2.3. Memorias FLASH

Veamos a continuación las características y el funcionamiento de la mayoría de memorias Flash con encapsulado PDIP que podemos encontrar en el mercado.

### 2.3.1. Patillaje

La asignación de señales a cada patilla de las memorias varía según el tamaño y el número de patillas de ésta, pero prácticamente todas tienen en común el mismo conjunto de señales (tabla 2.2). Las únicas excepciones son la señal  $\overline{RESET}$ , sólo disponible en algunas y  $V_{pp}$  sólo presente en las memorias de 12V.

Estudiando la documentación de todos los modelos de memorias flash de 5V de la siguiente lista, se observa que todos se ajustan a uno de los siguientes grupos de compatibilidad a nivel de patillaje. En las figuras 2.5 y 2.6 encontrará la disposición de señales de estos tipos más comunes de memorias flash.

- 16Kbit 24 pines: Atmel AT28C16
- 16Kbit 28 pines: Atmel AT28C17
- 64Kbit 28 pines: Atmel AT28C64, Xicor X28HC64
- 256Kbit 28 pines: Atmel AT28C256
- 256Kbit 28 pines: Atmel AT29C256

### 2.3. MEMORIAS FLASH

- 512Kbit 32 pines: Amd 29F512, Atmel AT49F512, Atmel AT29C512, SST 39SF512, SST 29EE0512, Xicor X28c512.
- 1Mbit 32 pines: Amd Am29F010B, Atmel AT28010, Atmel AT29C010A, Atmel AT49HF010, ST M29F010B, SST 29EE010, Winbond W39SF010, Xicor X28c010.
- 2Mbit 32 pines, con reset: Amd Am29F002B, Amic A29002, Atmel AT49F002, Atmel AT49F002T, Eon EN29F002, Mxic MX29F002, ST M29F002T, ST M29F002B, Winbond W49F002U, Winbond W29C020C.
- 2Mbit 32 pines, sin reset: Amd Am29F002NB, Amic A290021, Atmel AT29C020, Atmel AT49F002N, Atmel AT49F002NT, Eon EN29F002N, Mxic MX29F002T, Mxic MX29F002B, ST M29F002NT, SST29EE020, SST 39SF020, Winbond W49F020.
- 4Mbit 32 pines: Amd Am29F040B, Amic A29040, Amic A29040A, Atmel AT29C040A, Atmel AT49F040A, Bright BM29F040, Eon EN29F040, Mxic MX29F040, ST 29F040B, SST 28SF040A, Winbond W29C040, Xicor XM28C040.

A parte, tenemos una serie de memorias Flash de 12V de la familia Amd 28Fxxx y Mxic 28Fxxx, cuyo patillaje se diferencia respecto a las flash de 5V del mismo tamaño, sólo en el pin 1 que en vez de estar no conectado se corresponde con  $V_{pp}$ (caso excepcional es el de la Amd Am28F256, que es de 32 pines en lugar de los 28 de su equivalente en 5V). Según la documentación, para usar estas memorias en sólo lectura (que es nuestro caso) esta patilla puede estar conectada a tierra o a cualquier voltaje menor que  $V_{cc} + 2$ .

Señal	Descripción
$GND$ o $V_{ss}$	Masa o tierra
$V_{cc}$ o $V_{dd}$	Alimentación 5V
$DQ0-7$ ó $I/O0-7$	Líneas de datos
$A0-N$	Líneas de dirección
$\overline{CE}$	Chip enable. Activa o desactiva la memoria
$\overline{OE}$	Output enable. Activa o desactiva la salida del bus de datos
$\overline{WE}$	Write enable. Controla la escritura de datos o comandos
$RESET$	Reset. Reinicia el dispositivo y regresa al modo de lectura
$V_{pp}$	Alimentación 12V

Tabla 2.2: Señales de una memoria Flash

### **2.3. MEMORIAS FLASH**

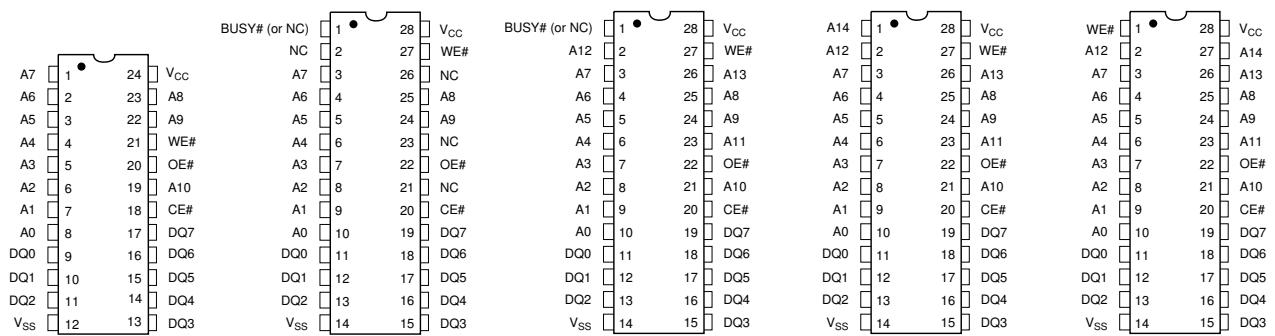


Figura 2.5: Patillaje AT28C17, AT28C64, AT28C256 y AT29C256

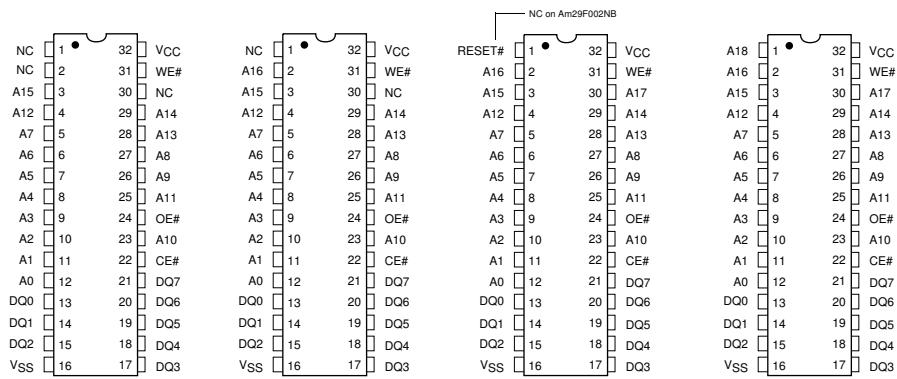


Figura 2.6: Patillaje Am29F512, Am29F010, Am29F002, Am29F040

### **2.3.2. Funcionamiento del bus**

Las memorias Flash pueden encontrarse en distintos modos de funcionamiento, dependiendo del estado en que se encuentren sus entradas. Los modos de funcionamiento más comunes son los siguientes:

### **Modo de reposo, espera o *standby***

Cuando no se está leyendo ni escribiendo en la memoria, es posible dejar ésta en modo de reposo poniendo a nivel alto la entrada  $\overline{CE}$ . En este modo se reduce considerablemente el consumo de energía y las salidas se mantienen en alta impedancia, ignorándose todas las demás entradas. Cuando  $\overline{CE}$  está a nivel bajo, la memoria se encuentra en modo activo, respondiendo al resto de entradas.

### Modo de lectura

La memoria se encuentra en este modo mientras  $\overline{CE}$  se encuentre a nivel bajo y  $\overline{WE}$  a nivel alto. Estando en este modo, y mientras  $\overline{OE}$  se encuentre a nivel alto, las salidas se mantendrán en alta impedancia. Cuando  $\overline{OE}$  pasa a nivel bajo, pasado un tiempo, la memoria mostrará por el bus de datos el valor almacenado en la dirección de memoria indicada por las líneas de dirección.

En la figura 2.7 se muestra el cronograma de un ciclo de lectura (extraído de las especificaciones de una Am29F040 de AMD).

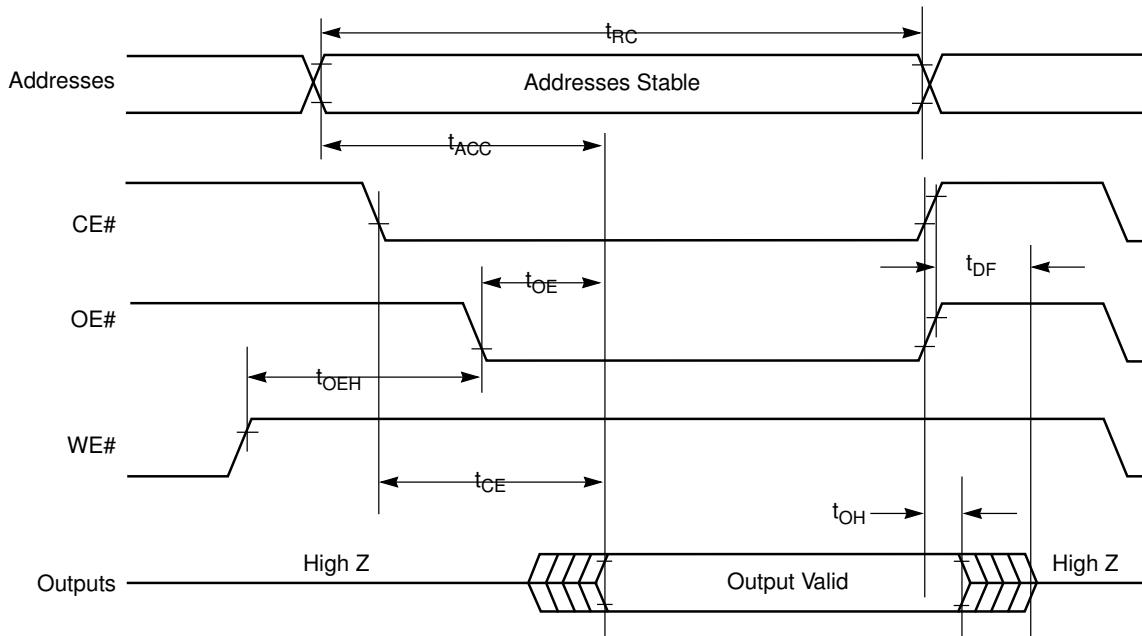


Figura 2.7: Cronograma de un ciclo de lectura de una Am29F040 de AMD

### Modo de escritura

Se usa para enviar comandos o datos a la memoria. Se inicia un ciclo de escritura mediante un pulso invertido en la entrada  $\overline{WE}$ . La memoria tomará la dirección presente en el bus de direcciones en el flanco de bajada de  $\overline{WE}$  y el valor en el bus de datos durante el flanco de subida.

En la figura 2.8 se muestra una operación de grabación de una Am29F040 de AMD. Se muestran los dos últimos ciclos de escritura (final del comando de desbloqueo, así como dirección y dato a grabar) y un par de ciclos de lectura.

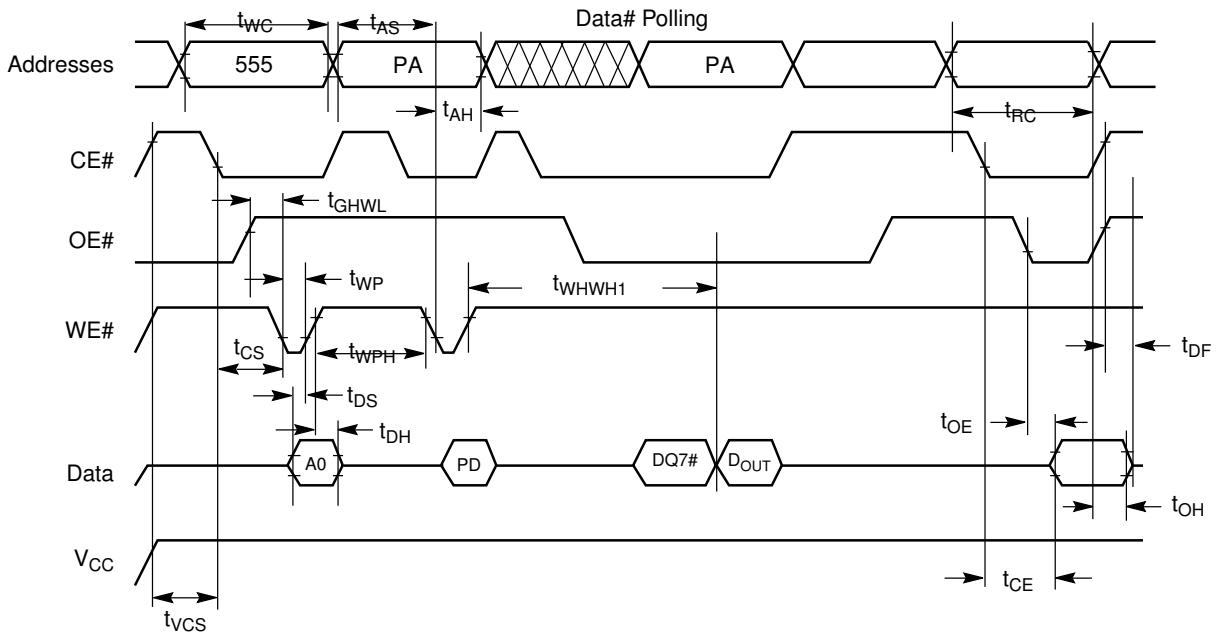


Figura 2.8: Cronograma de una operación de grabación de una Am29F040 de AMD

### Modo de autoidentificación

En este modo la memoria proporciona un código de fabricante, un código de dispositivo y, en algunas memorias, el estado de la protección de sectores. En este modo se suele entrar aplicando 12V en el pin A9 o bien, como veremos más adelante, mediante un comando. Una vez en este modo, la memoria responde con el código del fabricante cuando se le solicita la dirección *0xX00*, y con el código del modelo con la dirección *0xX01* (siendo X valores cualesquiera). Algunas memorias también ofrecen información sobre el estado de protección de los sectores al leer la dirección construida de la siguiente forma: *0xSAX02*, donde SA son los bits más significativos de la dirección del sector a consultar; la memoria nos devolverá *0x01* por el bus de datos en el caso de que el sector esté protegido y *0x00* si no lo está. Para salir de este modo se retiran los 12V de A9 o bien enviamos un comando de *Reset* (dependiendo del método usado para entrar).

### 2.3.3. Organización de las memorias

Una limitación que actualmente tienen las memorias Flash es que no se puede borrar y reescribir una sola posición de memoria, sino que debemos borrar un determinado conjunto de direcciones. Cada fabricante, e incluso cada familia de memorias de un mismo fabricante, tiene distintas formas de organizar el espacio de direcciones para minimizar este inconveniente.

Podemos encontrarnos memorias que no dividen el espacio de direcciones, por lo que

cada vez que queramos borrar una zona de memoria nos veremos forzados a borrar la memoria completa. En otras ocasiones se divide la memoria en grupos de tamaño relativamente grande (normalmente de 1 a 64KB) que reciben el nombre de sectores; cada sector puede ser borrado independientemente del resto. En ciertos modelos podemos encontrar uno o varios sectores especiales llamados *sectores de arranque*; estos sectores de arranque pueden protegerse contra escritura mientras que el resto de la memoria puede ser reescrita (el nombre de sector de arranque se debe a que generalmente esta zona se usa para almacenar el programa mínimo que es capaz de iniciar el dispositivo en el que se usa la memoria, y así poder reprogramar el resto de sectores en caso de que la información contenida en estos resultara dañada). También se encuentran memorias en las que cada uno de los sectores puede ser (des)protegido independientemente del resto.

Otra estrategia que siguen algunas familias de memorias es la división del espacio de direcciones en unidades llamadas *páginas*, generalmente más pequeñas de lo que suelen ser los sectores (128 bytes a 1KB suele ser lo habitual). Lo que distingue a este tipo de memorias es que la mínima unidad que podemos programar es una de estas páginas. Tras enviar el comando para iniciar la grabación, la memoria nos da un tiempo para enviar todos los bytes de la página que queremos grabar; tras cierto tiempo sin recibir ningún byte más, se borra automáticamente la página completa y se procede a su grabación. Aquellos bytes de la página que no hayamos escrito quedarán con el valor predeterminado tras el borrado (normalmente 0xFF - todos los bits a 1).

#### 2.3.4. Juegos de comandos

A diferencia de las PROM, EPROM y EEPROM, a las que hay que aplicar tensiones especiales para su grabación, las memorias Flash se programan aplicando las mismas tensiones que para su lectura, lo que hace posible que cualquier ruido en el circuito (por ejemplo al apagar o encender el dispositivo donde se usan) pudiera interpretarse como una grabación y alterar el contenido de cualquier posición de memoria. Para evitar esto, la mayoría de memorias Flash requieren que se les envíe un comando de desbloqueo antes de grabar un byte o una página. Un comando es una secuencia concreta de pares de datos/dirección que la memoria interpreta como una orden. Además de los comandos de desbloqueo de grabación, estas memorias suelen ofrecer otras instrucciones para el borrado de sectores o la memoria completa, autoidentificación y proteger o desproteger sectores.

La figura 2.9 muestra el conjunto de comandos de una memoria Flash Am29F040 de AMD, tal y como aparece en las especificaciones técnicas facilitadas por el fabricante.

Table 4. Am29F040B Command Definitions

Command Sequence	Bus Cycles Req'd	First Bus Read/Write Cycle		Second Bus Read/Write Cycle		Third Bus Write Cycle		Fourth Bus Read/Write Cycle		Fifth Bus Write Cycle		Sixth Bus Write Cycle	
		Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data
Read (Note 3)	1	RA	RD										
Reset (Note 4)	1	XXX	F0										
Autoselect Manufacturer ID	4	555	AA	2AA	55	555	90	X00	01				
Autoselect Device ID	4	555	AA	2AA	55	555	90	X01	A4				
Autoselect Sector Protect Verify (Note 5)	4	555	AA	2AA	55	555	90	SA X02	00				
		555	AA	2AA	55	555	90		01				
Byte Program	4	555	AA	2AA	55	555	A0	PA	PD				
Chip Erase	6	555	AA	2AA	55	555	80	555	AA	2AA	55	555	10
Sector Erase	6	555	AA	2AA	55	555	80	555	AA	2AA	55	SA	30
Erase Suspend (Note 6)	1	XXX	B0										
Erase Resume (Note 7)	1	XXX	30										

**Legend:**

RA = Address of the memory location to be read.

RD = Data read from location RA during read operation.

PA = Address of the memory location to be programmed. Addresses are latched on the falling edge of the WE# or CE# pulse.

PD = Data to be programmed at location PA. Data is latched on the rising edge of WE# or CE# pulse.

SA = Address of the sector to be erased or verified. Address bits A18–A16 uniquely select any sector.

**Notes:**

1. All values are in hexadecimal.
2. See Table 2 for descriptions of bus operations.
3. No unlock or command cycles required when device is in read mode.
4. The Reset command is required to return to the read mode when the device is in the autoselect mode or if DQ5 goes high.
5. The data is 00h for an unprotected sector and 01h for a protected sector. The complete bus address is composed of the sector address (A18–A16), A1 = 1, and A0 = 0.
6. Read and program functions in non-easing sectors are allowed in the Erase Suspend mode. The Erase Suspend command is valid only during a sector erase operation.
7. The Erase Resume command is valid only during the Erase Suspend mode.
8. Unless otherwise noted, address bits A18–A11 are don't care

Figura 2.9: Juego de comandos de una Am29F040B de AMD

### 2.3.5. Algoritmos de programación

Como vimos antes, algunas memorias se programan byte a byte y otras se programan por páginas. Cada uno de estos tipos de memoria exigirá un algoritmo de programación diferente. Pero no sólo eso, sino que los fabricantes ofrecen también distintos mecanismos para detectar el fin de la programación o borrado, o para indicar si ésta se ha realizado con éxito o ha fracasado; así nos podemos encontrar diversas combinaciones de estos mecanismos.

#### Programación byte a byte

La figura 2.10 muestra el diagrama de flujo del algoritmo básico de programación byte a byte. El proceso es el siguiente: primero se envía la secuencia de desbloqueo de programación, seguida de la dirección y datos a escribir; en este momento la memoria comienza el proceso de grabación del dato. El dispositivo debe entonces comprobar el estado del proceso, hasta que se

nos indique que la grabación ha concluido o bien que se ha producido un error. Se repite todo el proceso hasta escribir todas las direcciones que nos interesaban.

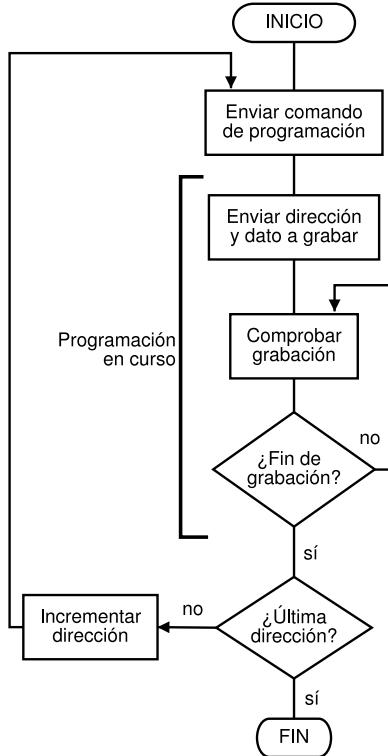


Figura 2.10: Algoritmo de programación byte a byte

### Programación por páginas

En el caso de las memorias programables por páginas, el mecanismo es similar. Primero se envía la secuencia de desbloqueo de programación y a continuación, uno tras otro, todos los bytes (pares dirección/dato) de una misma página. Cuando pasa cierto tiempo ( $Tblco$ ) sin recibir ningún dato más, la memoria ejecuta el proceso de borrado y grabación de la página; entonces el dispositivo debe esperar a que se le indique que el proceso ha concluido (figura 2.11).

¿Cómo puede detectar el dispositivo cuándo ha concluido la grabación? Pues bien, cada familia de memorias nos ofrece uno o varios de los siguientes tres métodos:

#### Esperar 10ms

Es el método más simple, pero nos hace perder mucho tiempo. Iniciamos la programación y esperamos 10ms, tras los cuales podemos asumir que ésta ha terminado. A continuación podríamos leer la posición o posiciones grabadas para asegurarnos de que están correctas.

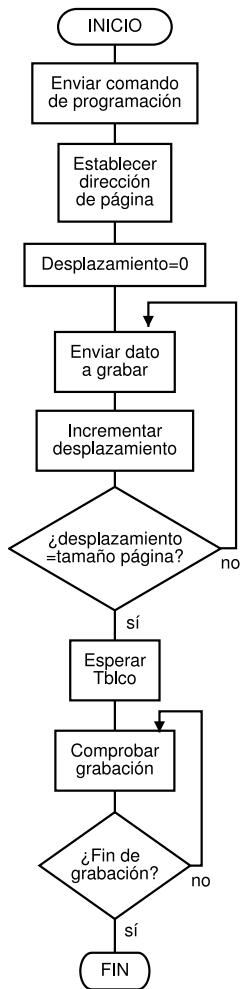


Figura 2.11: Algoritmo de programación por páginas

### Data polling

Mientras la programación está en curso, una lectura de la línea de datos *DQ7* devolverá el complemento del último dato programado en *DQ7*. Para ello el dispositivo debe proporcionar la dirección de este último dato. Una vez finalizada la programación, *DQ7* mostrará el valor programado (figuras 2.12 y 2.13).

### Toggle bit

Durante el proceso de programación, las lecturas consecutivas de *DQ6* devolverán valores alternos. Una vez finalizado, la lectura de *DQ6* devolverá el valor correspondiente a la dirección de memoria presente en el bus de direcciones (figuras 2.14 y 2.15).

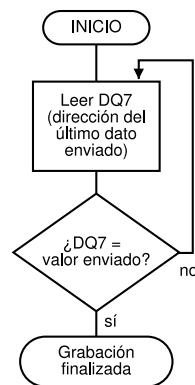


Figura 2.12: Algoritmo de detección de fin de grabación *data polling*

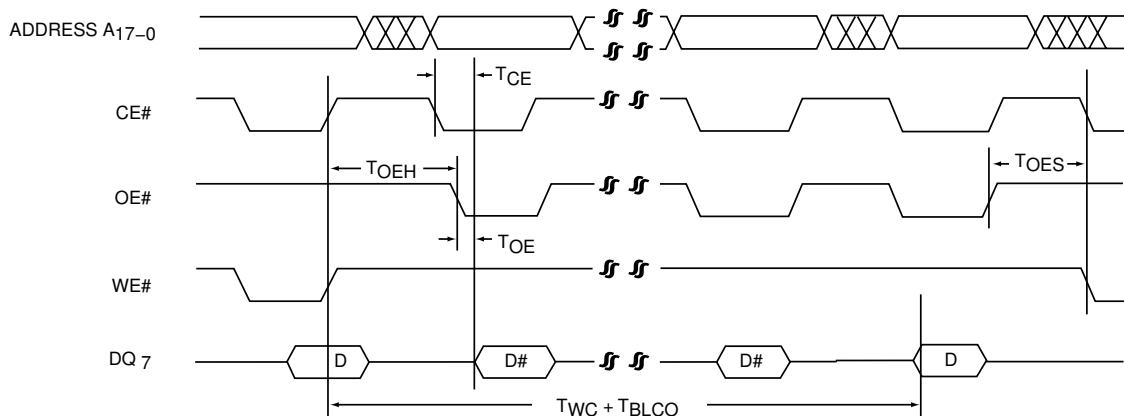


Figura 2.13: Cronograma del algoritmo *data polling*

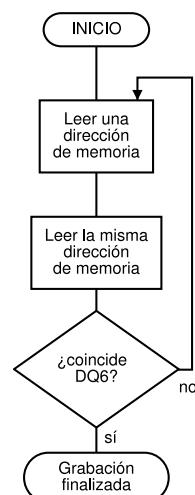
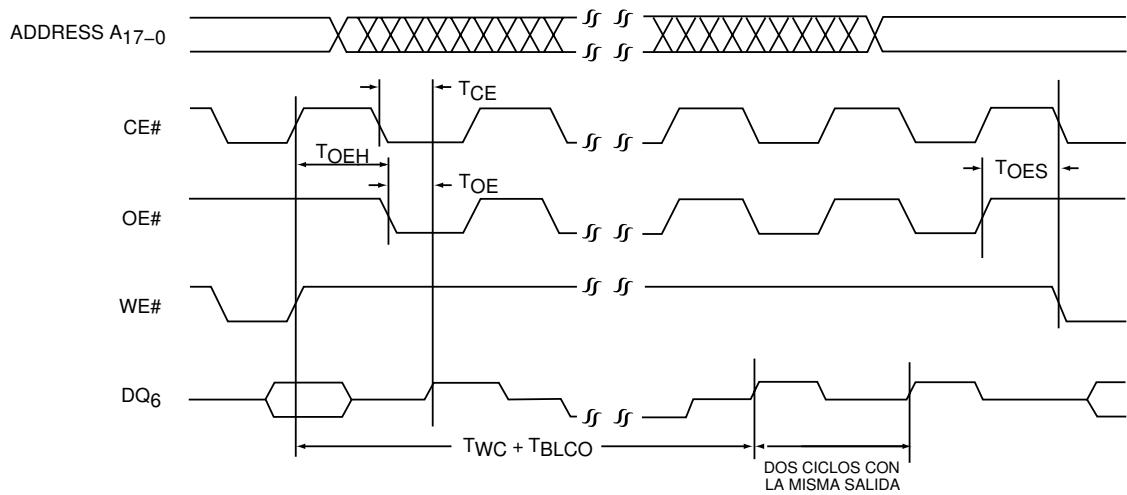
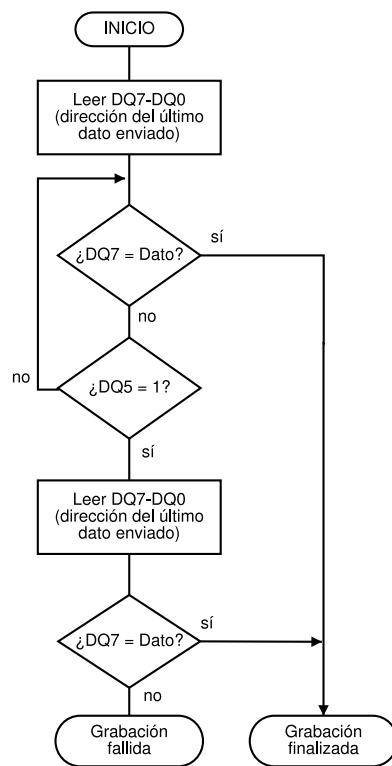


Figura 2.14: Algoritmo de detección de fin de grabación *toggle bit*


 Figura 2.15: Cronograma del algoritmo *toggle bit*

 Figura 2.16: Algoritmo de detección de fin de grabación *data polling* para una Am29F040

Nos queda ver qué ocurre cuando la grabación falla, bien porque se esté intentando grabar un 1 donde anteriormente había un 0 (cuando una posición de memoria se borra, todos sus bits quedan a 1; mediante grabaciones posteriores podemos ir estableciendo bits a 0, pero una vez a 0, la única forma de volver a grabar un 1 es mediante un borrado previo) o porque esa o esas posiciones de memoria estén dañadas.

En algunas memorias se dispone de un bit de *timeout* en la línea *DQ5*, que se pondrá a 1 cuando el contador del generador de pulsos interno llega a un determinado límite, tras el cual se considera que si la posición de memoria no se ha grabado o borrado aún, ya no conseguirá hacerlo (recordemos que en las memorias Flash todo el proceso de generación de pulsos de grabación se controla internamente). Tras una lectura positiva de *DQ5* se debe volver a comprobar si el proceso de grabación ha finalizado (*data polling* o *toggle bit*), ya que *DQ5* podría estar mostrando un 1 porque ese sea el valor programado. En la figura 2.16 tiene como ejemplo el diagrama de flujo del algoritmo *data polling* recomendado por AMD en las especificaciones técnicas de la Am29F040.

En otras simplemente el dispositivo tendrá que esperar un tiempo (generalmente en torno a los 10ms), tras el cual si la memoria sigue en proceso de grabación, puede considerar la operación fallida.

En ambos casos, hace falta enviar a la memoria un comando de *reset* para que salga de ese ciclo sin fin, regresando al modo de lectura.

# Capítulo 3

## El puerto paralelo

El puerto paralelo es un interfaz de entrada/salida que podemos encontrar en la práctica totalidad de ordenadores PC compatibles. El número de líneas de datos y control que nos ofrece se nos tornan ideales para el control de nuestro dispositivo. Por si fuera poco, el puerto paralelo funciona con niveles lógicos TTL ( $0V = 0$  lógico;  $5V = 1$  lógico), lo que nos permitirá atacar directamente nuestro circuito. Lo encontraremos generalmente en la parte trasera de nuestro ordenador en forma de conector hembra de tipo D de 25 patillas.

El estándar IEEE 1284, publicado por primera vez en 1994, define los 5 modos de funcionamiento siguientes:

1. *Modo de compatibilidad o SPP (Standard Parallel Port - Puerto Paralelo Estándar)*

Ofrece un canal de salida asíncrono de 8 bits de datos, así como un número de líneas de control y estado de acuerdo a la definición original del puerto paralelo del IBM PC.

2. *Modo nibble*

Ofrece un canal de entrada asíncrono de 4 bits de datos usando cuatro de las líneas de estado del dispositivo. Alternando este modo con el modo de compatibilidad podemos implementar un canal bidireccional, enviando los datos en grupos de 8 bits, pero recibiéndolos en dos grupos consecutivos de 4 bits.

3. *Modo byte*

Ofrece un canal de entrada asíncrono de 8 bits de datos usando para ello las 8 líneas de datos y dejando libres las líneas de control y estado para la sincronización. Alternando este modo con el modo de compatibilidad podemos implementar un canal bidireccional de 8 bits, gestionando el ordenador el sentido del mismo.

4. *Modo EPP (Enhanced Parallel Port - Puerto Paralelo Mejorado)*

Proporciona un canal bidireccional asíncrono de 8 bits, controlado por el ordenador. A

### **3.1. CARACTERÍSTICAS FÍSICAS**

---

través de las líneas de datos del puerto, permite la ejecución de ciclos de datos y de direcciones en ambos sentidos.

#### **5. Modo ECP (*Extended Parallel Port - Puerto Paralelo Extendido*)**

Proporciona un canal bidireccional asíncrono de 8 bits, controlado tanto por el ordenador como por el periférico. Una línea de control permite distinguir ciclos de datos y de direcciones. Ofrece una serie de comandos para el envío de datos comprimidos y la gestión de canales (lo que permite que varios dispositivos compartan el mismo bus).

Los modos de compatibilidad y nibble funcionan en cualquier puerto paralelo original estándar. El modo byte sólo se encontraba en ciertas tarjetas que ofrecían un puerto bidireccional. Los modos EPP y ECP requieren un hardware más moderno, fácil de encontrar en cualquier equipo PC fabricado en los últimos años (cualquier PC equipado con un Pentium o superior traerá de serie, casi con total seguridad, al menos un puerto paralelo capaz de soportar estos modos).

Mientras que en los modos de compatibilidad, nibble y byte es el software el que se debe encargar de generar e interpretar las señales de control y estado, en los modos EPP y ECP es el hardware quien se encarga de ello, reduciendo el número de operaciones de entrada/salida necesarias, lo que permite incrementar la velocidad de transferencia y reducir el consumo de tiempo de CPU. En estos modos, el puerto paralelo puede alcanzar tasas de transferencia de hasta 1 ó 2 Mbytes/s, comparadas a los aproximadamente 150Kbytes/s máximo que se alcanzan en los modos compatibles. En el modo ECP tiene además la ventaja de permitir el uso de canales DMA y colas FIFO, dejando libre el procesador mientras se transfieren los datos.

## **3.1. Características físicas**

### **3.1.1. Conectores y asignación de pines**

Como hemos comentado anteriormente, físicamente el puerto paralelo se nos muestra como un conector hembra de tipo DB de 25 patillas (el estándar IEEE1284 lo llama conector IEEE1284-A) en la parte trasera del ordenador. En el lado del periférico suele presentarse en forma de conector *Centronics* hembra de 36 patillas (IEEE1284-B). Veremos a continuación la función de cada una de las principales señales. El estándar IEEE-1284 recomienda también el uso de otro conector de 36 patillas, al que denomina IEEE1284-C, por sus ventajas en cuanto a tamaño reducido, facilidad de colocación y mejores características eléctricas, aunque al parecer

### 3.1. CARACTERÍSTICAS FÍSICAS

el mercado, por motivos de compatibilidad, se ha negado a adoptarlo y es muy raro encontrar un equipo que lo use.

La tabla 3.1 muestra la asignación de señales a las distintas patillas de los conectores, el tipo de señal (E - entrada, S - salida) y qué registros de E/S nos permite acceder a ellas. Una *n* delante del nombre de la señal indica que ésta sigue lógica inversa, es decir, la señal se considera activa cuando se encuentra en estado bajo; por ejemplo, mientras todo marcha correctamente la señal *nError* se encuentra en estado alto, pasando a estado bajo en caso de producirse algún fallo. No confunda esto con la inversión de señales que hace el hardware de 4 de las señales de control y estado (que consiste en que cuando la señal está a 1, en su respectivo bit del puerto de control o estado tenemos un 0, y viceversa).

Pin D25	Pin Centronics	Señal SPP	Dirección	Registro	Inversión hard.
1	1	nStrobe	E/S	Control	Sí
2	2	Data0	S	Datos	
3	3	Data1	S	Datos	
4	4	Data2	S	Datos	
5	5	Data3	S	Datos	
6	6	Data4	S	Datos	
7	7	Data5	S	Datos	
8	8	Data6	S	Datos	
9	9	Data7	S	Datos	
10	10	nAck	E	Status	
11	11	Busy	E	Status	Sí
12	12	Paper-Out	E	Status	
13	13	Select	E	Status	
14	14	nAuto-Linefeed	E/S	Control	Sí
15	32	nError / nFault	E	Status	
16	31	nInitialize	E/S	Control	
17	36	nSelect-In	E/S	Control	Sí
18-25	19-30	Ground	Gnd		

Tabla 3.1: Asignación de señales en modo SPP

#### 3.1.2. Características eléctricas

El interfaz opera a niveles lógicos TTL. Los niveles de voltaje varían en el rango entre 0 y +5 V, soportando picos positivos máximos de 5.5V y picos negativos de hasta -0.5V. El estándar IEEE1284 define dos tipos de dispositivos, a los que denomina de nivel 1 y de nivel 2. Los requisitos para los dispositivos de nivel 1 aseguran la compatibilidad con aquellos dispositivos

### **3.1. CARACTERÍSTICAS FÍSICAS**

---

anteriores a la definición del estándar, y los dispositivos más recientes deberían ajustarse a los requisitos de nivel 2.

#### **Dispositivos de nivel 1.**

Los dispositivos de nivel 1 controlan el interfaz con circuitos asimétricos TTL de 5V y deben ajustarse a las siguientes limitaciones:

##### **Salidas**

1. Las salidas generarán señales TTL de +5 V.
2. Una salida en circuito abierto, en estado alto no debe superar los +5.5V.
3. Una salida en circuito abierto, en estado bajo no debe bajar de los -0.5V.
4. El voltaje de salida en estado alto debe ser de al menos +2.4V ofreciendo una corriente de 0.32mA.
5. El voltaje de salida en estado bajo no debe superar los +0.4V, absorviendo una corriente de 14mA.
6. Las resistencias de pull-up, si es que las hay, deben ser al menos de  $1.8k\Omega \pm 5\%$  para la señales de control y estado y de no menos de  $1.0k\Omega \pm 5\%$  para las de datos.
7. La capacitancia sin compensar no debe superar los 50pF.

##### **Entradas**

1. Las entradas operarán con niveles TTL de +5V.
2. El punto de conmutación a estado alto no debe superar los +2.0V.
3. El punto de conmutación a estado bajo debe estar al menos en los +0.8V.
4. La entrada, excitada a nivel alto, no debe consumir más de 0.32mA a +2.0V.
5. La entrada, excitada a nivel bajo, no debe generar una intensidad de más de 12mA a +0.8V.
6. Las resistencias de pull-up, si las hay, no deben ser de menos de  $470\Omega \pm 5\%$  para las señales de control y estado, ni de menos de  $1000\Omega \pm 5\%$  para las de datos. Se recomienda que se use estos valores. Las intensidades aportadas por las resistencias de pull-up, deben tenerse en cuenta en las limitaciones de los puntos 4 y 5.

### 3.2. PROGRAMACIÓN DEL PUERTO PARALELO. REGISTROS HARDWARE

7. La capacitancia sin compensar no debe superar los 50pF. Se pueden compensar capacitancias superiores con la adición de una resistencia de pull-up, siempre y cuando cumpla los requisitos del punto 6.
8. Los tiempos máximos de subida o bajada, con un cable de 2m, serán de 120ns, medidos entre los puntos de 0.8 y 2.0V.
9. El circuito debe ser capaz de recibir picos de tensión entre los -2.0V y los +7.0V sin sufrir daños o funcionar incorrectamente.

#### **Dispositivos de nivel 2**

Los requisitos para un dispositivo de nivel 2 son demasiado exigentes para los objetivos de este proyecto y entran más en el área de la ingeniería electrónica que de la ingeniería informática, por lo que no los veremos.

## **3.2. Programación del puerto paralelo. Registros hardware**

En un sistema PC compatible podemos encontrar uno o varios puertos paralelos, a los que se les da el nombre de LPT1, LPT2, ... y así sucesivamente. Cada uno de estos puertos tiene asignado un rango en el mapa de puertos de entrada/salida, que son uno de los mecanismos que tiene el procesador central para comunicarse con su entorno. Los rangos más comunes suelen ser 0x378-0x37F y 0x278-0x27F, respectivamente para el primer y el segundo puerto paralelo, aunque pueden variar según la configuración en la BIOS del sistema o *jumpers* en la placa. Como los puertos de E/S asociados a un puerto paralelo determinado son consecutivos, nos basta con asociar cada uno de ellos con su dirección base: 0x378 y 0x278 respectivamente para LPT1 y LPT2.

Veamos cada uno de los puertos de E/S y su función.

### **3.2.1. Registro de datos**

Situado en la dirección base, se trata originalmente de un puerto de sólo escritura. En ese caso, los valores que se escriban en él serán los que aparecerán en las líneas de datos y si se realiza una lectura, se obtendrá el último valor enviado. En ordenadores más recientes puede activarse el modo bidireccional, en cuyo caso al leer del puerto obtendremos el valor que nos esté enviando el dispositivo externo. Tenga en cuenta que el sentido de la transferencia de datos no va implícito en el tipo de acceso al puerto (lectura o escritura), sino que debemos indicarlo explícitamente cambiando un bit en el puerto de control, como veremos a continuación.

### 3.2. PROGRAMACIÓN DEL PUERTO PARALELO. REGISTROS HARDWARE

---

Dirección	Tipo registro	Bit	Descripción
Base + 0	Escritura	Bit 7	Dato 7 (pin 9)
		Bit 6	Dato 6 (pin 8)
		Bit 5	Dato 5 (pin 7)
		Bit 4	Dato 4 (pin 6)
		Bit 3	Dato 3 (pin 5)
		Bit 2	Dato 2 (pin 4)
		Bit 1	Dato 1 (pin 3)
		Bit 0	Dato 0 (pin 2)

Tabla 3.2: Registro de datos

#### 3.2.2. Registro de estado

El puerto de estado es un puerto de sólo lectura; cualquier escritura será ignorada. Este registro se compone de 5 líneas de estado, un bit de estado de IRQ y de dos bits de uso reservado. Téngase en cuenta que el bit 7 es activo a nivel bajo, es decir, si el bit 7 muestra un 0 lógico, significa que a la entrada del pin 11 hay +5V, y que el bit 2 sigue lógica inversa, es decir, que si está a 1, es que no hay pendiente ninguna interrupción.

Dirección	Tipo registro	Bit	Descripción
Base + 1	Lectura	Bit 7	Busy (pin 11)
		Bit 6	Ack (pin 10)
		Bit 5	Paper out (pin 12)
		Bit 4	Select in(pin 13)
		Bit 3	Error (pin 15)
		Bit 2	IRQ (neg)
		Bit 1	Reservado
		Bit 0	Reservado

Tabla 3.3: Registro de estado

#### 3.2.3. Registro de control

El registro de control es en principio un puerto de sólo escritura. Cuando se conecta una impresora al puerto paralelo, se utilizan cuatro señales para controlar ésta: *Strobe*, *Auto Linefeed*, *Initialize* y *Select Printer*.

Sin embargo, estas cuatro salidas también pueden usarse como entradas. Están construidas con colector abierto, de forma que tienen dos estados posibles: un estado bajo (0V) y un estado de alta impedancia (circuito abierto). Para permitir la lectura de estas señales deben ponerse antes en alta impedancia, de forma que el dispositivo externo pueda gobernar su estado. Para

### 3.3. CONFIGURACIÓN DEL PUERTO PARALELO EN LA BIOS

ello, escribiremos 0100 en la mitad menos significativa del registro de control (recuerde que 3 de estas señales están invertidas por el hardware, por lo que a la salida aparecerá 1111).

Los bits 4 y 5 sirven respectivamente para activar la generación de interrupciones controladas por la línea *ACK* y para controlar el sentido de la información en el bus de datos (en puertos paralelos bidireccionales).

Dirección	Tipo registro	Bit	Descripción
Base + 2	Lectura \ Escritura	Bit 7	Sin uso
		Bit 6	Sin uso
		Bit 5	Act. bidir.
		Bit 4	Act. IRQ ACK
		Bit 3	Select printer ( <i>pin 17</i> )
		Bit 2	Initialize ( <i>pin 16</i> )
		Bit 1	Auto Linefeed ( <i>pin 14</i> )
		Bit 0	Strobe ( <i>pin 1</i> )

Tabla 3.4: Registro de control

#### **3.2.4. Registro de control extendido**

Cuando el puerto paralelo se pone en modo ECP, el sistema ofrece una serie de nuevos registros en la dirección base + 0x400. El que ahora nos atañe es el registro extendido de control (Extended Control Register - ECR), que se encuentra en la dirección base + 0x402, y en concreto nos vamos a centrar en los bits 5 a 7, que son los que nos permite que un puerto ECP emule el comportamiento de los otros modos.

Dirección	Tipo registro	Bit	Descripción
Base + 0x400	Lectura \ Escritura	Bits 5-7	Modo
		Bit 4	ECP Interrupt Bit
		Bit 3	DMA Enable Bit
		Bit 2	ECP Service Bit
		Bit 1	FIFO Full
		Bit 0	FIFO Empty

Tabla 3.5: Registro de control extendido - ECR

### **3.3. Configuración del puerto paralelo en la BIOS**

La mayoría de los puertos paralelos que podemos encontrar actualmente puede configurarse en la BIOS para funcionar de varios modos, de los cuales los más comunes son los

### 3.3. CONFIGURACIÓN DEL PUERTO PARALELO EN LA BIOS

Bits 7:5	Modo	Descripción
000	Standard Mode	Modo estándar, sin funcionalidad bidireccional
001	Byte Mode \ PS/2 Mode	Modo SPP en modo bidireccional. Usar el bit 5 para controlar el sentido
010	Parallel Port FIFO Mode	En este modo los datos escritos en la cola FIFO de datos serán enviados al dispositivo. Las señales de sincronización serán generadas por el hardware, siguiendo el estándar Centronics.
011	ECP FIFO Mode	Modo ECP propiamente dicho. Al activar este modo es posible que los registros SPP desaparezcan.
100	EPP Mode	Modo EPP
101	Reservado	
110	FIFO Test Mode	Modo de prueba
111	Configuration Mode	En este modo dos nuevos registros de configuración sustituyen a los registros de las direcciones base + 0x400 y base + 0x401

Tabla 3.6: Modos de operación registro ECR

siguientes:

**Printer Mode** A veces llamado **Default** o Normal. Es el modo más básico. Funciona como un puerto paralelo estándar, sin posibilidad de activar el modo bidireccional. El bit 5 del registro de control es ignorado.

**Standard & Bidirectional (SPP)** En este modo podremos controlar el sentido de las líneas de datos cambiando el bit 5 del registro de control.

**EPP 1.7 & SPP** En este modo tendremos acceso a los registros EPP además de a los registros SPP (datos,control y estado). Asimismo se puede acceder al bit 5 del registro de control para activar el modo bidireccional. Esta versión de EPP carece de bit de *time-out*.

**EPP 1.9 & SPP** Similar al anterior. Esta versión sí soporta el bit de *time-out* y soluciona algunos problemas de la versión 1.7, que ignoraba en ciertas ocasiones el estado de la línea *wait*.

**ECP** En este modo se tendrá acceso a todos los modos del registro extendido de control de ECP (véase apdo. [3.2.4](#)), salvo el modo EPP.

**ECP & EPP 1.7 ó 1.9** Igual que el anterior, pero con todos los modos disponibles. Como hemos visto antes, es recomendable el uso de la versión 1.9.

La mejor opción es configurar el puerto en la BIOS como **ECP & EPP** y que sea el software el que cambie el modo de funcionamiento mediante el registro de control extendido de ECP.

### 3.4. El protocolo *Centronics*

*Centronics* es un antiguo estándar para transferir datos de un ordenador a una impresora. La mayoría de las impresoras usan este mecanismo de sincronización.

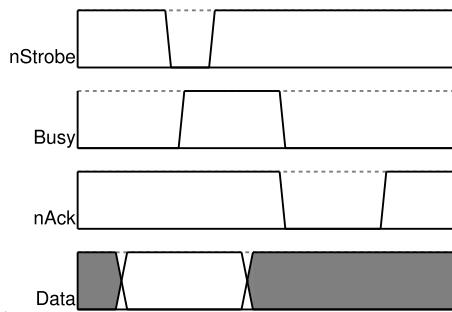


Figura 3.1: Protocolo Centronics

Como se observa en la figura 3.1, primero se vuelcan los datos en los pines 2 a 7. A continuación el sistema comprueba si la impresora está ocupada o no (es decir, comprueba que la señal *Busy* esté a nivel bajo). En caso de que esté libre, genera un pulso negativo de al menos  $1\mu s$  en la línea *nStrobe* que la impresora interpreta como el comienzo de la transmisión. A continuación la impresora indica que ahora está ocupada procesando el dato, poniendo a alto la señal *Busy*. Una vez que la impresora ha leído el dato (cosa que normalmente sucede en el flanco de subida de *nStrobe*), retira la señal *Busy* y genera un puso negativo de alrededor de  $5\mu s$  en la línea *nAck*. A menudo el sistema ignorará la línea *nAck* para ahorrar tiempo, comenzando otro ciclo inmediatamente tras la retirada de *Busy*.

Este protocolo se suele implementar por software, usando los registros del puerto paralelo estándar. Sin embargo los puertos paralelos modernos, en modo ECP, permiten que el hardware se encargue de la sincronización (véase el apdo. 3.2.4); en este caso el programador lo único que tiene que hacer es escribir el dato en el puerto de E/S. El hardware se encargará de comprobar el estado de la impresora y de generar la señal de *Strobe*. Tenga presente que en este modo tampoco se suele comprobar la señal *nAck*.

## 3.5. El puerto paralelo mejorado - EPP

El *puerto paralelo mejorado EPP* fue diseñado conjuntamente por *Intel, Xircom y Zenith Data Systems* en su primera especificación, EPP 1.7, y más tarde incluido en el estándar 1284 del IEEE, publicado en el año 1994. Posteriormente se publicó la revisión 1.9 del protocolo. Con EPP se consiguen ratios de transferencia del orden de los 500Kbytes/s a los 2MBytes/s. Esto se consigue en gran parte debido a que toda la gestión de las señales de sincronización es realizada por el hardware, en vez de hacerse por software, como era el caso del protocolo *Centronics*.

En el diseño de dispositivos simples, EPP se usa más que ECP porque todo el control lo lleva el puerto paralelo, mientras que en ECP el dispositivo externo es el que negocia el canal inverso y lleva la sincronización, lo que exige un controlador externo dedicado.

### 3.5.1. Señales de EPP

En el modo EPP la asignación de señales a cada una de las líneas del puerto es diferente de la utilizada en el modo SPP, como podemos observar en la tabla 3.7.

Las patillas 12, 13 y 15, correspondientes a las señales *PaperOut*, *Select* y *Error* del modo SPP, no tienen ninguna función predefinida en el modo EPP, quedando libres para su libre uso por parte del usuario. Su estado se puede obtener en cualquier momento a través del registro de control de SPP. Lamentablemente no queda disponible ninguna salida.

Pin	Señal SPP	Señal EPP	Entrada/Salida	Función
1	Strobe	nWrite	Salida	En estado L indica una escritura. En estado H, una lectura
2-9	Data0-7	Data0-7	Entrada/Salida	Bus de datos bidireccional
10	nAck	Interrupt	Entrada	Línea de interrupción. Produce una interrupción hardware en los flancos de subida.
11	Busy	nWait	Entrada	Usada para la sincronización. Los ciclos EPP comienzan en los flancos de bajada y acaban en los de subida.
12	PaperOut	Libre	Entrada	Libre - no usada
13	Select	Libre	Entrada	Libre - no usada
14	AutoLinefeed	nDataStrobe	Entrada	En estado L, indica una transferencia de datos
15	Error	Libre	Entrada	Libre - no usada
16	Initialize	Reset	Salida	Reset. Activa en estado L
17	SelectPrinter	nAddressStrobe	Salida	En estado L, indica transferencia de dirección
18-25	Ground	Ground	GND	Masa o tierra

Tabla 3.7: Asignación de señales en modo EPP

### 3.5.2. El protocolo EPP

El funcionamiento del protocolo EPP es transparente desde el punto de vista del software, ya que el programador sólo tiene que realizar una operación de entrada o salida al puerto correspondiente, y de toda la señalización se encarga el hardware.

EPP ofrece operaciones de lectura y escritura, distinguiendo entre datos y direcciones. Veamos con detalle cada una de ellas.

#### Ciclo de escritura de datos

1. Se pone en estado bajo la línea *nWrite* y se vuelca el dato en el bus de datos.
2. Se comprueba el estado de *nWait*; si está en estado bajo el dispositivo está libre, por tanto se puede comenzar el ciclo y se baja la línea *nDataStrobe*.
3. El sistema espera a que el dispositivo indique que está listo para capturar el dato, esperando que la línea *nWait* suba.
4. Se retira *nDataStrobe* y el dispositivo capture el dato.
5. En la versión 1.7 del protocolo, el sistema puede iniciar inmediatamente el si-

guiente ciclo. En la versión 1.9, el dispositivo tiene 125ns para retirar *nWait*, indicando que está listo para el siguiente ciclo.

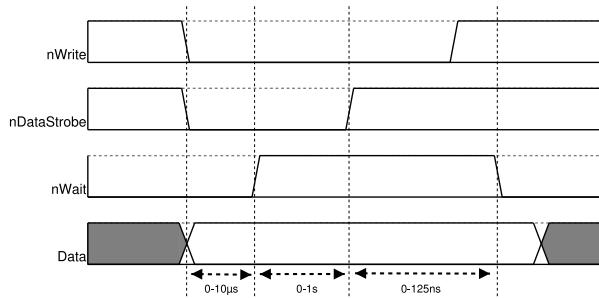


Figura 3.2: Protocolo EPP - Escritura de datos

### Ciclo de escritura de dirección

1. Se pone en estado bajo la línea *nWrite* y se vuelca la dirección en el bus de datos.
2. Se comprueba el estado de *nWait*; si está en estado bajo el dispositivo está libre, por tanto se puede comenzar el ciclo y se baja la línea *nAddrStrobe*.
3. El sistema espera a que el dispositivo indique que está listo para capturar la dirección, esperando que la línea *nWait* suba.
4. Se retira *nAddrStrobe* y el dispositivo capture la dirección.
5. En la versión 1.7 del protocolo, el sistema puede iniciar inmediatamente el siguiente ciclo. En la versión 1.9, el dispositivo tiene 125ns para retirar *nWait*, indicando que está listo para el siguiente ciclo.

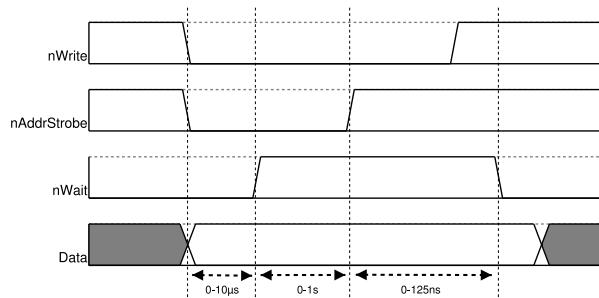


Figura 3.3: Protocolo EPP - Escritura de dirección

### Ciclo de lectura de datos

1. Se comprueba el estado de *nWait*; si está en estado bajo el dispositivo está libre, por tanto se puede comenzar el ciclo y se baja la línea *nDataStrobe*.

2. El sistema espera a que el dispositivo indique que el dato está disponible en las líneas de datos, esperando que la línea *nWait* suba.
3. El sistema captura el dato y retira *nDataStrobe*.
4. En la versión 1.7 del protocolo, el sistema puede iniciar inmediatamente el siguiente ciclo. En la versión 1.9, el dispositivo tiene 125ns para retirar *nWait*, indicando que está listo para el siguiente ciclo.

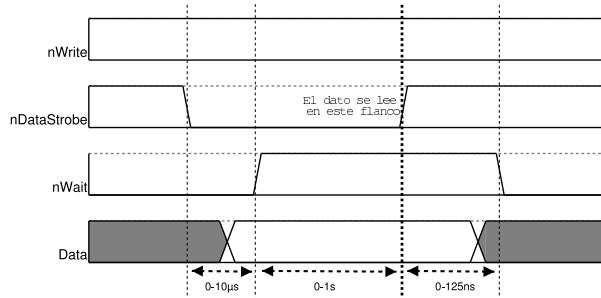


Figura 3.4: Protocolo EPP - Lectura de datos

### Ciclo de lectura de dirección

1. Se comprueba el estado de *nWait*; si está en estado bajo el dispositivo está libre, por tanto se puede comenzar el ciclo y se baja la línea *nAddrStrobe*.
2. El sistema espera a que el dispositivo indique que la dirección está disponible en las líneas de datos, esperando que la línea *nWait* suba.
3. El sistema captura la dirección y retira *nAddrStrobe*.
4. En la versión 1.7 del protocolo, el sistema puede iniciar inmediatamente el siguiente ciclo. En la versión 1.9, el dispositivo tiene 125ns para retirar *nWait*, indicando que está listo para el siguiente ciclo.

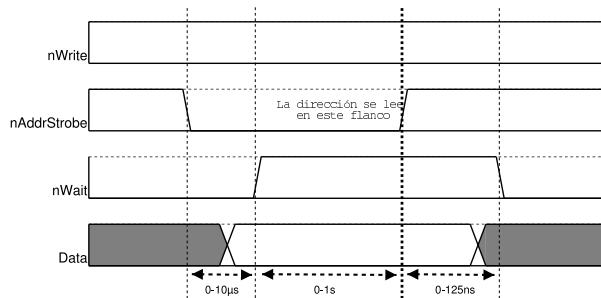


Figura 3.5: Protocolo EPP - Lectura de dirección

### Registros hardware en modo EPP

A los tres puertos del modo SPP, EPP añade una serie de nuevos registros. En la tabla 3.8 puede observar el juego completo de registros.

Dirección	Nombre del puerto	Lectura/Escritura
Base + 0	Puerto de datos (SPP)	Escritura
Base + 1	Puerto de estado (SPP)	Lectura
Base + 2	Puerto de control (SPP)	Escritura
Base + 3	Puerto de direcciones (EPP)	Lectura/escritura
Base + 4	Puerto de datos (EPP)	Lectura/escritura
Base + 5	Indefinido (transferencias 16/32 bits)	-
Base + 6	Indefinido (transferencias 32 bits)	-
Base + 7	Indefinido (transferencias 32 bits)	-

Tabla 3.8: Registros EPP

Los tres primeros puertos se comportan de forma similar a como lo hacen en modo SPP, por tanto puede comunicarse con un dispositivo *Centronics* de la misma forma que lo haría en modo SPP, llevando toda la sincronización mediante software.

Para comunicarse con un dispositivo compatible con EPP todo lo que tiene que hacer es escribir el dato en el puerto de datos EPP (dirección base + 4), y la controladora del puerto paralelo se encargará de generar y esperar todas las señales necesarias. De igual forma, para enviar una dirección, tan sólo hay que escribirla en el registro de direcciones (dirección base + 3). De forma similar, para leer un dato o una dirección, tan sólo hay que realizar una lectura del puerto correspondiente.

Observe que todo el control lo lleva siempre el puerto paralelo. Si el dispositivo quiere solicitar una transferencia de datos o direcciones debe generar una interrupción del sistema generando un flanco de subida en la línea *Interrupt*, y de alguna forma indicarle al sistema qué operación quiere realizar. El mecanismo concreto quedará a juicio del diseñador del dispositivo y su software de control (por ejemplo, podrían usarse cualquiera de las líneas de entrada libres o establecer un conjunto de códigos de operación que la rutina que atienda la interrupción se encargara de leer e interpretar).

El puerto de estado tiene una pequeña modificación. El bit 0, que en SPP estaba reservado, se convierte aquí en el bit de *time-out*. Este bit se pondrá a 1 cuando el dispositivo tarde mucho en responder; en concreto, si el dispositivo no indica estar preparado (retirando la señal *nWait*) antes de  $10\mu s$  desde que se activa *nDataStrobe* o *nAddrStrobe*.

Los otros tres registros permiten realizar transferencias de 16 o 32 bits con una sola operación de E/S si el puerto soporta esta función.

### Otras consideraciones

Antes de poder realizar escrituras o lecturas en modo EPP el puerto paralelo debe estar configurado adecuadamente. En reposo, el puerto paralelo debe tener las líneas *nAddrStrobe*, *nDataStrobe*, *nWrite* y *nReset* inactivas, es decir, en estado alto. En algunos puertos puede ser necesario poner las líneas en este estado manualmente antes de iniciar ningún ciclo EPP. Por tanto lo primero que haremos será escribir *XXXX0100* en el registro de control SPP.

En otras tarjetas no es posible realizar un ciclo de escritura si ésta está configurada en modo inverso. Por tanto, también es recomendable asegurarse de que el bit 5 del puerto de control está a 0 antes de continuar.

Otro problema que podemos encontrarnos está relacionado con el bit de *time-out*. Algunos puertos pueden dejar de funcionar correctamente mientras este bit se encuentre a 1. Un caso típico es que todas las lecturas de datos o de instrucciones nos devuelvan siempre 0xFF. Por tanto, es recomendable que tras cada escritura o lectura se compruebe si se produjo *time-out*, y en caso afirmativo limpiar este bit. El proceso para limpiar este bit no es estándar, sino que depende del fabricante del *chipset*. En algunos modelos este bit se pone a 0 tras una doble lectura, en otros es necesario escribir un 0 y en otros un 1.

*3.5. EL PUERTO PARALELO MEJORADO - EPP*

---

# **Capítulo 4**

## **Diseño del hardware**

### **4.1. Diseño lógico del circuito**

La mayor parte de los ordenadores PC y compatibles fabricados a partir del año 95 (equipos Pentium y superiores) incorporan un puerto paralelo con soporte EPP y ECP. ¿Por qué no aprovechar las ventajas que nos ofrecen estos puertos paralelos mejorados? Decidimos pues, una vez descartado el protocolo ECP debido a que requiere un sistema de control externo bastante complejo, usar si es posible el protocolo EPP.

EPP es un protocolo sencillo, que requiere una lógica externa de control bastante simple. Además distingue transferencias de direcciones y datos, por lo que se adapta perfectamente a nuestras necesidades.

De las 4 operaciones que nos ofrece EPP, nuestro circuito hará uso de tres:

- Escritura de direcciones
- Lectura de datos
- Escritura de datos

#### **4.1.1. Envío de direcciones**

Las 8 líneas de datos del puerto paralelo nos permiten realizar transferencias de 8 bits, pero las memorias que vamos a usar tienen hasta 19 líneas de dirección, por tanto usaremos 3 registros de 8 bits de carga en paralelo para almacenar una dirección. De esta forma, para enviar una dirección completa se deberán realizar tres escrituras de dirección sucesivas, en cada una de las cuales se cargará uno de los registros. A priori tenemos dos formas de organizar los registros:

#### 4.1. DISEÑO LÓGICO DEL CIRCUITO

- Una consiste en conectar directamente las líneas de datos del puerto paralelo a las entradas de datos de los registros, y mediante alguna señal indicar en cuál de los registros queremos realizar la escritura. Esta solución tiene la ventaja de que sólo habría que modificar la parte que nos interese de la dirección completa; esto nos permitiría, por ejemplo, incrementar la dirección de memoria para realizar accesos secuenciales a la memoria realizando, de media, una única operación de escritura. Como inconvenientes tiene una mayor complicación del circuito y la necesidad de al menos dos líneas extra de salida del puerto paralelo. EPP sólo nos deja, a lo sumo, una línea de salida para otros usos (la señal de *Reset* sería la única que podría usarse para este menester), por lo que descartamos esta opción.
- La otra consiste en una configuración en forma de registros de desplazamiento, es decir, colocando los tres registros en cascada con una señal de reloj común, de forma que en cada operación de escritura los registros tomen del uno al otro los valores y el primero los tome del puerto paralelo. Ésta es la opción por la que hemos optado.

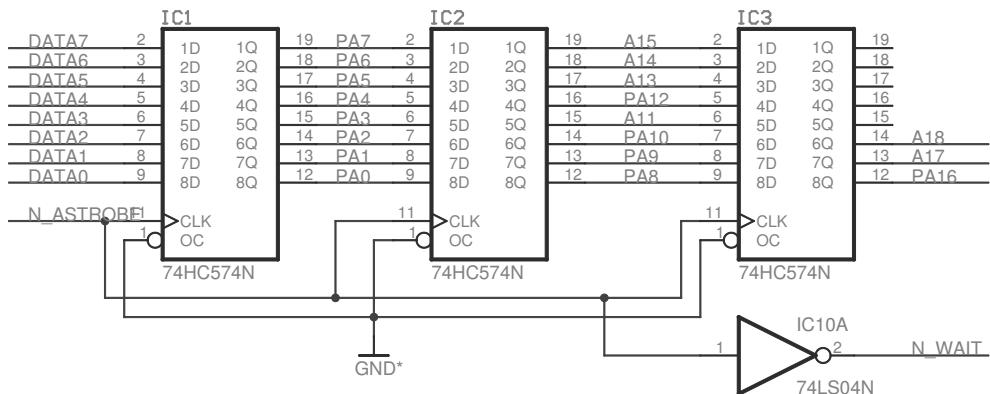


Figura 4.1: Lógica de direccionamiento

Usaremos pues tres registros 74574. Estos registros cargan los datos en los flancos ascendentes del reloj. Observando el cronograma de la figura 3.3 vemos que en el flanco ascendente de *nAddrStrobe* la dirección está aún disponible en las líneas de datos, así que podemos usar directamente esta señal para controlar la captura de las direcciones por los registros.

Nos queda generar de alguna forma la señal *nWait* que el puerto paralelo espera como indicación, en primer lugar, de que el dispositivo está listo para la captura de la dirección (paso de *nWait* a nivel alto), y en segundo lugar, de que ésta ya se ha realizado (vuelta a nivel bajo). En principio parece que la misma señal *nAddrStrobe* pasada por un inversor nos puede servir, ya que el retardo añadido por el inversor más el tiempo que el PC tarde en retirar *nAddrStrobe*

tras recibir  $nWait$  superará el tiempo de *setup* de los registros, es decir, el tiempo mínimo que deben llevar los datos a la entrada del registro antes de que se puedan capturar.

### 4.1.2. Lectura de datos

Para la lectura y grabación de datos se ha considerado que lo más sencillo es conectar directamente las líneas de datos del puerto paralelo y de la memoria, tomando precauciones para que nunca ambos vuelquen datos a la vez provocando un cortocircuito.

Suponemos que en pasos anteriores ya hemos cargado la dirección en los registros, de forma que la memoria volcará el dato al bus cuando pongamos *OE* a nivel bajo. Observando el cronograma de un ciclo de lectura de la figura 3.4 vemos que podemos obtener fácilmente *OE* a partir de *nDataStrobe* y *nWrite*. *OE* debe ponerse a nivel bajo cuando lo haga *nDataStrobe*, pero sólo si *nWrite* sigue a nivel alto, es decir, si se trata de una lectura. En la figura 4.2 se muestra la tabla de verdad y la reducción de Karnaugh de esta función.

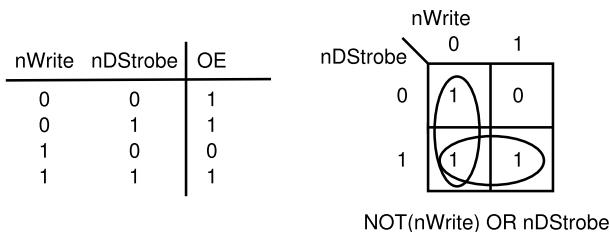


Figura 4.2: Tabla de verdad y mapa de Karnaugh de *OE*

La señal *nWait* la obtendremos de forma similar a como hicimos antes, invirtiendo esta vez *nDataStrobe*. Sin embargo, ahora debemos esperar lo suficiente para que la memoria acceda al dato y lo vuelque al bus antes de indicarle al sistema que el dato está disponible, es decir, antes de poner a 1 esta señal, por lo que introduciremos un elemento de retardo cuyo valor calcularemos luego. Una vez que el PC detecte la subida de *nWait*, capturará el dato, retirará *nDataStrobe* y esperará que *nWait* vuelve a bajar para dar por concluida la transferencia.

El circuito de control resultante se muestra en la figura 4.3.

### 4.1.3. Escritura de datos

Las memorias flash capturan la dirección en el flanco descendente de *WE* y el dato en el flanco ascendente. Como la dirección está disponible desde ciclos anteriores, para generar *WE* vamos a usar *nWrite* y *nDataStrobe*, de forma que cuando ambas estén a nivel bajo *WE* lo haga también (figura 4.4). La subida de *nDataStrobe*, y por tanto de *WE*, no se producirá hasta que no

#### 4.1. DISEÑO LÓGICO DEL CIRCUITO

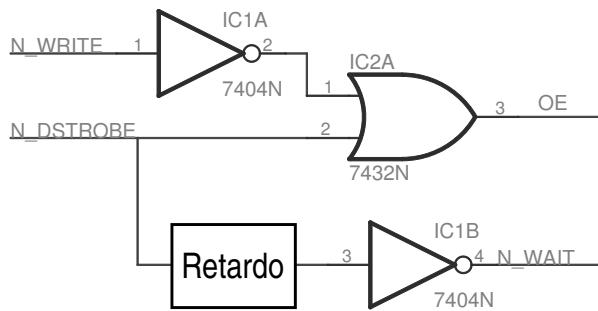


Figura 4.3: Lógica de control resultante para la lectura de un dato

pongamos  $nWait$  a nivel alto, lo que nos permite, ajustando el retardo de  $nWait$ , asegurarnos de que se cumplen los requisitos de tiempos de *setup* de la memoria antes de realizar la escritura.

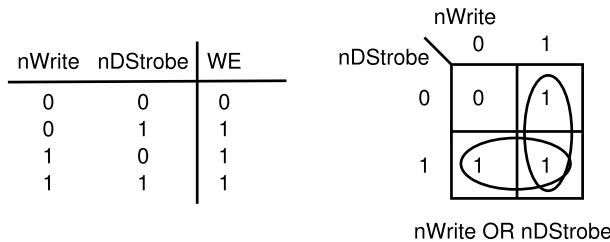


Figura 4.4: Tabla de verdad y mapa de Karnaugh de *WE*

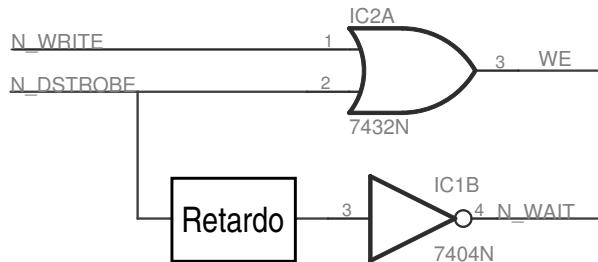


Figura 4.5: Lógica de control resultante para la escritura de un dato

Durante las operaciones de escritura nos encontramos que las dos entradas de las funciones lógicas que hemos desarrollado para *OE* y *WE* cambian simultáneamente. Esto, como se puede observar en la figura 4.6, puede producir efectos indeseados dependiendo de cuál de ellas cambie primero; en este caso el efecto es aún más visible debido al retardo adicional producido por la puerta NOT. Vemos que al comienzo de las escrituras de datos se produce un pequeño pulso descendente en *OE*. El resultado es incierto ya que no sabemos qué hará la memoria si le llegan al mismo tiempo solicitudes de lectura y de escritura. Si la memoria considera la situación como una lectura, no sólo probablemente falle la escritura, sino que nos encontramos ante

una situación peligrosa: tanto la memoria como el puerto paralelo están volcando datos al bus, pudiéndose producir un cortocircuito y dañarse la memoria, el puerto o ambos.

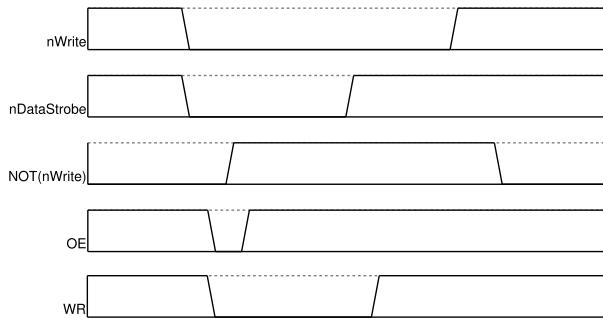


Figura 4.6: *Glitch* en *OE* durante las escrituras de datos

La solución pasa por retrasar *nDataStrobe* lo suficiente para que cuando ésta cambie, *nWrite* ya lo haya hecho si es que se trata de una escritura. El retardo de un par de puertas NOT será suficiente.

#### 4.1.4. Selección del tipo y tamaño de memoria

Para la selección del tipo y tamaño de la memoria conectada al programador se usará un bloque de pines similar al de los programadores de Frohwein y Sshua.

En las secciones 2.2.1 y 2.3.1 se hizo un estudio previo del patillaje de los modelos más usuales que podemos encontrar de memorias EPROM y Flash en formato PDIP. La disposición de los pines de selección de tipo y tamaño de memoria de los circuitos de Frohwein y Sshua nos permite establecer combinaciones para todas las memorias estudiadas, salvo aquellas memorias Flash de 2Mbit con pin de reset, que necesitarán tener conectado el pin 1 a *Vcc* para dejar inactiva esta entrada y las memorias EPROM de 4Mbit, cuya patilla 31 ya no es la entrada de pulsos de grabación (*PGM*), sino la línea de dirección *A18*. Parece ser que Frohwein ya tenía en mente la posibilidad de estas ampliaciones cuando realizó su diseño original, pues dejó tres pines del bloque sin usar que se prestan muy fácilmente para la asignación de estas señales.

Vemos que el pin 2 (inferior izquierdo) del bloque se encuentra justo debajo del pin al que llega la línea *A18* desde los registros de desplazamiento (pin 4). Si conectamos este pin número 2 al pin 31 del zócalo de la memoria, podremos conectar éste a la línea *A18* colocando un jumper entre los pines 2 y 4 del bloque.

Por otro lado tenemos que el pin 3 del bloque está conectado al pin 1 del zócalo de la memoria, que se corresponde con la señal de reset de las memorias de 2Mbit comentadas. Conectando el pin 1 a *Vcc* a través de una resistencia de pull-up, permitiremos dejar a nivel alto

el pin 1 de la memoria para inhabilitar el reset colocando un jumper entre los pines 1 y 3 del bloque. Otra opción posible habría sido usar el pin 5 en vez del 1.

Si observamos las 9 organizaciones de pines de memorias EPROM mostradas en las figuras 2.1, 2.2 y 2.3 comprobamos que algunas podrían agruparse, ya que sólo se diferencian unas de otras en que tienen alguna línea de dirección sin conectar respecto a otra. Tal es el caso de los integrados de 64Kbit y 128Kbit ( $A_{13}$  sin conectar en el primero) o los de 1Mbit y 2Mbit ( $A_{17}$  sin conectar en el 27C010). Nos quedan por tanto 7 combinaciones del grupo de pines para la configuración de las memorias EPROM (figura 4.7)

24 pines		28 pines			32 pines		
16Kbit	32Kbit	64Kbit/128Kbit	256Kbit	512Kbit	1Mbit/2Mbit	4Mbit	
○ ○	○ ○	█ █	○ ○	○ ○	○ ○	○ ○	
○ ○	○ ○	○ ○	○ ○	○ ○	○ ○	○ ○	
█ ○	█ ○	█ █	█ ○	█ ○	█ ○	█ ○	
█ ○	█ ○	█ ○	█ ○	█ ○	█ ○	█ ○	
█ ○	█ ○	█ ○	█ ○	█ ○	█ ○	█ ○	
○ ○	○ ○	○ ○	○ ○	○ ○	○ ○	○ ○	
○ ○	○ ○	○ ○	○ ○	○ ○	○ ○	○ ○	
○ ○	○ ○	○ ○	○ ○	○ ○	○ ○	○ ○	
○ ○	○ ○	○ ○	○ ○	○ ○	○ ○	○ ○	

Figura 4.7: Configuración de jumpers para memorias EPROM

De igual forma, de las 9 variantes de asignación de señales a pines en memorias Flash de 5V de las figuras 2.5 y 2.6 obtenemos 6 combinaciones de configuración distintas agrupando las memorias de 16Kbit/28 pines y 64Kbit/28 pines por un lado, y las de 512Kbit, 1Mbit y 2Mbit (con y sin *reset*) por otro. En este último grupo podemos añadir también las Flash de 12V, puesto que conectamos el pin 1 a  $V_{cc}$  mediante una resistencia de pull-up que según la documentación es una entrada válida para  $V_{pp}$  para la lectura.

Estas 6 combinaciones se muestran en la figura 4.8.

La figura 4.9 muestra el circuito resultante de combinar todo lo visto hasta ahora, así como una serie de triestados para dejar en alta impedancia las señales de control de la memoria cuando el circuito esté apagado, la fuente de alimentación (descrita a continuación), condensadores de desacoplo para los integrados y el conector *Centronics*.

#### 4.1.5. Fuente de alimentación

Como queremos un programador de bajo coste, lo ideal es permitir la alimentación del circuito con un pequeño transformador sin imponer muchas restricciones en cuanto a voltajes y polaridad, de forma que casi cualquier fuente de alimentación que el usuario tenga en casa

## 4.1. DISEÑO LÓGICO DEL CIRCUITO

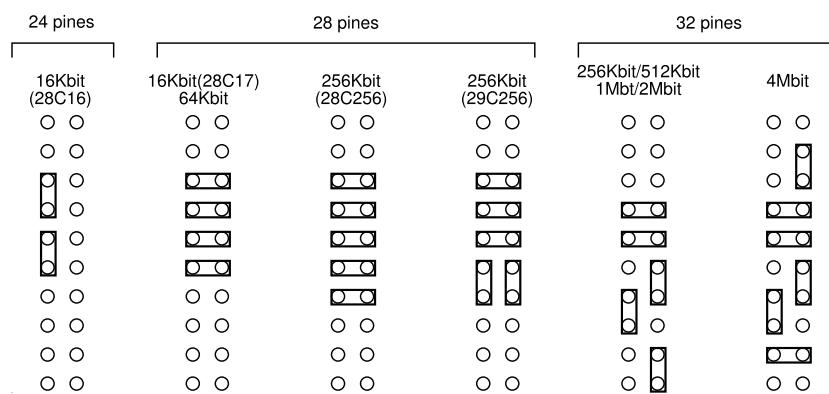


Figura 4.8: Configuración de *jumpers* para memorias Flash

pueda servir.

El circuito de rectificación y estabilización del programador original de Jeff Frohwein se adapta perfectamente a nuestras necesidades. Un puente de diodos se encarga de darnos una corriente con una polaridad determinada independientemente de la polaridad de la corriente de entrada, e incluso permitiría usar una fuente de corriente alterna para alimentar nuestro circuito. Seguidamente, con un regulador de corriente 7805 obtenemos los 5V necesarios para alimentar los integrados de la placa. Un par de condensadores en paralelo con la entrada y la salida del regulador ayuda a estabilizar la alimentación eliminando picos y rizados.

El resto del circuito consta de un simple interruptor y una resistencia y un diodo led en serie para indicar cuándo están activadas las entradas y alimentado el zócalo de la memoria.

### 4.1.6. Cálculo del retardo

Para la implementación del circuito vamos a usar integrados de la familia HCT, que son dispositivos CMOS de alta velocidad de conmutación y compatibles con niveles TTL. Esto nos permitirá disfrutar de la velocidad y bajo consumo de los circuitos CMOS y trabajar cómodamente con el puerto paralelo (TTL) y las memorias flash (compatibles CMOS y TTL).

Función	Integrado	Retardo típico
Puertas NAND	74HCT00	10ns
Puertas NOR	74HCT02	9ns
Inversores	74HCT04	8ns
Inversores Schmitt trigger	74HCT14	17ns
Puertas OR	74HCT32	9ns
Triestados	74HCT125	12ns

Tabla 4.1: Tiempos de retardo de algunos elementos de la familia HCT

#### 4.1. DISEÑO LÓGICO DEL CIRCUITO

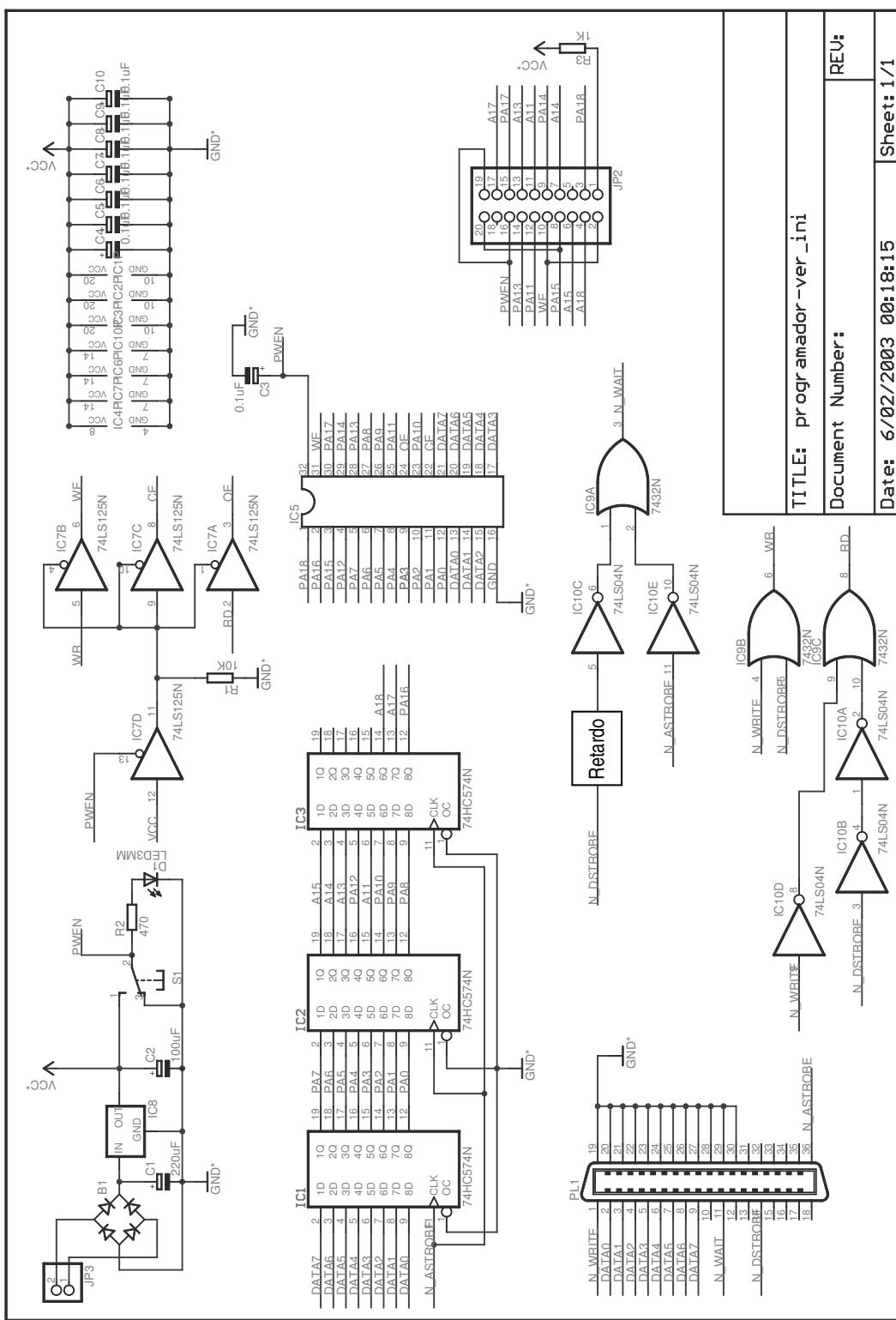


Figura 4.9: Versión inicial del programador

En la tabla 4.1 se muestran los tiempos de retardo de algunos elementos de esta familia. Con la configuración actual, la señal *OE* tiene un retardo total respecto a *nDataStrobe* de  $2 * T_{NOT} + T_{OR} + T_{TRI} = 2 * 8 + 9 + 12 = 37\text{ns}$ . Esta señal nos interesa que tenga el menor retardo posible para que la memoria flash comience a volcar el dato al bus cuanto antes. El circuito admite una pequeña optimización: la sustitución de las puertas OR por puertas NAND tal y como se ve en la figura 4.10. Esto reduce el retardo de *OE* a  $T_{NOT} + T_{NAND} + T_{TRI} = 8 + 10 + 12 = 30\text{ns}$ . Pero ésta no es la única ventaja que nos aporta este cambio sino que además se reduce el número de inversores necesarios. Aunque se produce un incremento del retardo en la señal *WE* esto no nos perjudica, e incluso puede ser beneficioso al aumentar el margen del tiempo de *data setup* de la memoria.

Vamos a calcular los requisitos del retardo para una operación de lectura. Teníamos por un lado que el retraso total de *OE* respecto a *nDataStrobe* es de 30ns. Tras consultar las especificaciones de un buen número de memorias hemos visto que las más lentas, por ejemplo la SyncMOS F29C51004 de 120ns de tiempo de acceso, tardan hasta 60ns desde la bajada de *OE* en volcar el dato al bus de forma estable. Por tanto, el retardo total de subida *nWait* debe ser como mínimo de 90ns (nuestros 30ns de retardo más el tiempo de acceso a la memoria). Este retardo total será la suma del retardo de la puerta NAND (10ns) y el retardo de bajada introducido por nuestro elemento retardante. Por tanto tenemos que este retardo debe ser al menos de 80ns. Por otro lado, según vimos, el protocolo EPP exige que desde la subida de *nDataStrobe* hasta la bajada de *nWait* no pasen más de 125ns; si a esto le restamos los 10ns de la puerta NAND, tenemos que el retardo de subida puede ser como máximo de 115ns. Resumiendo:

**80ns < Retardo bajada**

**Retardo subida < 115ns**

En las operaciones de escritura deben cumplirse cuatro condiciones. La primera es que debe respetarse el tiempo de *data setup*, es decir, el tiempo que el dato debe estar en las líneas de entrada antes de su captura (en el flanco de subida de *WE*); en las memorias más lentas llega a ser de 55ns. Tenemos pues que:

$$\begin{aligned}
 &\text{Retardo subida } nWait + \text{Tiempo respuesta PC (0 a 1s)} + \text{Retardo WE (30ns)} > \text{Data Setup Time (55ns)} \\
 &\text{Retardo bajada} + \text{Retardo NAND (10ns)} + \text{Tiempo respuesta PC} + \text{Retardo WE} > 55\text{ns} \\
 &\text{Retardo bajada} > 55\text{ns} - 10\text{ns} - \text{Tiempo respuesta PC} - 30\text{ns} \\
 &\text{Retardo bajada} > 15\text{ns} - \text{Tiempo respuesta PC}
 \end{aligned}$$

y considerando los valores extremos del tiempo de respuesta del PC:

#### 4.1. DISEÑO LÓGICO DEL CIRCUITO

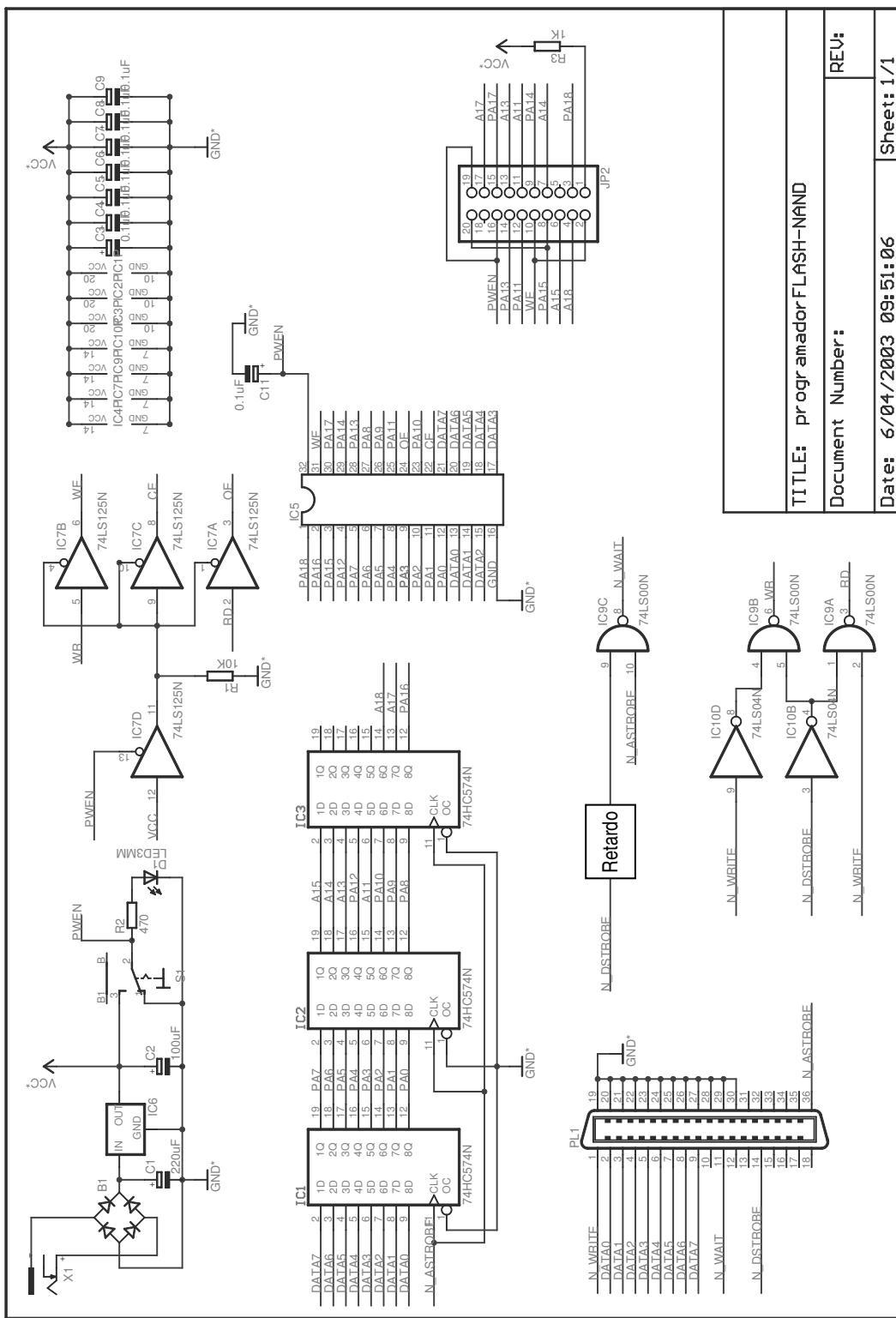


Figura 4.10: Programador con puertas NAND

#### 4.1. DISEÑO LÓGICO DEL CIRCUITO

(Retardo bajada > 15ns - 0ns) y (Retardo bajada > 15ns - 1s)  
Retardo bajada > 15ns

La segunda condición es que el ancho del pulso en *WE* debe ser de un tamaño mínimo determinado. Algunas de las memorias estudiadas requieren un pulso de hasta 70ns. El ancho del pulso viene determinado por el retardo de *nWait* más el tiempo de respuesta del PC:

Retardo subida *nWait* + Tiempo respuesta PC (de 0 a 1s) > Ancho mínimo del pulso (70ns)  
Retardo bajada + Retardo NAND (10ns) + Tiempo respuesta PC > 70ns  
Retardo bajada > 60ns - Tiempo respuesta PC  
(Retardo bajada > 60ns - 0ns) y (Retardo bajada > 60ns - 1s)  
Retardo bajada > 60ns

Cuando el puerto paralelo detecta la retirada de *nWait* considera que el ciclo ha concluido y procede a retirar el dato del bus. Por tanto, el retraso de *nWait* respecto a *nDataStrobe* debe ser mayor que el de *WE* respecto a *nDataStrobe* para evitar que la memoria retire el dato antes de que lo capture la memoria:

Retardo *bajada nWait* > Retardo *WE* (30ns)  
Retardo subida + Retardo NAND (10ns) > Retardo *WE*  
Retardo subida > 20ns

Y, por último, al igual que en las lecturas, el retardo total de *nWait* no puede superar los 125ns, y por tanto nuestro retardo deber ser de 115ns como máximo:

Retardo subida <115ns

Juntando todas las condiciones nos queda que:

**60ns <Retardo bajada**  
**20ns <Retardo subida <115ns**

Considerando ambas operaciones de lectura y escritura tenemos que nuestro retardo debe cumplir:

**80ns <Retardo bajada**  
**20ns <Retardo subida <115ns**

o si usamos un elemento que produzca un retardo simétrico, éste debe estar comprendido entre los siguientes límites y lo más alejado posible de los extremos:

$$80\text{ns} < \text{Retardo} < 115\text{ns}$$

#### 4.1.7. Obtención del retardo

Vamos a evaluar las alternativas que tenemos para obtener un retardo adecuado. Lo más sencillo sería disponer de un elemento discreto que nos ofreciera un retardo dentro de nuestros márgenes. Tras mucho buscar descubrimos que pocos son los fabricantes de este tipo de dispositivos; tan sólo de *Dallas Semiconductor* (empresa absorbida por *Maxim*) se pudo encontrar alguno que sirviera para nuestros propósitos: el DS1005. Consiste en un integrado que reproduce la señal de entrada en cada una de sus 5 salidas con retardos igualmente espaciados; en concreto el DS1005-150, que ofrece retardos de 30, 60, 90, 120 y 150ns, haciendo uso de la salida de 90ns o bien el DS1005-125, con su salida de 100ns de retardo, nos hubieran servido. Pero al parecer es muy difícil conseguir este dispositivo en España, especialmente en pequeñas cantidades, y tras preguntar a varios distribuidores de componentes electrónicos nos dimos por vencidos.

Otra posible opción consiste en disponer una serie de puertas lógicas en cadena, de forma que sumen sus retardos. Podríamos usar por ejemplo un número par de inversores y aprovechar así los que nos sobran del 74HCT04 que ya usamos. Pero resulta que necesitaríamos un total de 12 inversores para generar un retardo de 96ns (punto medio entre 80 y 115ns). Esto supondría añadir dos nuevos 74HCT04, por lo que nos planteamos intentar encontrar otra solución que requiera añadir como mucho un integrado.

Una estrategia que en principio nos pareció interesante fue usar en cadena los 4 inversores que nos sobran del 74HCT04 y colocar en medio de la cadena un pequeño condensador para retardar la señal. Así que, osciloscopio en mano, nos pusimos a realizar pruebas. Se hicieron mediciones con los integrados SN74HCT04 de Philips y el MC74HCT04 de Motorola, colocando dos inversores en cadena y un condensador en medio. La tabla 4.2 muestra los resultados.

	Condensador 833pF	Condensador 1nF
Retraso subida SN74HCT04	106 ns	122 ns
Retraso subida MC74HCT04	76 ns	84 ns
Retraso bajada SN74HCT04	66 ns	77 ns
Retraso bajada MC74HCT04	40 ns	55 ns

Tabla 4.2: Variaciones del retardo de un par de inversores producidas por un condensador

Lo primero que observamos es que el condensador no produce un retardo simétrico, sino que afecta sobre todo a las bajadas de las señales; de todas formas nuestros requisitos tampoco son simétricos así que esto no supone ningún problema, tan sólo determinará el punto de colocación del condensador en la cadena de inversores. Lo que sí es un gran problema es que los integrados de distintos fabricantes muestran un comportamiento muy dispar el uno del otro ante la presencia del condensador: el SN se retrasa muchísimo más de lo que lo hace el MC. No nos podemos permitir variaciones tan grandes, por lo que descartamos esta idea.

Mientras realizamos estas mediciones observamos que la simple diferencia de capacitancia de la sonda del osciloscopio entre las posiciones x1 y x10 afectaba significativamente a los retardos; esto nos hizo pensar que nuestro circuito debería poderse ajustar a las diferencias de capacitancia de los distintos puertos paralelos en los que se fuera a usar el programador y las variaciones de los retardos entre integrados de distintos fabricantes.

Finalmente acabamos volviendo a la sencilla solución de colocar varios inversores en cadena, usando inversores de tipo *Schmitt trigger* que tienen un mayor retardo debido a la lógica adicional que incluyen para el control de disparo con histéresis. La siguiente tabla muestra algunas de las combinaciones de estos dos tipos de inversores y su retardo total.

	Retardo total
4 inversores 74HCT14	68 ns
4 inversores 74HCT14 + 2 74HCT04	84 ns
6 inversores 74HCT14	102 ns
6 inversores 74HCT14 + 2 74HCT04	118 ns

Tabla 4.3: Combinaciones de inversores en cadena y sus retardos totales

Con estas cuatro combinaciones pensamos que tenemos cubierto un buen margen de posibilidades y, con un par de *jumpers*, haremos que el usuario pueda seleccionar entre ellas (figura 4.11).

#### 4.1.8. Coste

Como referencia del coste que puede tener nuestro programador, la tabla 4.4 muestra el precio al que podría salir el conjunto de componentes necesarios para construirlo, sin contar el coste de fabricación de la placa PCB.

Estos precios son los que ofrece la tienda “online” Farnell<sup>1</sup> a fecha de septiembre de 2003. La mayoría de los precios son para unidades sueltas, aunque algunos son para compras mínimas de 5, 10 o 50 unidades, por lo que el precio al comprar pocas unidades podría subir. Sin embargo,

<sup>1</sup><http://www.farnell.com>

#### 4.1. DISEÑO LÓGICO DEL CIRCUITO

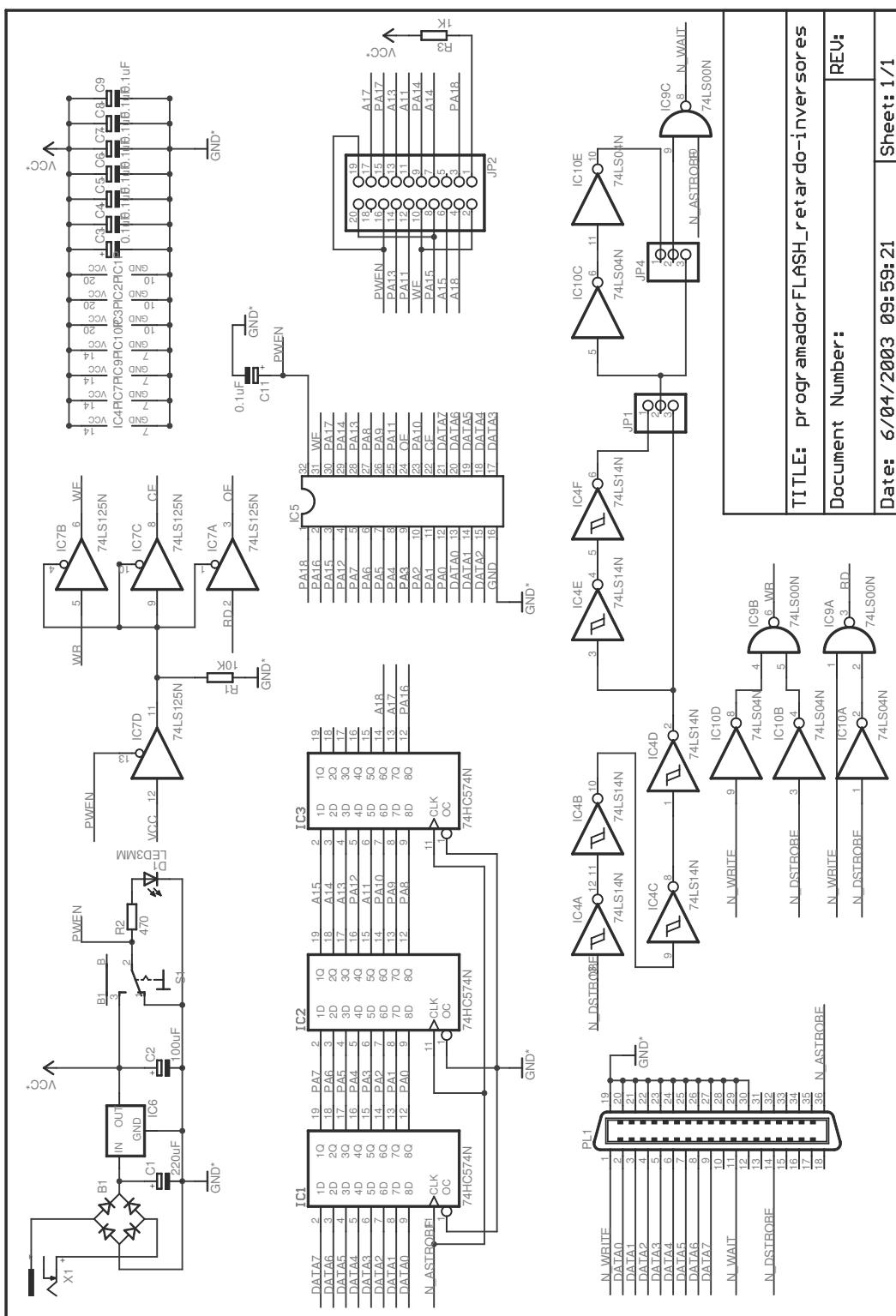


Figura 4.11: Programador con retardo basado en inversores

## 4.2. DISEÑO PCB DEL CIRCUITO

---

parece ser que esta tienda no es precisamente barata, de forma que es posible que en cualquier tienda de electrónica local se pueda conseguir un precio más ajustado.

Cantidad	Componente	Precio unidad	Comprando...	Precio total
3	74HCT574	0.59	1	1.77
1	74HCT14	0.36	1	0.36
1	74HCT125	0.34	1	0.34
1	74HCT00	0.30	1	0.30
1	74HCT04	0.30	1	0.30
1	KA7805	0.49	1	0.49
1	Puente diodos 1.5A 50V	1.82	1	1.82
1	Interruptor	2.81	1	2.81
1	LED rojo 5MM	0.19	5	0.19
8	Cond. 0.1 $\mu$ F 35V	0.41	10	3.28
1	Cond. 220 $\mu$ F 25V	0.26	10	0.26
1	Cond. 100 $\mu$ F 16V	0.11	10	0.11
2	Cond. 0.1nF 100V	0.20	10	0.40
1	Resist. 270 $\Omega$ 0.5W	0.04	50	0.04
1	Resist. 1K $\Omega$ 0.5W	0.04	50	0.04
1	Resist. 10K $\Omega$ 0.5W	0.04	50	0.04
1	Conect. centronics PCB 36H	7.99	1	7.99
<b>TOTAL Euros</b>				20.54

Tabla 4.4: Coste de componentes

## 4.2. Diseño PCB del circuito

Antes de pasar al diseño de la placa veremos brevemente algunas consideraciones que vamos tener en cuenta para reducir el posible ruido en las señales, algunas de las cuales ya hemos aplicado:

- Separación en la medida de lo posible de la parte analógica y la digital, evitando así que el ruido en la línea de alimentación de entrada, sin estabilizar ni filtrar y quizás incluso sin rectificar, afecte al resto del circuito.

- Cada circuito integrado llevará lo más cerca posible un pequeño condensador de desacoplamiento ( $0.1\mu F$ ) conectado a sus entradas de alimentación. Esto evita el rizado en la alimentación cuando los integrados comutan. Durante algunas pruebas preliminares comprobamos que la adición de dos condensadores de  $0.1nF$  a la fuente de alimentación producían una mayor reducción del ruido en las líneas de alimentación, y por ende, en las de datos y control.
- Se usarán planos de masa en vez de pistas convencionales para proporcionar apantallamiento electromagnético y reducción de ruido.
- En nuestra placa prototipo se dejó espacio para la colocación de pequeños condensadores en las líneas de datos, por si eran necesarios para filtrar ruido. En las pruebas preliminares demostraron ser innecesarios, por tanto en el diseño final no se contemplan.

A la hora de diseñar la placa se tendrán también en cuenta las siguientes directrices:

#### 4.2.1. Colocación de componentes

- Lo primero es decidir la colocación de los conectores y los circuitos integrados, luego se distribuirán los demás componentes a su alrededor.
- Los conectores deben colocarse en los bordes de la placa para facilitar su conexión y desconexión.
- Los integrados se colocarán de forma ordenada y equidistantemente repartidos por la placa, preferiblemente con orientación horizontal o vertical, nunca en diagonal.
- Se debe procurar colocar cerca aquellos componentes que tienen más conexiones entre sí.
- Los integrados que tengan conexiones directas de entrada o salida a los conectores, se intentarán colocar cerca de éstos.
- Se debe dejar un espacio entre los componentes y los bordes de la placa.

#### 4.2.2. Trazado de líneas

- Deben evitarse los ángulos agudos y los cambios bruscos de dirección en las pistas.
- La conexión entre conductores debe realizarse en ángulo recto.

- Las pistas no deben acercarse demasiado a los bordes de la placa.
- Las líneas de alimentación deben tener al menos 1mm de ancho para evitar su calentamiento.
- Se deben evitar los bucles porque producen inductancias parásitas.
- Se recomienda que el trazado de la alimentación se realice en forma de bus, de forma que recorra linealmente toda la placa alimentando uno a uno cada circuito integrado.
- Las líneas que lleven señales de alta frecuencia tales como los relojes es conveniente que estén también trazadas en forma de bus, haciendo una distribución lineal de las señales.
- Intentaremos reducir el número de vías para simplificar el montaje de la placa.

Para el trazado de las pistas usaremos el programa Eagle, que es con el que hemos realizado también los diagramas esquemáticos. Eagle ofrece una herramienta de trazado automático de pistas, pero tras varias pruebas llegamos a la conclusión de que la heurística que usa es demasiado simple y no íbamos a conseguir buenos resultados con ella: a pesar de que incrementamos el coste de las vías seguía generando un enorme número de ellas; además suele darle demasiadas vueltas a algunas señales. Ya que tarda poco tiempo en realizar un trazado (apenas 1 minuto en un PIII-450) debería incorporar alguna opción para repetir el proceso varias veces tomando caminos alternativos y quedarse con el más óptimo.

Finalmente se hizo el trazado completamente de forma manual. El diseño final de nuestro programador queda tal y como se muestra en las figuras [4.12](#), [4.13](#) y [4.14](#).

## *4.2. DISEÑO PCB DEL CIRCUITO*

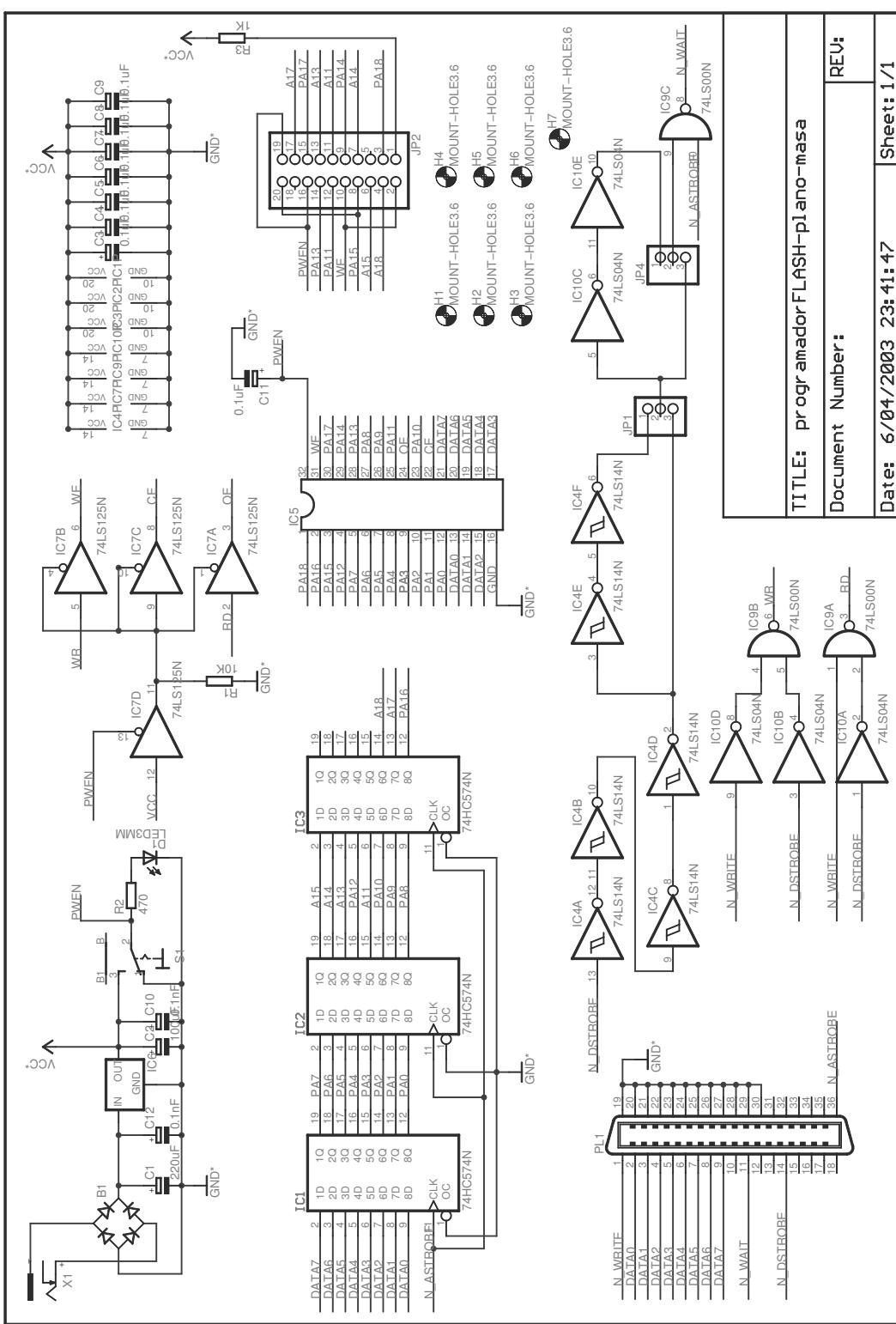


Figura 4.12: Diseño final. Esquemático

## 4.2. DISEÑO PCB DEL CIRCUITO

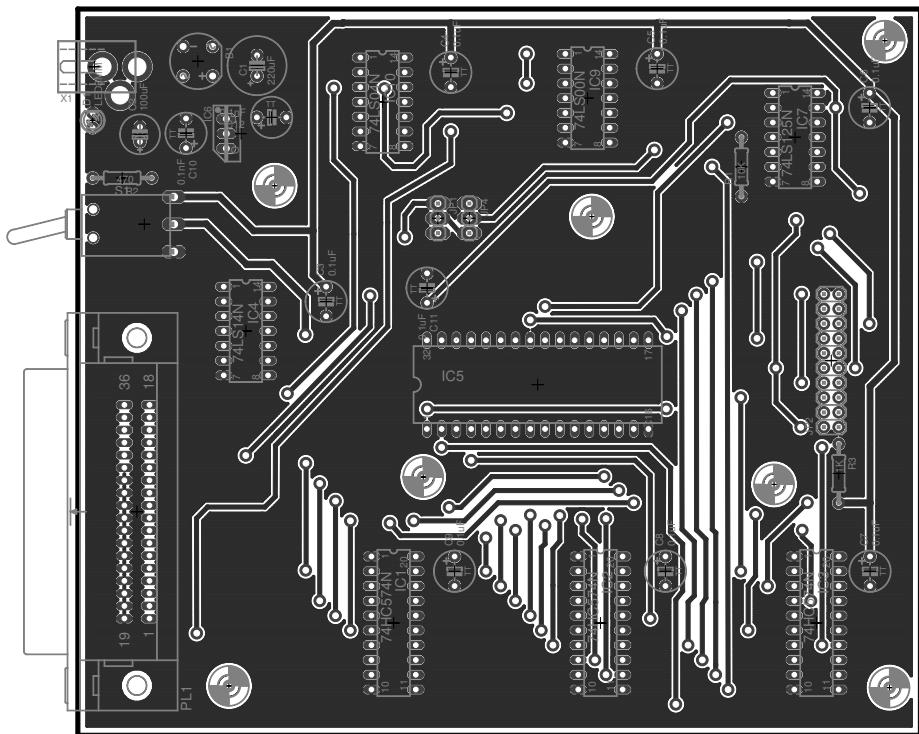


Figura 4.13: Diseño final. Cara superior trazado PCB

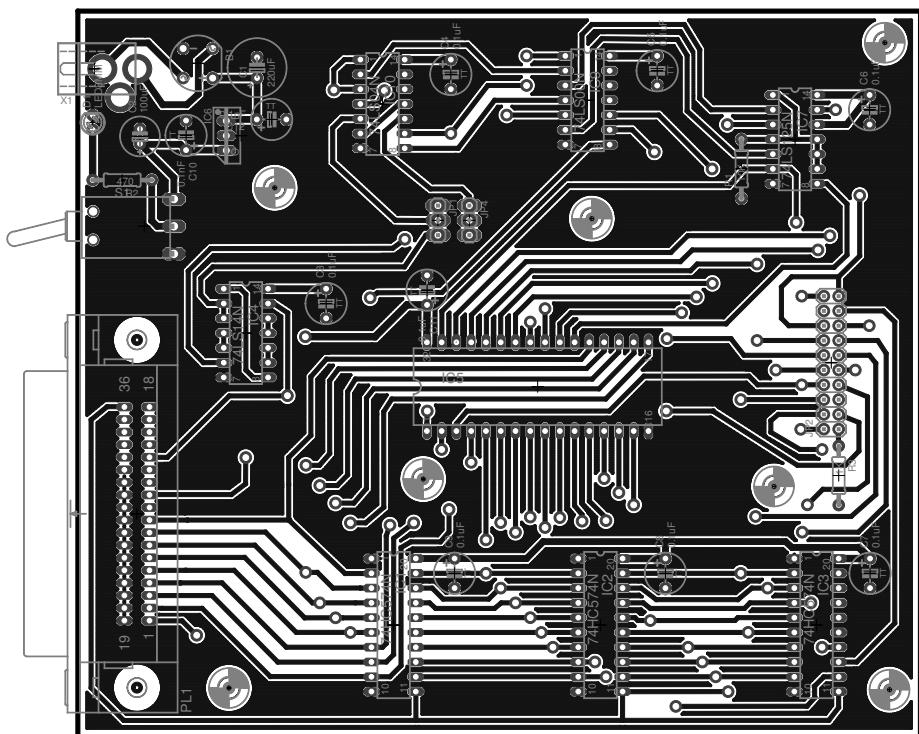


Figura 4.14: Diseño final. Cara inferior trazado PCB

---

*4.2. DISEÑO PCB DEL CIRCUITO*

# **Capítulo 5**

## **Fabricación y montaje de la placa**

Tradicionalmente las placas para los prototipos se han obtenido por procesos químicos. Se usa una placa de baquelita o de fibra de vidrio, con una capa de cobre a cada lado sobre las que lleva una emulsión protectora fotosensible. Con la ayuda de una hoja con todos los taladros impresos se procede a realizar dichos taladros en la placa. Se imprimen las pistas de cada una de las caras en dos láminas transparentes (normalmente de acetato) que se sujetan a cada lado de la placa de forma que coincidan los taladros y se exponen ambas caras a luz actínida durante el tiempo indicado en las instrucciones que acompañan a la placa (que depende de la potencia y tipo de la luz usada). La luz degradará la película protectora, mientras que las zonas protegidas (las pistas y pads) no se verán afectadas. A continuación se introduce la placa en un baño de sosa cáustica para eliminar la película protectora dañada por la luz, quedándose intactas aquellas zonas en las que no dio la luz; a este proceso se le llama revelado. El siguiente paso es introducir la placa en una solución corrosiva (por ejemplo una mezcla de ácido clorhídrico, agua oxigenada y agua) que eliminará el cobre de las zonas no protegidas, dejando por tanto las pistas y pads “impresos” sobre la placa. El último paso consiste en retirar la capa de barniz protector con, por ejemplo, acetona.

Pero el Departamento de Electrónica dispone en el Parque Tecnológico de una máquina para fabricar placas PCB por medios mecánicos, así que decidimos hacer uso de ella. A partir de los ficheros Gerber generados por Eagle la máquina realiza los taladros pertinentes y traza las pistas cortando las láminas de cobre que recubren una placa de fibra de vidrio, aislando así las pistas y pads del resto del cobre.

Tuvimos que fabricar dos placas. La primera salió mal porque desafortunadamente las brocas que quedaban para la máquina eran un poco más gruesas de lo que debieran, de forma que se comieron gran parte de los pads y dejaron unas pistas muy estrechas. Una cosa que aprendimos de este primer intento fue que generar los planos de masa en el trazado lo único

---

que hacía era que la máquina cortara el borde de las pistas y pads dos veces, una para la pista y otra para el plano, lo que es una pérdida de tiempo (el proceso de corte de las pistas no es precisamente rápido). Así que se realizó una segunda versión del trazado PCB con pistas más anchas, pads más grandes y con las líneas de masa sin trazar, ya que usaríamos el cobre restante tras el trazado de la placa como planos de masa, tras comprobar que ninguna conexión a tierra se quedaría aislada.

Una vez comprobada la placa con un multímetro para asegurarnos de que hubiera continuidad en las pistas y que no hubiera cortocircuitos se procedió a la soldadura de los componentes.

Lo primero que se soldó fueron las vías, soldando pequeños trozos de cable rígido entre ambas caras. Conforme se iban soldando vías se iba comprobando todo con la ayuda del polímetro. Las primeras soldaduras se hicieron haciendo uso de resina para soldar, que en otro tipo de placas ayuda a que el estaño sólo se adhiera a las zonas de cobre siendo rechazando por la baquelita, evitando así los cortocircuitos. Sin embargo, la separación entre las zonas de cobre, debido al proceso de corte son porosas y profundas, lo que provoca que en ocasiones se quede algo de resina con pequeñas partículas de estaño, provocando la aparición de resistencias y capacidades parásitas. Afortunadamente, tras raspar y limpiar un poco la separación entre las pistas afectadas se solucionó el problema y, por supuesto, se descartó el uso de la resina.

Una vez soldadas y comprobadas todas las vías se fueron soldando y comprobando poco a poco los componentes de la fuente de alimentación. A continuación se hizo lo mismo con los zócalos para los integrados, bloques de *pines*, el conector *Centronics*, resistencias y condensadores, en este orden. Y como último paso se colocaron los circuitos integrados en sus zócalos correspondientes.

Posteriormente se hicieron algunas pruebas preliminares con la placa conectada al PC pero sin colocar ninguna memoria en el zócalo, con el objetivo de comprobar que todas las señales se generaban correctamente y llegaban a su destino. También se comprobó que la señal *OE* no sufriera el problema descrito en la figura 4.6 y que los registros cargaban correctamente las tres partes en las que se compone una dirección.

# Capítulo 6

## Análisis y diseño del software

Para el análisis y diseño de la aplicación de control del programador y gestión de la base de datos de memoria se usará notación UML siguiendo una metodología clásica orientada a objetos que consta de las siguientes fases:

- Análisis de requisitos. El objetivo de esta fase es especificar el comportamiento deseado del sistema.
- Análisis. Esta fase consiste en la identificación de los componentes fundamentales del dominio y sus relaciones. El resultado es una representación simplificada del sistema.
- Diseño. El diseño consiste en el desarrollo conceptual de una solución al problema, definiendo la arquitectura que tendrá la aplicación y procurando evitar en lo posible entrar en detalles de implementación.

### 6.1. Análisis de requisitos. Casos de uso

**Caso de uso:** Seleccionar memoria.

**Resumen:** El usuario selecciona el modelo de memoria con el que va a operar.

**Actor:** El usuario.

**Precondición:** El programa está en modo de operaciones con memorias.

**Descripción:**

- Trayectoria Básica

## *6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO*

---

1. El sistema muestra una lista de fabricantes.
2. El usuario escoge un fabricante.
3. El sistema muestra una lista de dispositivos del fabricante.
4. El usuario escoge un modelo de la lista.
5. El sistema selecciona la memoria y muestra un resumen de las características del dispositivo.

**Postcondición:** Se ha seleccionado un modelo de memoria con el que se operará a partir de ese momento.

## 6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO

**Caso de uso:** Seleccionar puerto paralelo.

**Resumen:** El usuario indica al sistema el puerto paralelo al que está conectado el programador.

**Actor:** El usuario.

**Precondición:** El programa está en modo de operaciones con memorias.

**Descripción:**

■ Trayectoria básica:

1. El usuario elige la opción de configuración del puerto paralelo.
2. El sistema muestra una lista de los puertos paralelos disponibles en el sistema.
3. El usuario elige uno de los puertos.
4. El usuario indica si se debe comprobar el timeout de EPP o no.
5. El usuario confirma los cambios.
6. El sistema establece el puerto paralelo seleccionado para usarlo a partir de ese instante.

■ Trayectorias alternativas:

- Cancelación de la operación.
  1. El usuario cancela la operación.
  2. El sistema conserva la configuración anterior.

**Postcondición:** Se ha seleccionado un puerto paralelo a través del cuál comunicarse con el programador a partir de ese momento.

## 6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO

**Caso de uso:** Autoidentificar memoria.

**Resumen:** El sistema intentará identificar automáticamente el modelo de memoria conectado al programador.

**Actor:** El usuario.

**Precondición:** El programa está en modo de operaciones con memorias.

**Descripción:**

- Trayectoria básica:
  1. El usuario solicita la autoidentificación de la memoria.
  2. El sistema reconoce la memoria, la selecciona y muestra un resumen de las características de la memoria.
- Trayectorias alternativas:
  - Imposible reconocer memoria.
    1. El sistema no es capaz de reconocer la memoria, muestra un mensaje informando al usuario y conserva la selección de memoria anterior.

**Postcondición:** Se ha seleccionado un modelo de memoria con el que se operará a partir de ese momento.

## 6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO

**Caso de uso:** Comprobar estado protección de sectores.

**Resumen:** Se comprueba el estado en que se encuentra la protección contra escritura de los sectores de la memoria.

**Actor:** El usuario.

**Precondición:** El programa está en modo de operaciones con memorias. Hay un modelo de memoria seleccionado.

**Descripción:**

- Trayectoria básica:
  1. El usuario solicita el estado de protección de sectores.
  2. El sistema interroga a la memoria y muestra el estado de cada sector al usuario.
- Trayectorias alternativas:
  - Operación no soportada.
    1. Si la memoria no está organizada en sectores o no soporta esta función el sistema muestra un mensaje informando al usuario.

**Postcondición:** Se ha informado al usuario del estado del mecanismo de protección de los sectores.

## 6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO

**Caso de uso:** Leer.

**Resumen:** Se realiza una operación de lectura de memoria.

**Actor:** El usuario.

**Precondición:** El programa está en modo de operaciones con memorias. Hay un modelo de memoria seleccionado.

**Descripción:**

- Trayectoria básica:
  1. El usuario pasa a modo de lectura.
  2. El usuario selecciona el tipo de lectura entre: una dirección (byte), sector, rango de direcciones y memoria completa.
  3. El usuario establece los parámetros necesarios en función del tipo de lectura:
    - a) Lectura de un byte: introduce la dirección a leer.
    - b) Lectura de un sector: selecciona el sector a leer.
    - c) Lectura de un rango: introduce las direcciones de comienzo y fin.
    - d) Lectura de la memoria completa: no hay parámetros.
  4. En los casos b, c y d el usuario introduce además el nombre del fichero en el que se volcarán los datos.
  5. El usuario ordena el comienzo de la lectura.
  6. El sistema informa del progreso de la operación.
- Trayectorias alternativas:
  - Faltan parámetros.
    1. Si el usuario se olvida de introducir alguno de los parámetros necesarios el sistema muestra un mensaje de error y aborta la operación.
  - El fichero de destino no se pudo escribir.
    1. Por alguna causa no se pudo escribir en el fichero de destino de la lectura (falta de permisos, fichero de sólo lectura, fichero en uso, etc). Se informa al usuario y se aborta la operación.

**Postcondición:** Se muestra al usuario el dato contenido en la dirección indicada en el caso de una lectura de una sola dirección o el fichero destino contiene los datos almacenados en la memoria en el resto de casos.

## 6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO

**Caso de uso:** Grabar.

**Resumen:** Se realiza una operación de escritura.

**Actor:** El usuario.

**Precondición:** El programa está en modo de operaciones con memorias. Hay un modelo de memoria seleccionado.

**Descripción:**

- Trayectoria básica:

1. El usuario pasa a modo escritura
2. El usuario selecciona el tipo de escritura entre: una dirección (byte), sector, rango de direcciones y memoria completa.
3. El usuario establece los parámetros necesarios en función del tipo de escritura:
  - a) Escritura de un byte: introduce la dirección a escribir.
  - b) Escritura de un sector: selecciona el sector a escribir.
  - c) Escritura de un rango: introduce las direcciones de comienzo y fin.
  - d) Escritura de la memoria completa: no hay parámetros.
4. El usuario elige el origen de los datos a escribir entre: un valor determinado, valores aleatorios y datos de un fichero. En caso de escritura a partir de fichero deberá además introducir el nombre y ruta del fichero.
5. El usuario indica si desea realizar una verificación tras la escritura.
6. El usuario ordena el comienzo de la grabación.
7. El sistema informa del progreso de la operación.
8. Una vez finalizada la escritura se informa al usuario del resultado de la grabación.
9. En caso de así haberlo indicado, se procede a la verificación de la grabación leyendo los datos de la memoria y comparándolos con los datos de origen. Se muestra el resultado de la comparación al usuario.

- Trayectorias alternativas:

- Faltan parámetros.
  1. Si el usuario se olvida de introducir alguno de los parámetros necesarios el sistema muestra un mensaje de error y aborta la operación.

## *6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO*

---

- El fichero de origen de datos no se pudo leer.
  1. Por alguna causa no se pudo leer el fichero (falta de permisos, fichero no existente, etc). Se informa al usuario y se aborta la operación.
- Fallo de grabación.
  1. La memoria no responde, ha indicado un fallo de escritura o falla la verificación. Se informa al usuario y se aborta el resto del proceso.

**Postcondición:** La dirección o direcciones de memoria contienen ahora los datos de origen.

## 6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO

**Caso de uso:** Borrar.

**Resumen:** Se borra el contenido de la memoria o parte de ella.

**Actor:** El usuario.

**Precondición:** El programa está en modo de operaciones con memorias. Hay un modelo de memoria seleccionado.

**Descripción:**

- Trayectoria básica:
  1. El usuario pasa a modo de borrado.
  2. El usuario elige entre memoria completa o sólo un sector.
  3. En el caso del borrado de un sector el usuario indica el sector a borrar.
  4. El usuario ordena el comienzo del borrado.
  5. Se muestra el resultado de la operación al usuario.
- Trayectorias alternativas:
  - Faltan parámetros
    1. Si el usuario ha elegido la opción de borrar sector y no ha seleccionado un sector, el sistema muestra un mensaje de error y aborta la operación.

**Postcondición:** El sector (o la memoria) ha sido borrado.

## 6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO

**Caso de uso:** Comprobar borrado.

**Resumen:** Se comprueba si un sector (o la memoria completa) está borrado.

**Actor:** El usuario.

**Precondición:** El programa está en modo de operaciones con memorias. Hay un modelo de memoria seleccionado.

**Descripción:**

- Trayectoria básica:
  1. El usuario pasa a modo de borrado.
  2. El usuario elige entre memoria completa o sólo un sector.
  3. En el caso de comprobación de un sector el usuario indica el sector.
  4. El usuario ordena el comienzo de la comprobación.
  5. Se muestra el resultado de la operación al usuario.
- Trayectorias alternativas:
  - Faltan parámetros
    1. Si el usuario ha elegido la opción de comprobar sector y no ha seleccionado un sector, el sistema muestra un mensaje de error y aborta la operación.

**Postcondición:** Se ha informado al usuario del estado de borrado de la memoria.

---

## *6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO*

**Caso de uso:** Ejecutar comando.

**Resumen:** Ejecución de comandos simples de las memorias.

**Actor:** El usuario.

**Precondición:** El programa está en modo de operaciones con memorias. Hay un modelo de memoria seleccionado.

**Descripción:**

- Trayectoria básica:

1. El sistema muestra al usuario una lista (posiblemente vacía) con los comandos adicionales (es decir, aparte de los básicos de lectura, escritura y borrado) que soporta la memoria actualmente seleccionada.
2. El usuario elige uno de estos comandos.
3. El usuario ordena la ejecución del comando.

**Postcondición:** Se ha ejecutado el comando indicado por el usuario.

---

## *6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO*

**Caso de uso:** Añadir fabricante.

**Resumen:** Se añade un nuevo fabricante a la base de datos.

**Actor:** El usuario.

**Precondición:** El programa se encuentra en modo de gestión de la base de datos.

**Descripción:**

- Trayectoria básica:
  1. El usuario pasa a modo de gestión de fabricantes.
  2. El usuario indica que desea añadir un nuevo fabricante.
  3. El usuario introduce un nombre y un código de identificación del fabricante.
  4. Se guarda el nuevo fabricante en la base de datos.
- Trayectorias alternativas:
  - Ya existe un fabricante con el mismo código de identificación.
    1. Se muestra un mensaje de error y se cancela la operación.

**Postcondición:** Se ha añadido el nuevo fabricante a la base de datos.

## 6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO

**Caso de uso:** Modificar fabricante.

**Resumen:** Se modifica el nombre o código de un fabricante.

**Actor:** El usuario.

**Precondición:** El programa se encuentra en modo de gestión de la base de datos.

**Descripción:**

- Trayectoria básica:
  1. El usuario pasa a modo de gestión de fabricantes.
  2. El usuario selecciona un fabricante de la lista mostrada por el sistema.
  3. El usuario cambia el nombre o el código del fabricante.
- Trayectorias alternativas:
  - Se ha cambiado el código del fabricante y ya existía un fabricante con el nuevo código.
    1. Se muestra un mensaje de error y se cancela la operación.
  - Se ha intentado cambiar el código de un fabricante que tenía dispositivos asociados.
    1. Se muestra un mensaje de error y se cancela la operación.

**Postcondición:** Se han actualizado el nombre y/o código del fabricante en la base de datos.

## 6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO

**Caso de uso:** Eliminar fabricante.

**Resumen:** Se elimina un fabricante.

**Actor:** El usuario.

**Precondición:** El programa se encuentra en modo de gestión de la base de datos.

**Descripción:**

- Trayectoria básica:
  1. El usuario pasa a modo de gestión de fabricantes.
  2. El usuario selecciona un fabricante de la lista mostrada por el sistema.
  3. El usuario indica que desea borrar el fabricante.
  4. Se elimina el fabricante.
- Trayectorias alternativas:
  - Existen en la base de datos dispositivos del fabricante seleccionado.
    1. Se informa al usuario y se cancela la operación.

**Postcondición:** Se ha eliminado de la base de datos el fabricante solicitado.

## 6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO

**Caso de uso:** Añadir juego de comandos.

**Resumen:** Se añade un nuevo juego de comandos a la base de datos.

**Actor:** El usuario.

**Precondición:** El programa se encuentra en modo de gestión de la base de datos.

**Descripción:**

- Trayectoria básica:
  1. El usuario pasa a modo de gestión de juegos de comandos.
  2. El usuario indica que desea añadir un nuevo juego de comandos.
  3. Se pide al usuario que introduzca el nombre que quiere darle al juego de comandos.
  4. Se crea el nuevo juego de comandos.

**Postcondición:** Se ha añadido a la base de datos un nuevo juego de comandos con una serie de comandos básicos (aún vacíos).

## 6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO

**Caso de uso:** Eliminar juego de comandos.

**Resumen:** Se elimina un juego de comandos.

**Actor:** El usuario.

**Precondición:** El programa se encuentra en modo de gestión de la base de datos.

**Descripción:**

- Trayectoria básica:
  1. El sistema muestra una lista de juegos de comandos y el usuario elige uno.
  2. El usuario indica que desea borrar el juego de comandos.
  3. Se elimina el juego de comandos.
- Trayectorias alternativas:
  - Existe en la base de datos dispositivos con el juego de comandos seleccionado.
    1. Se informa al usuario de que el juego de comandos está en uso y se cancela la operación.

**Postcondición:** Se ha eliminado de la base de datos el juego de comandos y todos los comandos de los que consta.

---

## *6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO*

**Caso de uso:** Copiar juego de comandos.

**Resumen:** Se copia íntegramente un juego de comandos.

**Actor:** El usuario.

**Precondición:** El programa se encuentra en modo de gestión de la base de datos.

**Descripción:**

■ Trayectoria básica:

1. El usuario pasa a modo de gestión de juegos de comandos.
2. El sistema muestra una lista de juegos de comandos y el usuario elige entre el que desea copiar.
3. El usuario indica que desea copiar el juego de comandos.
4. El sistema solicita un nombre para el nuevo juego de comandos.
5. Se copia el juego de comandos.

**Postcondición:** Se ha añadido a la base de datos un nuevo juego de comandos con el mismo contenido que el juego de comandos original.

---

## *6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO*

**Caso de uso:** Añadir comando.

**Resumen:** Se añade un nuevo comando a un juego de comandos.

**Actor:** El usuario.

**Precondición:** El programa se encuentra en modo de gestión de la base de datos.

**Descripción:**

■ Trayectoria básica:

1. El usuario pasa a edición de juegos de comandos.
2. El sistema muestra una lista de juegos de comandos.
3. El usuario selecciona un juego de comandos.
4. El usuario elige la opción de añadir comando.
5. Se pregunta al usuario el nombre del nuevo comando.
6. Se crea un nuevo comando vacío.

**Postcondición:** Se ha añadido un nuevo comando, por ahora vacío, al juego de comandos indicado.

---

## *6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO*

**Caso de uso:** Modificar comando.

**Resumen:** Se modifican las secuencias de un comando.

**Actor:** El usuario.

**Precondición:** El programa se encuentra en modo de gestión de la base de datos.

**Descripción:**

- Trayectoria básica:
  1. El usuario pasa a edición de juegos de comandos.
  2. El sistema muestra una lista de juegos de comandos.
  3. El usuario selecciona un juego de comandos.
  4. El usuario modifica las secuencias del comando.
  5. Se guardan los cambios.

**Postcondición:** Se han guardado en la base de datos las modificaciones realizadas por el usuario.

## *6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO*

---

**Caso de uso:** Eliminar comando.

**Resumen:** Se elimina un comando.

**Actor:** El usuario.

**Precondición:** El programa se encuentra en modo de gestión de la base de datos.

**Descripción:**

- Trayectoria básica:
  1. El usuario pasa a edición de juegos de comandos.
  2. El sistema muestra una lista de juegos de comandos.
  3. El usuario selecciona un juego de comandos.
  4. El usuario elige la opción de eliminar comando.
  5. Se elimina el comando.
- Trayectorias alternativas:
  - El comando es un comando básico.
    1. No se pueden eliminar comandos básicos ya que son necesarios para las operaciones básicas con memorias. Por tanto, se muestra un mensaje de error al usuario y se cancela la operación.

**Postcondición:** Se ha eliminado el comando de la base de datos.

## *6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO*

---

**Caso de uso:** Copiar comando.

**Resumen:** Se copia un comando.

**Actor:** El usuario.

**Precondición:** El programa se encuentra en modo de gestión de la base de datos.

**Descripción:**

- Trayectoria básica:
  1. El usuario pasa a modo de edición de comandos.
  2. El usuario elige un juego de comandos de una lista.
  3. El sistema muestra la lista de comandos de los que se compone el juego.
  4. El usuario elige un comando
  5. El usuario indica que quiere copiar el comando.
  6. El sistema solicita el nombre para el nuevo comando.
  7. Se añade un nuevo comando al juego actual, con las secuencias del comando original.

**Postcondición:** Se ha creado un nuevo comando del juego actual con las mismas secuencias del comando original.

## 6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO

**Caso de uso:** Añadir dispositivo.

**Resumen:** Se añade un nuevo dispositivo a la base de datos.

**Actor:** El usuario.

**Precondición:** El programa se encuentra en modo de gestión de la base de datos.

**Descripción:**

- Trayectoria básica:
  1. El usuario pasa a modo de edición de dispositivos.
  2. El usuario indica que quiere añadir un nuevo dispositivo.
  3. El sistema le ofrece una lista de fabricantes y el usuario selecciona un fabricante de la lista.
  4. El sistema pide al usuario un nombre para el dispositivo.
  5. El sistema pide el código de identificación del dispositivo.
  6. Se crea un nuevo dispositivo con parámetros por defecto.
- Trayectorias alternativas:
  - Ya existe un dispositivo del mismo fabricante con el mismo identificador.
    1. Se muestra un mensaje de error al usuario y se cancela la operación.

**Postcondición:** Se ha añadido un nuevo dispositivo a la base de datos.

## 6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO

**Caso de uso:** Modificar dispositivo.

**Resumen:** Se modifican los parámetros de un dispositivo.

**Actor:** El usuario.

**Precondición:** El programa se encuentra en modo de gestión de la base de datos.

**Descripción:**

- Trayectoria básica:
  1. El usuario pasa a modo de gestión de dispositivos.
  2. El sistema muestra una lista de fabricantes, de entre los cuales el usuario elige uno.
  3. A continuación se muestra una lista de los dispositivos del fabricante seleccionado. El usuario elige uno.
  4. El usuario realiza modificaciones de los parámetros del dispositivo (tamaño, tipo, sectores, etc)
  5. El usuario confirma la modificación de los cambios.
- Trayectorias alternativas:
  - El usuario cancela los cambios.
    1. El usuario puede cancelar en cualquier momento los cambios realizados, recuperando el dispositivo los parámetros anteriores.
  - El usuario abandona el modo de gestión de dispositivos.
    1. Se ofrece al usuario las opciones de guardar los cambios realizados o de cancelarlos, actuando en consecuencia según una de las dos trayectorias anteriores.

**Postcondición:** Se ha actualizado la base de datos con los nuevos parámetros.

## 6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO

**Caso de uso:** Copiar dispositivo.

**Resumen:** Se copia un dispositivo.

**Actor:** El usuario.

**Precondición:** El programa se encuentra en modo de gestión de la base de datos.

**Descripción:**

- Trayectoria básica:
  1. El usuario pasa a modo de gestión de dispositivos.
  2. El sistema muestra una lista de fabricantes, de entre los cuales el usuario elige uno.
  3. A continuación se muestra una lista de los dispositivos del fabricante seleccionado. El usuario elige uno.
  4. El usuario indica que quiere copiar el dispositivo.
  5. El sistema le solicita que elija el fabricante del nuevo dispositivo.
  6. El sistema solicita al usuario el código de identificación del nuevo dispositivo.
  7. Se añade el nuevo dispositivo a la base de datos.
- Trayectorias alternativas:
  - Ya existía un dispositivo del mismo fabricante y código de identificación.
    1. Se muestra un mensaje de error al usuario y se aborta la operación.

**Postcondición:** Se ha añadido a la base de datos el nuevo dispositivo con los mismos parámetros que tenía el dispositivo original.

## *6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO*

---

**Caso de uso:** Eliminar dispositivo.

**Resumen:** Se elimina un dispositivo.

**Actor:** El usuario.

**Precondición:** El programa se encuentra en modo de gestión de la base de datos.

**Descripción:**

- Trayectoria básica:
  1. El usuario pasa a modo de gestión de dispositivos.
  2. El sistema muestra una lista de fabricantes, de entre los cuales el usuario elige uno.
  3. A continuación se muestra una lista de los dispositivos del fabricante seleccionado. El usuario elige uno.
  4. El usuario indica que quiere eliminar el dispositivo.

**Postcondición:** Se ha eliminado el dispositivo de la base de datos.

## 6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO

**Caso de uso:** Exportar dispositivo.

**Resumen:** Se guarda en fichero la información relativa a un dispositivo.

**Actor:** El usuario.

**Precondición:** El programa se encuentra en modo de gestión de la base de datos.

**Descripción:**

■ Trayectoria básica:

1. El usuario pasa a modo de gestión de dispositivos.
2. El sistema muestra una lista de fabricantes, de entre los cuales el usuario elige uno.
3. A continuación se muestra una lista de los dispositivos del fabricante seleccionado. El usuario elige uno.
4. El usuario indica que quiere exportar el dispositivo.
5. El sistema indica al usuario que suministre el nombre del fichero en el que desea guardar los datos del dispositivo.
6. El sistema guarda en el fichero la información correspondiente al dispositivo seleccionado.

■ Trayectorias alternativas:

- Se ha producido un error durante la escritura del fichero.
  1. Se muestra al usuario un mensaje de error y se aborta la operación.

**Postcondición:** El fichero indicado contiene ahora la información asociada al dispositivo seleccionado.

---

## *6.1. ANÁLISIS DE REQUISITOS. CASOS DE USO*

**Caso de uso:** Importar dispositivo.

**Resumen:** Se lee de un fichero la información relativa a un dispositivo.

**Actor:** El usuario.

**Precondición:** El programa se encuentra en modo de gestión de la base de datos.

**Descripción:**

- Trayectoria básica:
  1. El usuario pasa a modo de gestión de dispositivos.
  2. El sistema indica al usuario que suministre el nombre del fichero en desde el que desea recuperar los datos del dispositivo.
  3. El sistema lee del fichero la información correspondiente al dispositivo y la incorpora a la base de datos.
- Trayectorias alternativas:
  - Se ha producido un error durante la lectura del fichero.
    1. Se muestra al usuario un mensaje de error y se aborta la operación.

**Postcondición:** Se han incorporado a la base de datos la información almacenada en el fichero.

## 6.2. Análisis del software

El elemento principal de la aplicación será la memoria. Cada memoria estará asociada a un fabricante (identificados por un código y un nombre). Cada memoria tendrá un código que junto al código del fabricante la identificará únicamente. También será importante almacenar el tamaño y el número de pines que tiene la memoria. Las operaciones que permitirán todas las memorias serán las de identificación y lectura.

Cada memoria está asociada a un fabricante y pueden estar organizadas en sectores. Cada sector consta de una dirección de comienzo y un tamaño.

Las memorias serán de dos tipos: memorias de sólo lectura (ROM) y memorias programables. Éstas a su vez se pueden desglosar en memorias programables byte a byte y memorias programables por páginas. Las memorias programables ofrecerán funciones para su grabación, borrado y verificación. Las memorias programables por páginas deben especificar el tamaño de página.

Las memorias programables deben ofrecer un juego de comandos para iniciar y controlar el proceso de grabación. Cada juego de comandos, identificado por un nombre, consta de un conjunto de comandos, cada uno de los cuales consta de un identificador, una descripción y una secuencia de pares dirección/dato.

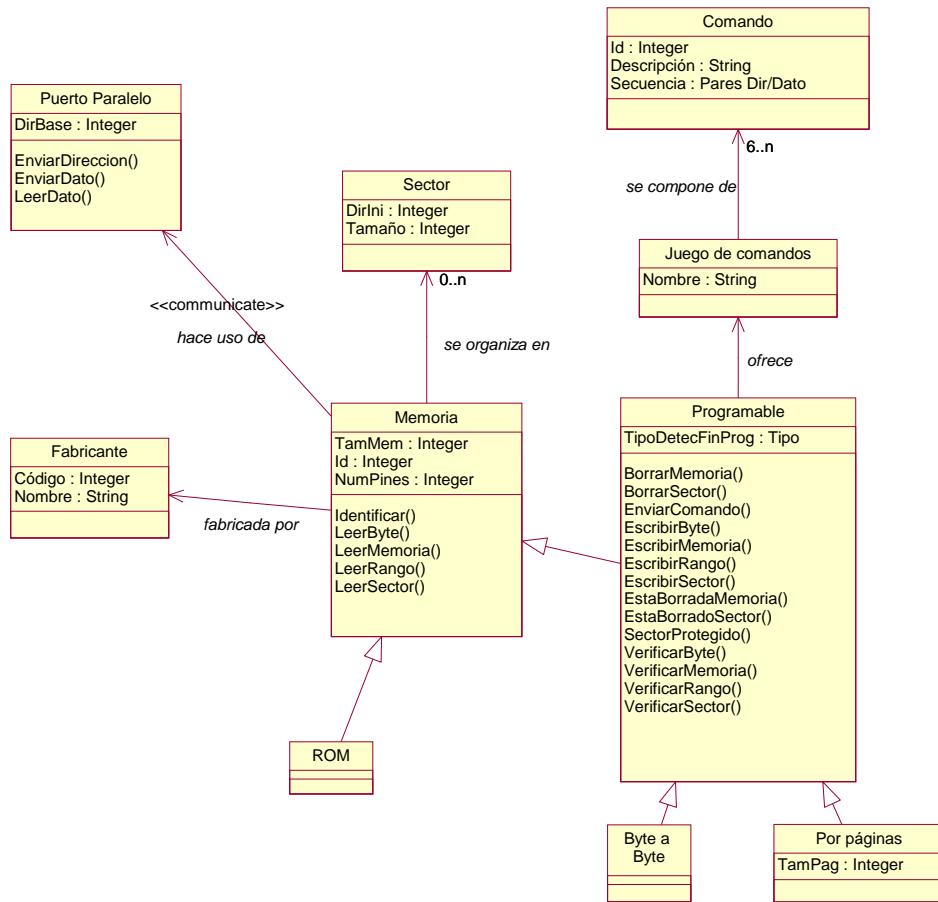
Para realizar la comunicación con la memoria física alojada en el programador, se hará uso de la clase Puerto Paralelo.

Todo esto queda reflejado en el modelo conceptual de la figura 6.1.

## 6.3. Diseño del software

La aplicación constará de dos partes principales en las que centraremos el diseño. Una de ellas es la base de datos que contendrá todas las memorias y sus características. La otra es el conjunto de rutinas de control de las operaciones que se realizan sobre la memoria. Cada una de estas partes contará con una serie de funciones de apoyo e interfaz con el usuario.

Todas las características de las memorias irán almacenadas en una base de datos relacional. Tendremos por un lado una lista de fabricantes y sus códigos de identificación y por otro una lista de dispositivos con sus características fundamentales (código de identificación, nombre, tamaño, número de pines, el tipo – de sólo lectura, programables por páginas, byte a byte, y mecanismos de indicación del estado de la grabación que soportan –, y el tamaño de página si es aplicable). Cada memoria puede tener asociada una lista de sectores, si es que está organizada de tal forma; como gran parte de las memorias suele tener sectores contiguos del



File: G:\Documents and Settings\luis\Mis documentos\programador.mdl 23:39:44 martes, 01 de julio de 2003 Class Diagram: Análisis / Modelo

Figura 6.1: Modelo conceptual

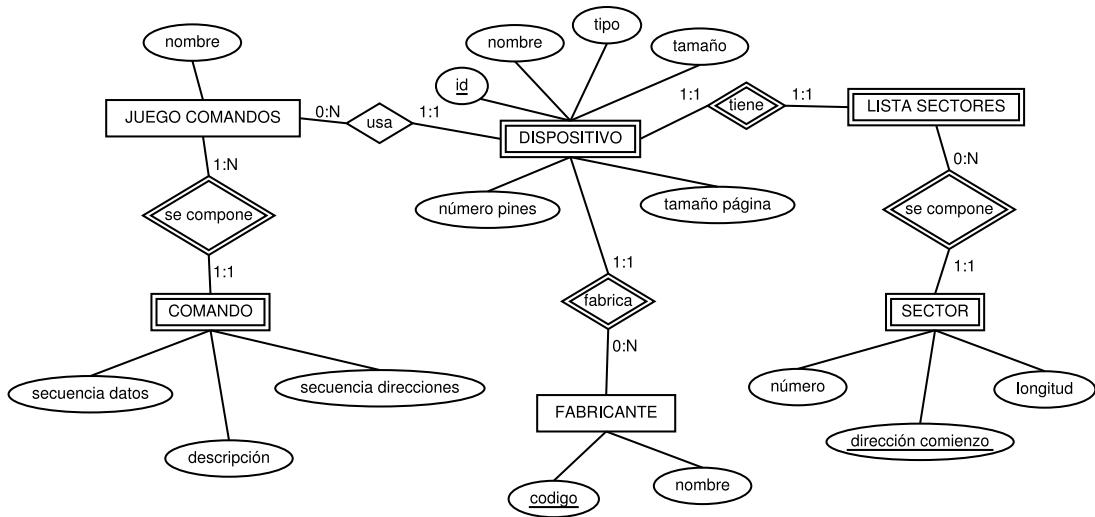


Figura 6.2: Diagrama E-R de la base de datos

mismo tamaño, en vez de almacenar cada uno de estos sectores por separado, almacenaremos la dirección de comienzo, el número y el tamaño de éstos. También habrá una serie de juegos de comandos y cada memoria estará asociada a uno de ellos. Todo esto queda reflejado en el diagrama de entidad-relación de la figura 6.2.

Una vez pasado este modelo de entidad-relación al relacional nos quedan las tablas de la figura 6.3.

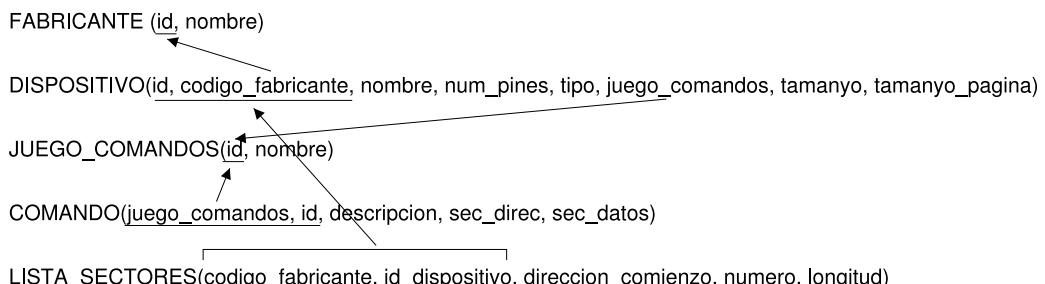


Figura 6.3: Modelo relacional de la base de datos

Las rutinas de control de las memorias serán implementadas por las clases **TMemoria** y **TMemProgramable**, que ofrecerán funciones transparentes al tipo de memoria (sólo lectura y programables por páginas o byte a byte y los diversos mecanismos de control que soporte) para leer, escribir y verificar direcciones dispersas, rangos de direcciones, sectores o la memoria completa, hacia o desde un *buffer* de memoria o un fichero. También ofrecerá métodos para borrar sectores o la memoria completa, comprobar el estado del sistema de protección contra escritura de los sectores, enviar comandos y obtener el código del fabricante y el identificador del modelo de la memoria.

Estas clases contarán con el apoyo de otras clases para almacenar las listas de sectores y comandos de la memoria con la que se esté trabajando en un determinado momento.

La comunicación con el puerto paralelo se realizará a través de la clase TParPort, que ofrece métodos para configurar el puerto paralelo a usar (estableciendo el puerto base de E/S), enviar direcciones, y enviar y recibir datos.

El diagrama UML de la figura [6.4](#) muestra estas clases y sus principales atributos y métodos. La descripción funcional de los métodos más complejos de la clase Flash se puede encontrar en el apartado de descripción del funcionamiento de las memorias flash del presente documento, en concreto en las figuras [2.10](#), [2.11](#), [2.12](#), [2.14](#) y [2.16](#).

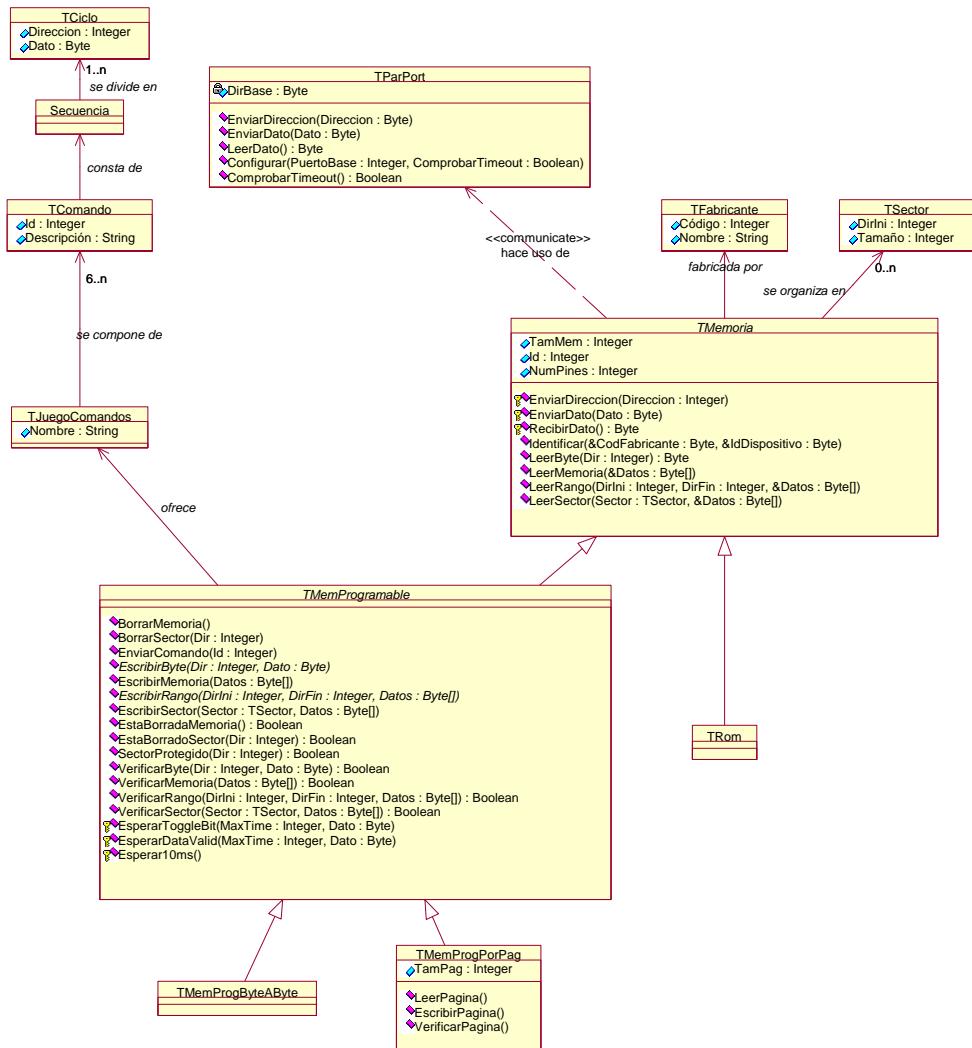
Por otro lado, la figura [6.5](#) muestra las clases de interfaz de la aplicación. La aplicación consta de 3 formularios principales: el formulario inicial y de control de memorias (TFormPrinc), el de configuración del puerto paralelo (TFormParPorConf) y el de gestión de la base de datos (TFormBD). Luego tenemos formularios complementarios como el de “Acerca de” y un formulario para la introducción de datos de nuevos dispositivos (TFormNDisp).

La ventana principal consta de varias pestañas: una para la selección del dispositivo con el que se va a trabajar (TSheetDispositivo), y otra para cada uno de los tipos de operaciones que se pueden realizar (TSheetLectura, TSheetEscritura, TSheetBorrado, TSheetComandos).

El formulario de gestión de la base de datos, se compone a su vez de otras tantas pestañas: una para la gestión de fabricantes (TSheetFabricantes), otra para la gestión de juegos de comandos (TSheetJuegosComandos) y una para la gestión de dispositivos (TSheetDispositivos).

En el apéndice [D.3](#) encontrará una especificación formal, usando diagramas de actividad como notación, del comportamiento del interfaz del usuario.

### 6.3. DISEÑO DEL SOFTWARE



File: \Mussi\varmiscl\failure\tmp\programador.mdl 1:02:14 miércoles, 02 de julio de 2003 Class Diagram: Control memorias / Control Page  
1

Figura 6.4: Clases de control

### **6.3. DISEÑO DEL SOFTWARE**

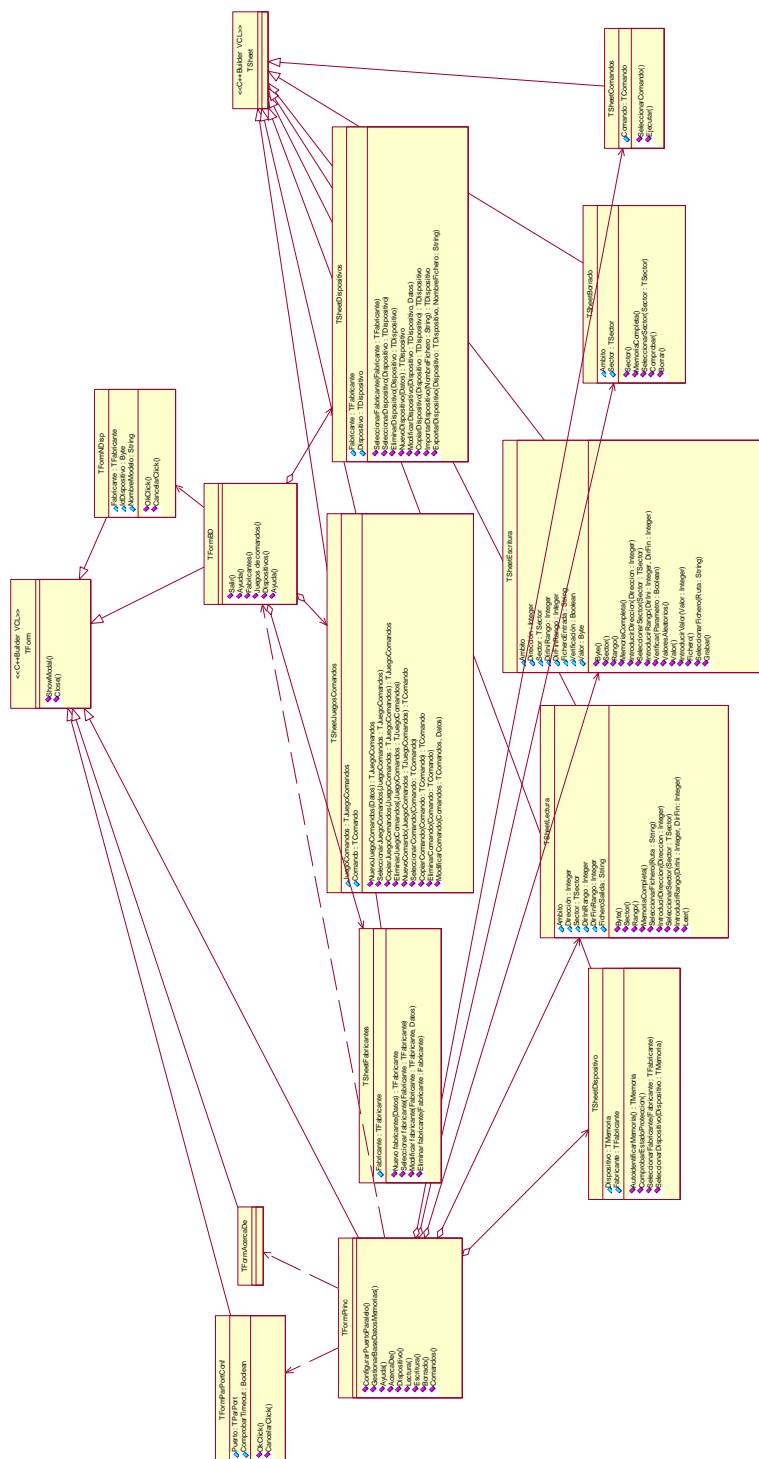


Figura 6.5: Clases de interfaz



# Capítulo 7

## Implementación del software

Para la implementación del software de control se usó el entorno de programación *C++ Builder* de Borland, por la versatilidad que nos ofrece a la hora de manejar bases de datos y crear interfaces de usuario. Además C++ hereda del lenguaje C su capacidad para, cuando es preciso, programar a un nivel cercano a la máquina.

Para la gestión de la base de datos se hizo uso del motor de base de datos BDE (*Borland Database Engine*) que nos ofrece el mismo entorno *C++ Builder*. BDE ofrece un acceso transparente y sencillo a varios formatos de ficheros de base de datos y gestores de bases de datos relacionales. En este caso se emplearon ficheros en formato *Paradox*, haciendo uso la herramienta *Database Desktop* del entorno para la definición y creación de las tablas.

Se escapa de los objetivos de este documento explicar el uso del entorno *C++ Builder* o los fundamentos de la programación en C/C++, por lo que nos limitaremos a comentar algunos detalles de la implementación de la aplicación.

### 7.1. Acceso a los puertos de E/S bajo Windows

Windows 9x permite el acceso directo a los puertos de entrada/salida del sistema usando las instrucciones *in* y *out* del procesador. Windows NT y sus derivados (Windows 2000 y Windows XP) al ser sistemas más seguros impiden el uso de estas instrucciones y consecuentemente el acceso a los puertos. Bajo estos sistemas operativos sólo los procesos en modo *kernel* pueden usar estas instrucciones. Por tanto, el acceso desde programas sin privilegios se debe realizar a través de controladores VxD (*Virtual Device Driver*). La programación de estos controladores no es nada trivial y exige mucho tiempo y dedicación, por lo que se buscaron implementaciones de terceros.

*DriverLINX Port IO*<sup>1</sup> es un controlador que permite que los programas puedan acceder a los puertos del sistema desde el 0x0100 al 0xFFFF de forma transparente bajo todas las variantes de Windows, desarrollado por una empresa llamada *Scientific Software Tools (SST), Inc.* Este software accede a los puertos directamente en Windows 9x y a través de un módulo en modo *kernel* en Windows NT y derivados. Este controlador puede ser lanzado bajo demanda, pero sólo desde una cuenta con privilegios de administrador. Si se desea que todos los usuarios tengan acceso a los puertos, el administrador lo puede instalar y configurar como servicio de NT, de forma que sea lanzado automáticamente al arrancar el sistema.

*TDLPortIO*[[12](#)] es un interfaz avanzado para el controlador *DriverLINX* que ofrece además funciones para la detección de los puertos paralelos en el sistema y gestiona automáticamente la carga del controlador en modo *kernel* cuando sea necesario (sólo si se tienen privilegios de administrador). Ofrece interfaces como componentes *Delphi*, *C++ Builder* y *Active X (OCX)*, así como una biblioteca de funciones DLL para su uso mediante el API Win32.

Para el desarrollo de la aplicación se usó la componente para *C++ Builder*, tras recompilar el código para la versión 5 de este entorno de programación, ya que los binarios que venían en el paquete software eran para la versión 4. Comentar además que se corrigió un fallo en las funciones de detección de puertos paralelos que causaba frecuentes accesos a direcciones de memoria ajena al espacio de direccionamiento de la aplicación, causando la interrupción de ésta.

## 7.2. Tiempo real

Aunque estrictamente Windows no es un sistema operativo de tiempo real, el API Win32 ofrece una función para cambiar la prioridad de un proceso: *SetPriorityClass*. Asignando a nuestro proceso una prioridad *REALTIME\_PRIORITY\_CLASS* nos aseguramos de que nuestro programa, mientras esté realizando una operación crítica, será interrumpido lo menos posible, teniendo preferencia incluso sobre algunos procesos del sistema operativo tales como el control del ratón o el volcado de *buffers* de escritura a disco. Una vez acabada esta operación crítica, debemos volver a prioridad normal (*NORMAL\_PRIORITY\_CLASS*).

Esta funcionalidad nos será muy útil en la grabación de memorias por páginas, reduciendo el riesgo de que se tarde demasiado en enviar el siguiente byte de una página y salte la grabación de ésta. Pero como dijimos antes, Windows no es un sistema operativo de tiempo real, por lo que no se nos garantiza que nuestro proceso no sufrirá interrupciones; además las especificaciones IEEE-1284 tampoco nos garantizan tiempos de respuesta del puerto paralelo muy pequeños.

---

<sup>1</sup><http://diskdude.cjb.net/files/cbuilder/DLPortIO/port95nt.exe>

Por estas dos razones es posible que la grabación de una página se inicie antes de tiempo, por lo que nuestro programa verificará la correcta grabación de cada página, y en caso necesario, la repetirá hasta un determinado número de veces. Veamos un trozo del código de grabación de páginas:

```
CmpOk = false;
ResGrab = SIN_ERROR;
Intentos = 0;
while (!CmpOk && (ResGrab == SIN_ERROR) && (Intentos < MAX_INTENTOS))
{
    Intentos++;
    SetPriorityClass(GetCurrentProcess(),REALTIME_PRIORITY_CLASS);
    EnviarComando(C_PROGRAM);
    for(Cont = 0 ; Cont < TamPag ; Cont++, Dir++)
    {
        EnviarDireccion(Dir);
        EnviarDato(Datos[Cont]);
    }
    SetPriorityClass(GetCurrentProcess(),NORMAL_PRIORITY_CLASS);
    EsperarTblco();
    ResGrab=EsperarProgramacion(TMAX_PAGE_PROG,Datos[Cont]);
    CmpOk = VerificarPagina(Dir,Datos);
}
```



# **Capítulo 8**

## **Pruebas**

Además de las pruebas parciales que se fueron realizando tanto de la placa durante su montaje como del software durante su desarrollo, se realizó una serie de pruebas finales del sistema completo, de las cuales se comentarán a continuación las más significativas.

### **8.1. Ruido en la alimentación**

Las primeras mediciones realizadas revelaron la presencia de ruido en las líneas de alimentación y, a causa de éste, en las de las señales de datos y control. En un primer intento de reducirlo se colocaron condensadores de 330pF conectados a tierra en las principales señales. Ciertamente se redujo el ruido pero más tarde se comprobó que en ciertas ocasiones fallaba la grabación de un dato, y tras muchas pruebas con la ayuda del analizador lógico se demostró que este fallo era debido a que en estas ocasiones se producía, debido al retardo extra que introducían los condensadores, un pequeño pulso de lectura que abortaba las escrituras.

Finalmente se eliminaron estos condensadores y nos contentamos con suavizar un poco el ruido colocando un par de condensadores de 0.1nF en la fuente de alimentación, justo antes y después del regulador de tensión.

### **8.2. Comprobación del retardo**

Con una memoria colocada en el zócalo se fueron tomando medidas de los tiempos de retardo de nuestro elemento retardante en cada una de las combinaciones de los *jumpers* que controlan su duración. Lo primero que salta a la vista es que el retardo es simétrico, con diferencias despreciables entre los tiempos de bajada y de subida, como se puede apreciar en la figura 8.1.

La figura 8.4 y la tabla 8.1 muestran el retardo que se produce en la señal en cada una de las 4 combinaciones que permiten los *jumpers*. Como se puede apreciar, en la práctica los valores de retardo son sensiblemente inferiores a los valores teóricos, con desviaciones de hasta 21 ns. Como resultado, en las dos primeras posiciones el retardo es bastante inferior al mínimo que calculamos que era necesario.

Combinación	Posición <i>jumpers</i>	Tiempo medido	Tiempo calculado
1	↓↓	59 ns	68 ns
2	↓↑	70 ns	84 ns
3	↑↓	86 ns	102 ns
4	↑↑	97 ns	118 ns

Tabla 8.1: Tiempos de retardo

Sin embargo, estudiando la señal *nWait* (figura 8.2) se observa que la corta duración del retardo se ve ampliamente compensada por la más que apreciable capacitancia que las entradas del puerto paralelo añaden al circuito resultando ahora, incluso, que en la cuarta posición el tiempo de retardo (tabla 8.2) sea notablemente superior al valor máximo que permiten las especificaciones del protocolo EPP (aunque en la práctica parece ser que el puerto paralelo es algo más tolerante, permitiendo que incluso en esta posición las operaciones de entrada y salida se realicen correctamente sin siquiera producirse un *timeout*).

Combinación	Posición <i>jumpers</i>	Tiempo medido
1	↓↓	96 ns
2	↓↑	107 ns
3	↑↓	124 ns
4	↑↑	135 ns

Tabla 8.2: Tiempos de retardo entre *nDataStrobe* y *nWait*

Otros valores interesantes de conocer son los tiempos de retraso de las señales *OE* y *WE* respecto a *nDataStrobe*: 21ns en ambos casos (figura 8.3).

Con todos los valores medidos ya podemos comprobar el cumplimiento de las condiciones necesarias para el correcto funcionamiento del circuito. Estos requisitos según ya vimos en el apartado 4.1.6 son los siguientes.

### Requisitos necesarios para las operaciones de lectura

Por un lado, para que incluso a las memorias más lentas les de tiempo acceder al dato, es necesario que el retraso total de *nWait* respecto a *OE* sea mayor que el tiempo de acceso de éstas. Esto se cumple para la primera combinación del retardo (y por tanto para el resto):

Retardo  $nWait$  - retardo  $OE$  = 96ns - 21ns = 76ns > 60ns = T. acceso mem. más lentas

La otra condición es que desde la subida de  $nDataStrobe$  hasta la bajada de  $nWait$  no pasen más de 125ns. Esta condición sólo la cumple las tres primeras combinaciones, ya que en la cuarta este tiempo alcanza un valor de 134ns (aunque como hemos comentado antes, el puerto paralelo no parece ser muy estricto en este sentido).

### Requisitos necesarios para las operaciones de escritura

En las operaciones de escritura debían cumplirse cuatro condiciones.

La primera es que hay que respetar el tiempo de *data setup* de las memorias. Esto implica que el tiempo de retardo de  $nWait$  más el de  $WE$  debe ser superior a 55ns en los peores casos. Esta condición es satisfecha con un amplio margen en todas las combinaciones.

La segunda condición es que el ancho del pulso de  $WE$  sea superior a 70ns. Como el ancho de este pulso será al menos igual al tiempo de retardo de  $nWait$ , tenemos que se cumple este requisito en todas las configuraciones del retardo..

Otra condición es que el retraso de  $nWait$  respecto a  $nDataStrobe$  sea mayor que el retraso entre  $WE$  y  $nDataStrobe$  para evitar que se retiren los datos del bus antes de que la memoria los capture. Incluso en la combinación de menor retardo de  $nWait$  se cumple esta condición.

Y por último, al igual que en las lecturas, desde la subida de  $nDataStrobe$  hasta la bajada de  $nWait$  no deben pasar más de 125ns. Como ya vimos, esto sólo se cumple en las tres primeras combinaciones.

En resumen, para que el circuito funcione correctamente y se cumplan estrictamente los requisitos necesarios, se debe usar alguna de las tres primeras configuraciones del retardo.

Las figuras 8.5 y 8.6 muestran ciclos completos de escritura de dirección, y de lectura y escritura de datos capturados con el analizador lógico.

## 8.2. COMPROBACIÓN DEL RETARDO

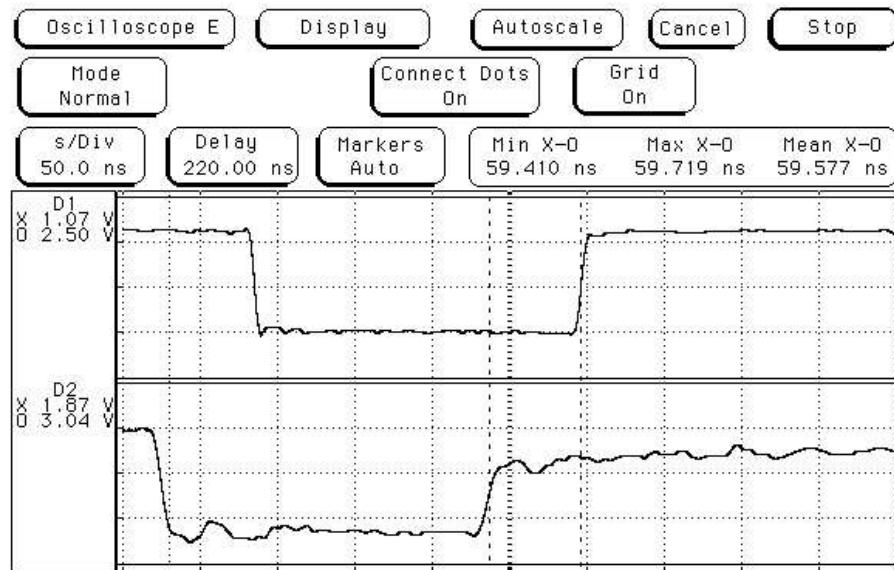
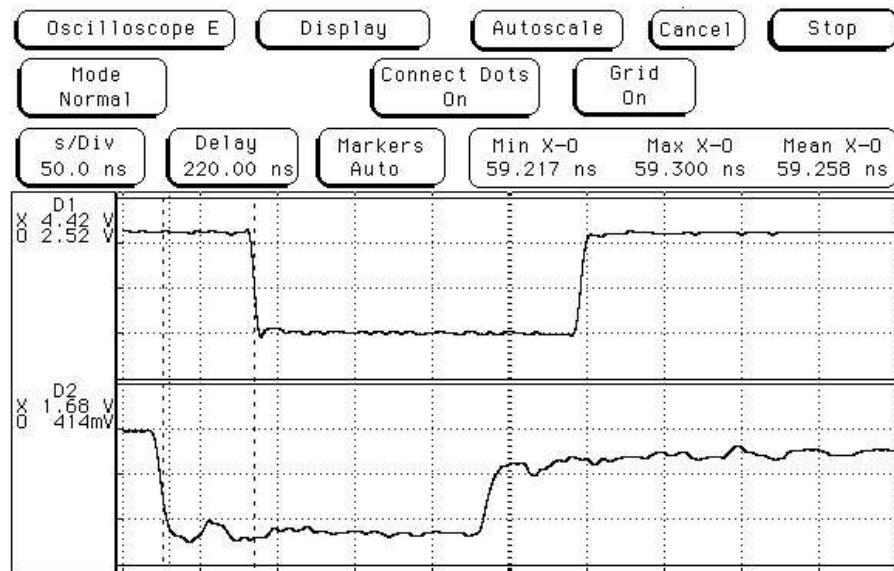


Figura 8.1: Simetría del retardo

## 8.2. COMPROBACIÓN DEL RETARDO

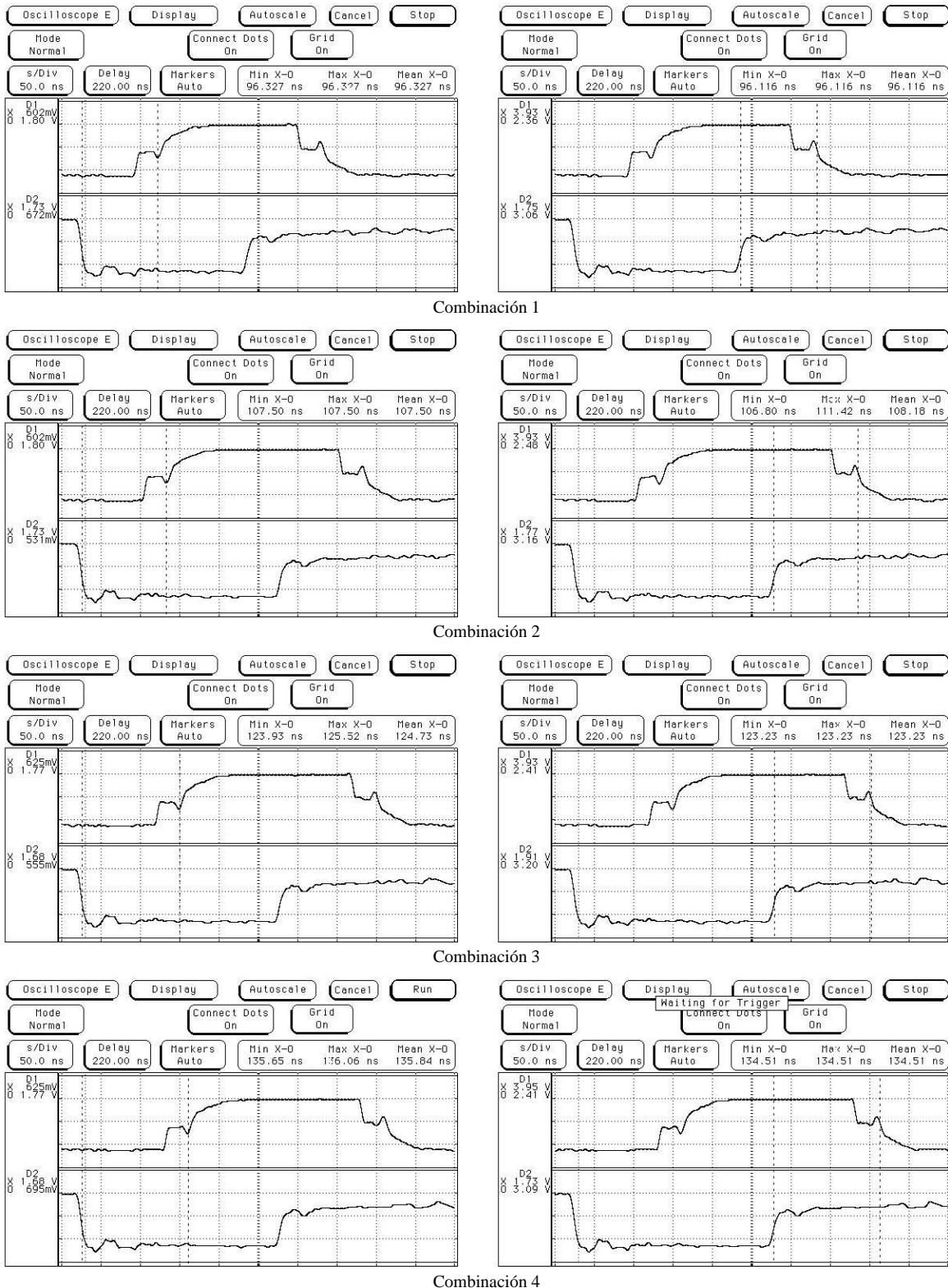


Figura 8.2: Retardo  $nWait$  respecto  $nDataStrobe$

## 8.2. COMPROBACIÓN DEL RETARDO

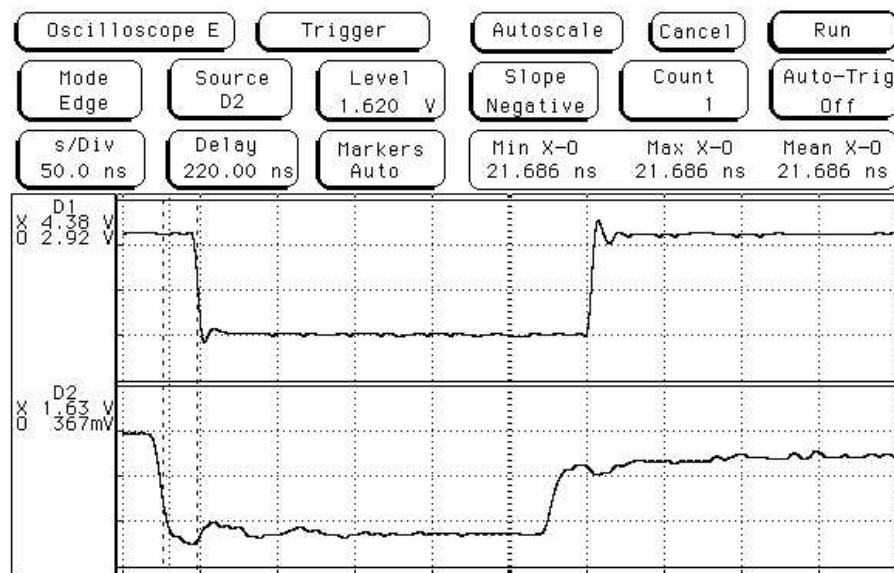
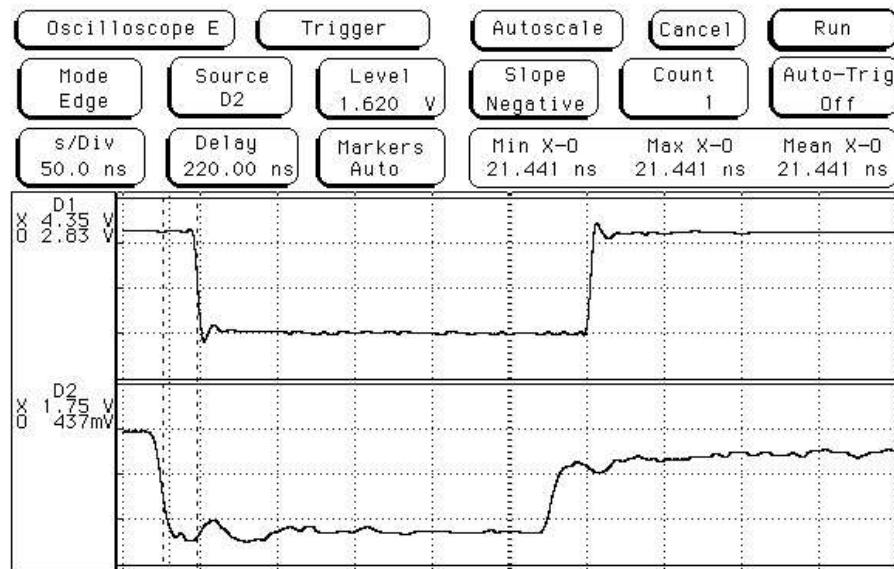


Figura 8.3: Retardo *OE* y *WE* respecto *nDataStrobe*

## 8.2. COMPROBACIÓN DEL RETARDO

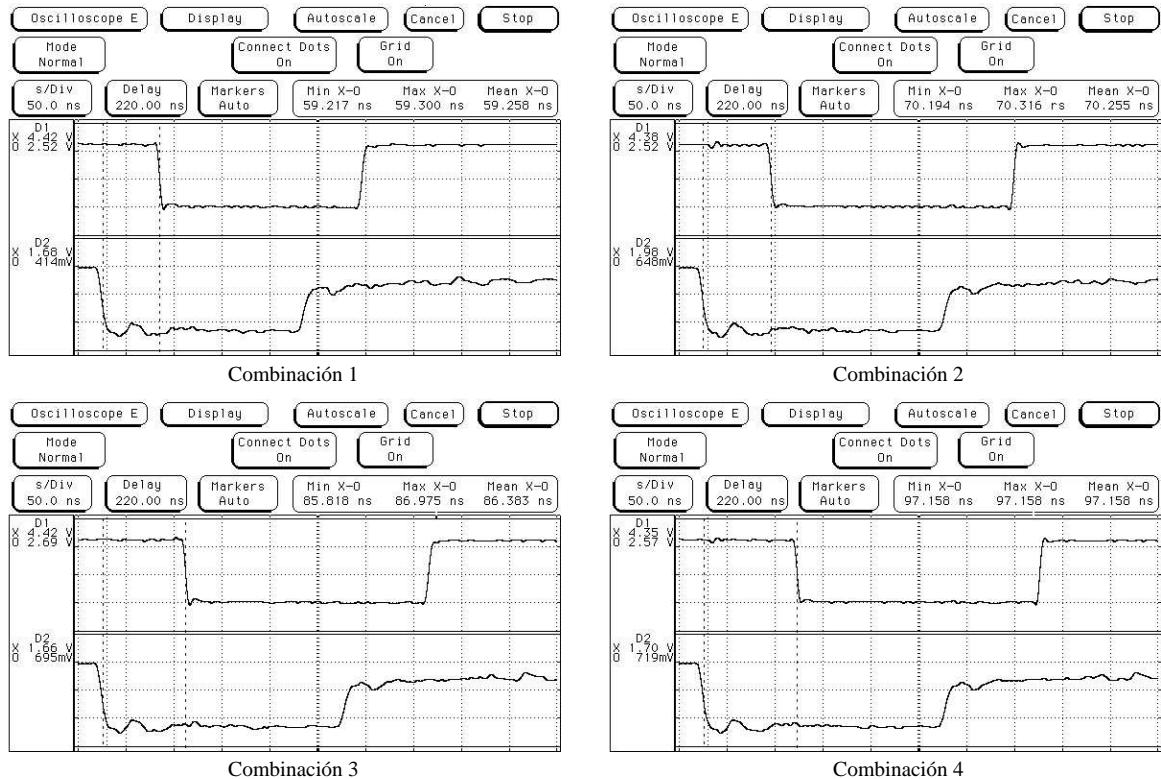


Figura 8.4: Efectos del retardo sobre la señal

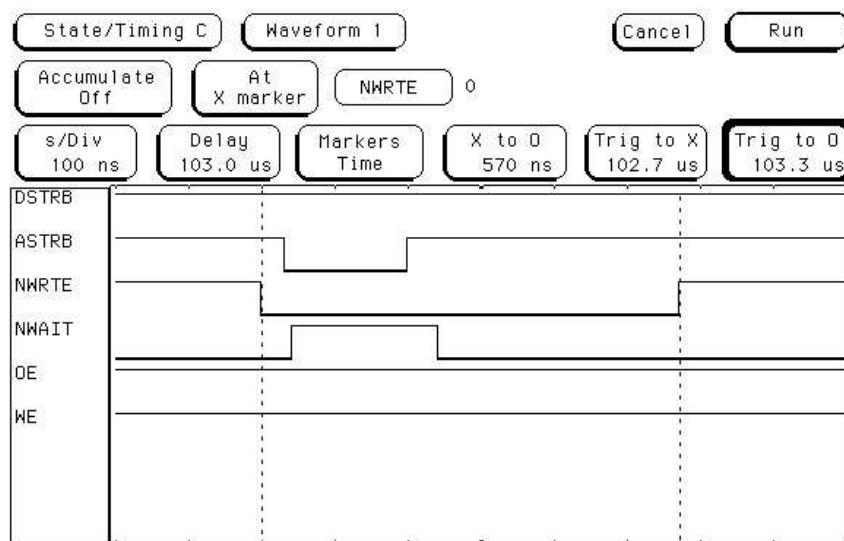


Figura 8.5: Ciclo de escritura de dirección

## 8.2. COMPROBACIÓN DEL RETARDO

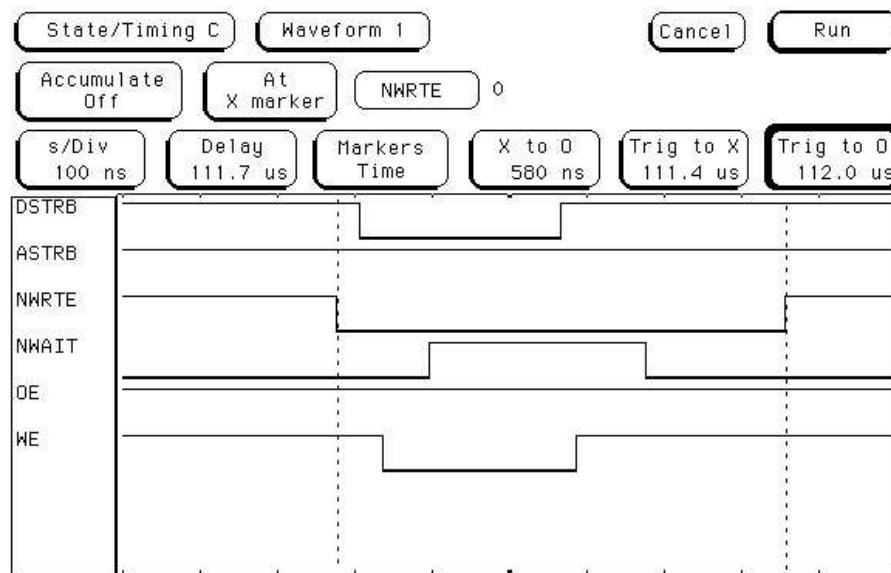
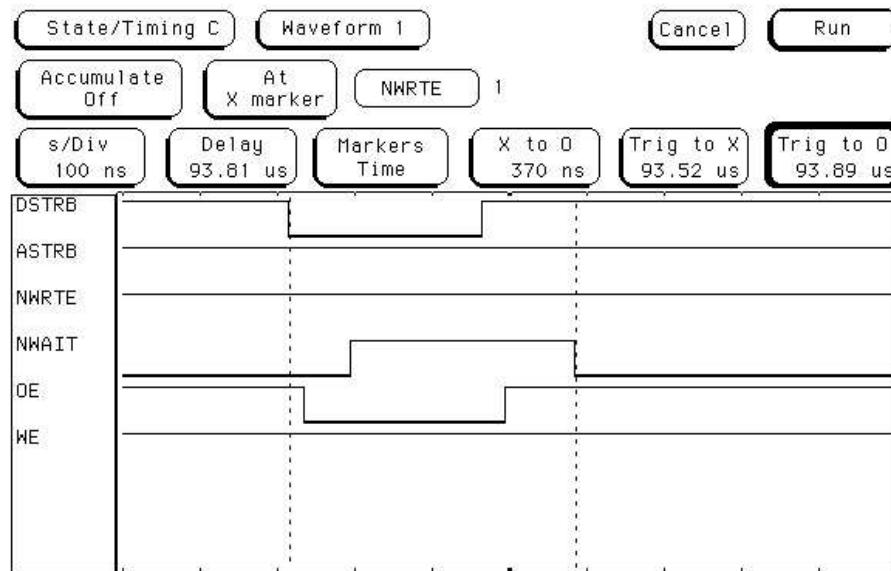


Figura 8.6: Ciclos de lectura y escritura de datos

### 8.3. Pruebas de lectura y escritura

Durante las primeras pruebas de lectura y escritura se detectaron eventuales comportamientos anómalos en el proceso. El analizador lógico fue de gran ayuda en estos casos para descubrir que eran debidos a fallos en el código, siendo localizados y subsanados.

Una vez superados los problemas iniciales se procedió a realizar pruebas exhaustivas de cada una de las opciones de lectura, escritura y borrado que ofrece el programa, con distintos tipos y tamaños de memoria.

Las figuras 8.7 y 8.8 muestran respectivamente un ciclo de lectura y un ciclo de escritura de un byte. En la primera se diferencian claramente las tres escrituras de dirección para cargar en los registros la dirección a leer y la lectura del dato. En la segunda se aprecia el envío de la secuencia de comienzo de grabación (cada una consta de 3 escrituras de dirección y una escritura de dato), el envío de la dirección y el dato a grabar, y una serie de lecturas para detectar la finalización de la grabación.

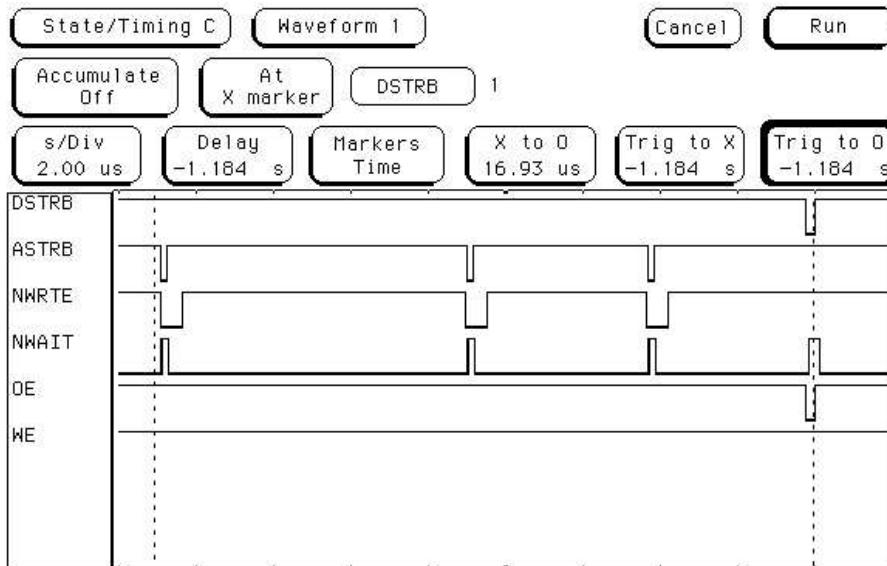


Figura 8.7: Captura ciclo de lectura de un byte

Una vez verificado que todo funcionaba correctamente se probó el programador en diversos equipos PC y con distintas versiones del sistema operativo Windows, tomando medidas de los tiempos de lectura y grabación. Los resultados se muestran en la tabla 8.3.

### 8.3. PRUEBAS DE LECTURA Y ESCRITURA

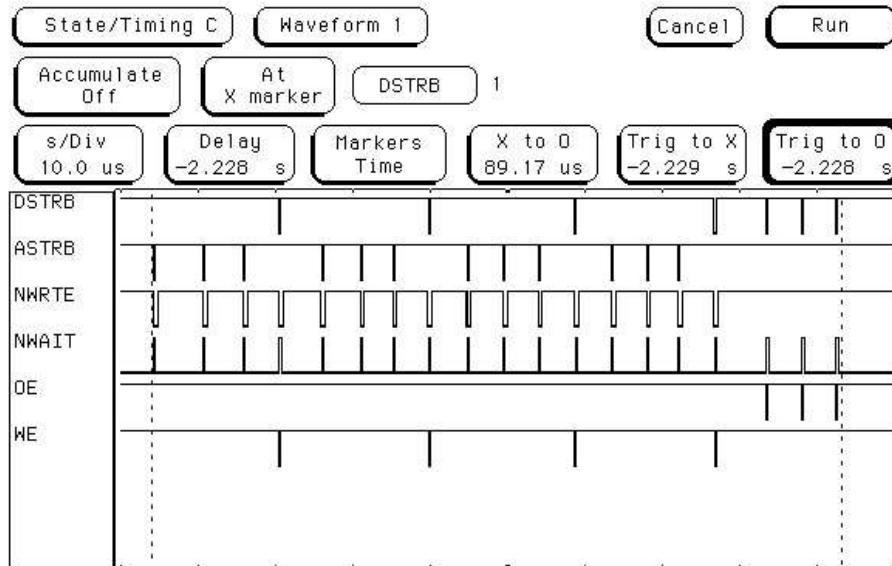


Figura 8.8: Captura ciclo de escritura de un byte

Configuración equipo	Sistema operativo	Tiempo lectura	Tiempo escritura
Pentium 133MHz - EPP 1.7 <sup>(a)</sup>	Windows 98	6 s	28 s
Pentium II 300MHz - EPP 1.7 y 1.9	Windows 95	5 s	24 s
Pentium III 1.0GHz - EPP 1.7 <sup>(b)</sup>	Windows 2000	5 s	24 s
Pentium IV 1.4GHz - EPP 1.7 y 1.9	Windows 2000	4 s	22 s
Pentium IV 1.6GHz - EPP 1.7 <sup>(c)</sup>	Windows 2000 y 98SE	4 s	22 s

(a) En la BIOS no se indica la versión de EPP, pero debe ser la 1.7 (placa base del año 1996).

(b) Con la versión 1.9 no funciona. Es posible que se deba a un fallo en la BIOS.

(c) En la BIOS no se indica la versión de EPP. Según el manual de la placa base implementa la versión 1.7.

Tabla 8.3: Tiempos de lectura y escritura en distintos equipos

# **Capítulo 9**

## **Conclusiones y mejoras posibles**

### **9.1. Conclusiones**

Consideramos que se han cumplido los objetivos que nos marcamos para la realización de este proyecto: se ha diseñado un programador lo suficientemente genérico como para ser capaz de leer y escribir un amplio surtido de memorias Flash de 5 voltios con encapsulado DIP, de montaje relativamente sencillo, con componentes fáciles de encontrar y de bajo coste.

La nota didáctica que nos ha aportado la realización de este proyecto ha sido amplia y variada. Durante su desarrollo se ha aprendido o puesto en práctica:

1. Funcionamiento de las memorias Flash.
2. Funcionamiento y uso del puerto paralelo en modo EPP.
3. Uso de herramientas y técnicas de trazado de placas PCB.
4. Fabricación y montaje de placas PCB.
5. Uso del analizador lógico y osciloscopio digital.
6. Análisis y diseño orientado a objetos con notación UML.
7. Análisis y diseño de bases de datos.
8. Programación en C++ bajo Borland *C++ Builder*.

La realización de este proyecto nos permite concluir que el puerto paralelo de un PC puede ser una herramienta muy versátil para el control de dispositivos simples, especialmente para

el aficionado a la electrónica digital que, con medios limitados, no puede permitirse caros dispositivos ni tarjetas de interfaz avanzadas. No obstante, también ha dejado patente que tiene considerables problemas de ruido y no se caracteriza precisamente por tener una respuesta rápida a la excitación de sus entradas.

## 9.2. Mejoras posibles

## Retardo con monoestables

Otra posible forma de generar  $nWait$  es mediante el uso de un par de monoestables activados por flanco y con entrada de reset. Por ejemplo podríamos usar un 74HCT123, que contiene dos monoestables independientes, con la configuración de la figura 9.1.

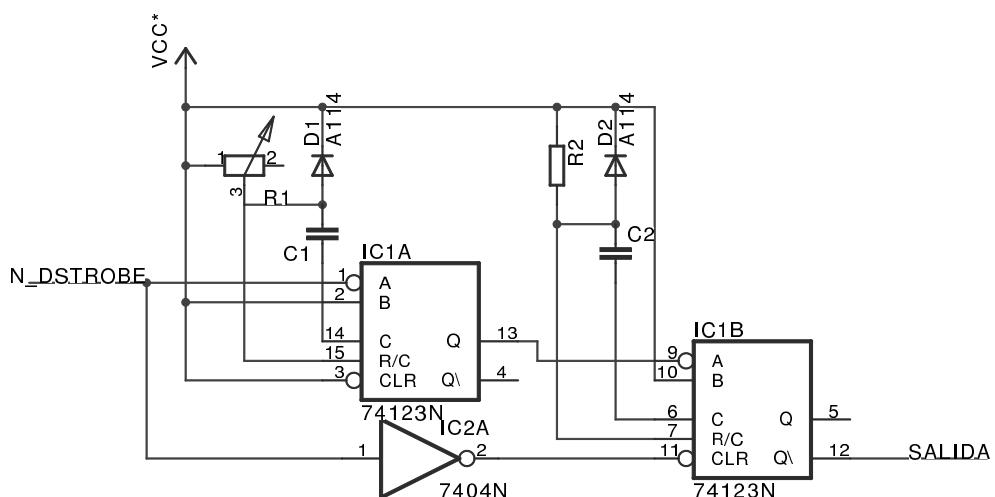


Figura 9.1: Retardo de  $nWait$  con monoestables

El flanco de bajada de  $nDataStrobe$  produce en la salida  $Q$  del primer monoestable un pulso ascendente de una longitud determinada por los valores de la resistencia variable y el condensador (el diodo, de germanio o de tipo Schottky, sólo sirve para que el condensador se descargue a través de él al apagar el circuito, evitando dañar así el monoestable). A su vez, el flanco de bajada de este pulso activará el segundo monoestable, generándose un pulso descendente en  $Q'$  de una longitud de hasta un segundo (para ello usaremos una resistencia R2 y un condensador C2 lo suficientemente grandes). La señal de reset (activa a 0) de este segundo monoestable está conectada a la función inversa de  $nDataStrobe$ , por lo que la vuelta a 1 de

ésta producirá un reset del segundo monoestable, retirándose por tanto la señal *nWait* cuando lo haga *nDataStrobe*, con un retardo igual a la suma del retardo de la puerta NOT (8ns) y el del reset del monoestable (23ns), es decir, de 31ns.

Por tanto, tenemos que el circuito genera un pulso con un retardo de bajada determinado por el primer monoestable (y ajustable gracias al potenciómetro) y un retardo de subida de 31ns que entra dentro de los márgenes que debía cumplir nuestro elemento retardante.

### Memorias flash de 12V

Se podría estudiar una forma sencilla de suministrar los 12V necesarios para programar este tipo de memorias, aunque sería necesaria la activación de algún conmutador por parte del usuario, ya que no tenemos salidas libres en el puerto paralelo para activar ningún tipo de relé o mecanismo similar.

### Protección/desprotección de memorias y sectores

Son muy pocas las memorias que permiten la protección y desprotección contra escritura mediante simples comandos. Se podría estudiar la viabilidad de modificar nuestro circuito para aplicar los voltajes necesarios para la protección y desprotección de algunos modelos con mecanismos más complejos.

### Memorias de mayor capacidad y otros encapsulados

La adaptación de nuestro circuito a memorias de mayor capacidad debería ser algo bastante sencillo. Nuestro circuito divide las líneas de dirección en tres grupos, en uno de los cuales hay 5 bits que no se usan. Usando estos 5 bits podríamos direccionar memorias de hasta 128Mbits, organizadas en palabras de 8 bits. También hay una gran variedad de modelos que permiten una organización dual en palabras de 8 bits o de 16 bits que podríamos leer y grabar en el modo de funcionamiento a 8 bits.

Cada vez es más frecuente encontrar memorias flash organizadas en palabras de 16 o 32 bits y esto exigiría cambiar la forma en que trabaja nuestro circuito en las transferencias de datos (necesitaríamos multiplexar el bus de datos).

En cuanto al soporte de otros encapsulados no debería suponer mayor problema que el de localizar los zócalos apropiados y la mayor dificultad a la hora de soldar éstos.

## **Memorias flash de menores voltajes**

Usando buffers de la familia 74LCX podríamos modificar nuestro circuito para leer y programar memorias Flash de 3.3V. Esta familia está formada por dispositivos CMOS de 3.3V con entradas que toleran niveles CMOS de 5V si ofrecen en sus salidas, a nivel alto, un voltaje igual a  $V_{cc}$  (lo que nos permite excitar tanto circuitos de 3.3V como 5V).

Podríamos usar por ejemplo un 74LCX244 (buffer unidireccional) para adaptar las señales de control. Para las líneas de datos podríamos usar un 74LCX245 (buffer bidireccional), ya que en teoría las entradas del puerto paralelo, al ser TTL, deben interpretar correctamente los 3.3V como nivel alto. Pero en la práctica los puertos paralelos son muy ruidosos y sensibles a las capacitancias parásitas, así que habría que ensayar la viabilidad de esta opción. En caso de que no fuera factible tenemos otra alternativa: montar un buffer bidireccional con dos 74LCX244 invertidos el uno respecto al otro, con el buffer que excite al puerto paralelo alimentado a 5V.

Asimismo disponemos de la familia 74VCX, que nos permite trabajar con voltajes entre los 1.8 y 3.3V, de forma que podríamos adaptar nuestro circuito para programar también memorias flash de 1.8V, usándola en combinación con la 74LCX.

La alimentación adicional de 3.3V o 1.8V la podríamos obtener bien añadiendo reguladores de tensión de estos niveles o bajando los 5V de nuestra actual fuente de alimentación mediante diodos dispuestos en serie.

# **Parte III**

## **Apéndices**



# **Capítulo A**

## **Manual del usuario**

### **A.1. Conexión del programador**

Para conectar el programador al puerto paralelo del PC necesitará un cable de impresora bidireccional (prácticamente cualquier cable de impresora relativamente reciente). Conecte el extremo de 25 patillas al ordenador y el otro al programador.

Conecte una fuente de alimentación de 7.5 a 9 voltios, que sea capaz de suministrar al menos 300mA y con conector de alimentación de 2.1mm de diámetro interior y 5.5mm de diámetro exterior. El programador acepta cualquier polaridad.

### **A.2. Instalación del software**

Si va a usar este software bajo Windows95/98/ME, o bajo Windows NT/2000/XP usando una cuenta con privilegios de administrador, el proceso de instalación se reduce a introducir el cdrom del software en su unidad de CD, ejecutar el programa *setup.exe* y seguir a continuación las instrucciones que se le mostrará en pantalla.

Si está instalando este software en una máquina con Windows NT, 2000 o XP y desea que los usuarios sin privilegios de administración también puedan hacer uso de él, deberá instalar además el servicio de sistema del controlador *DriverLINX*. Para ello, una vez realizado el proceso descrito anteriormente, diríjase al menú de *Inicio*, seleccione la carpeta *Programas*, luego *Programador FLASH* y pulse en *Instalar DriverLINX*.

## A.2. INSTALACIÓN DEL SOFTWARE

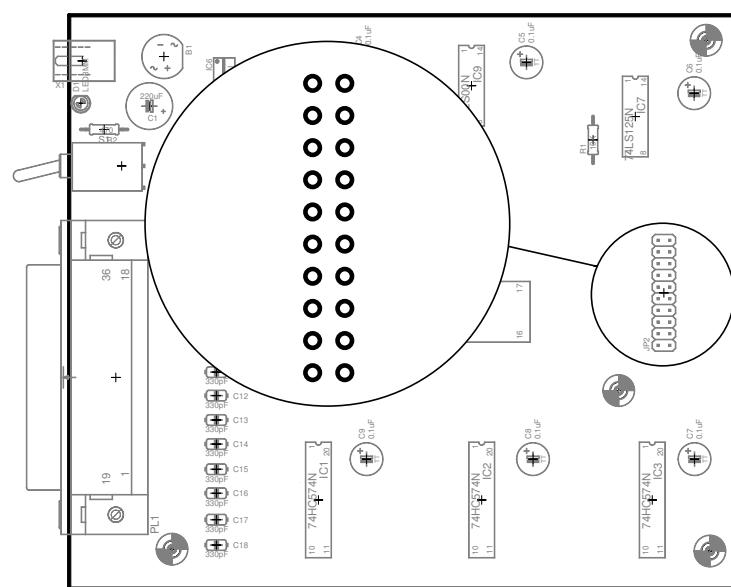


Figura A.1: Ubicación de los *jumpers* de selección de memoria

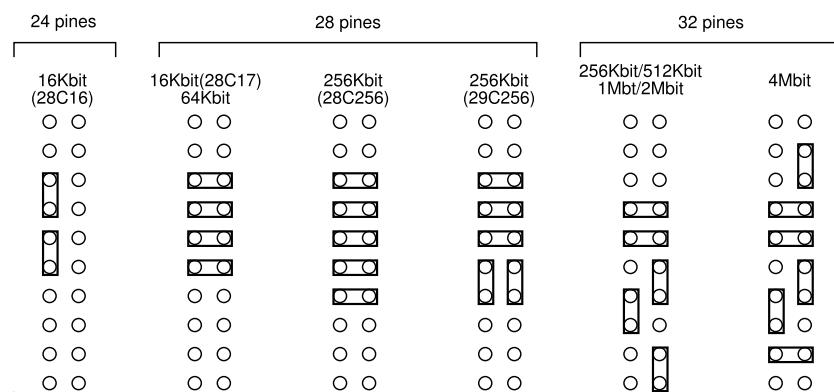


Figura A.2: Configuración de *jumpers* para memorias Flash

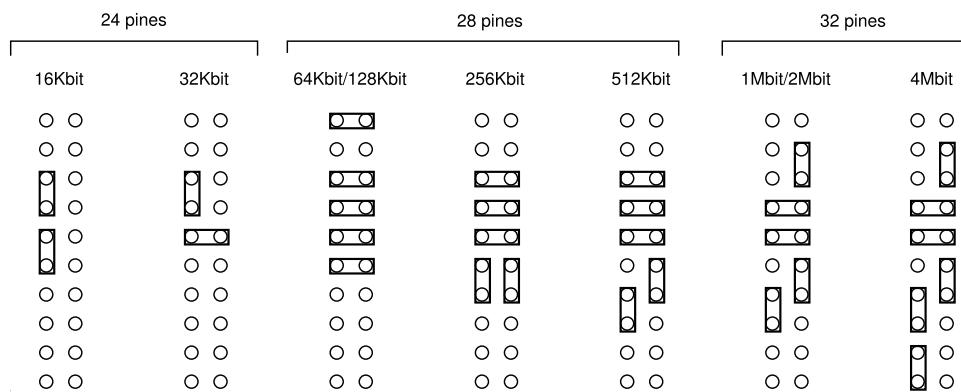


Figura A.3: Configuración de *jumpers* para memorias EPROM

## A.3. Colocación de la memoria

Antes de colocar la memoria hay que adaptar el circuito al tipo y tamaño de la memoria. Esto se hace configurando el grupo de *jumpers* que encontrará la derecha de la placa (figura A.1). Las figuras A.2 y A.3 muestran cómo deben colocarse los puentes para las de memorias Flash y EPROM (éstas sólo para lectura) compatibles con este programador. La orientación del grupo de pines en estos esquemas es la misma que se aprecia en la figura A.1.

La colocación y extracción de la memoria debe realizarse siempre con el circuito apagado para evitar causar daños a ésta. La orientación de la muesca del integrado debe coincidir con la orientación de la muesca en el zócalo. Los integrados de menos de 32 patillas se colocarán haciendo coincidir los extremos sin muesca del integrado y del zócalo.

## A.4. Manejo del programador

### A.4.1. Selección del puerto paralelo

Pulse en *Opciones* y elija la entrada de menú *Configurar puerto paralelo*. Seleccione el puerto de la lista desplegable.

### A.4.2. Selección del modelo de memoria (fig. A.4)

La primera pantalla que nos muestra la aplicación nos permite seleccionar el modelo de dispositivo que vamos a leer o grabar. La selección podemos hacerla de forma manual eligiendo en las listas desplegables el fabricante y modelo o intentando identificar automáticamente la memoria pulsando en *Autoidentificar*. Si la memoria soporta esta función mediante alguno de los mecanismos implementados en la aplicación nos devolverá un par de códigos que identifican únicamente al fabricante y modelo; si este modelo en concreto se encuentra en la base de datos se nos mostrará inmediatamente sus características, igual que si hubiéramos seleccionado el dispositivo manualmente.

Entre la información que nos muestra tenemos el número de sectores en que está organizada la memoria (si no está organizada en sectores se nos mostrará un 0), el tipo de memoria (si es programable byte a byte, programable por páginas o bien se trata de una memoria de sólo lectura o ROM) y el tamaño total. En el caso de que la memoria esté organizada en sectores se nos mostrará una lista con el tamaño, dirección de comienzo y final de cada uno de los sectores. Pulsando en *Estado Protec.*, y siempre que la memoria soporte esta característica, se nos mostrará el estado del sistema de protección de cada uno de los sectores.

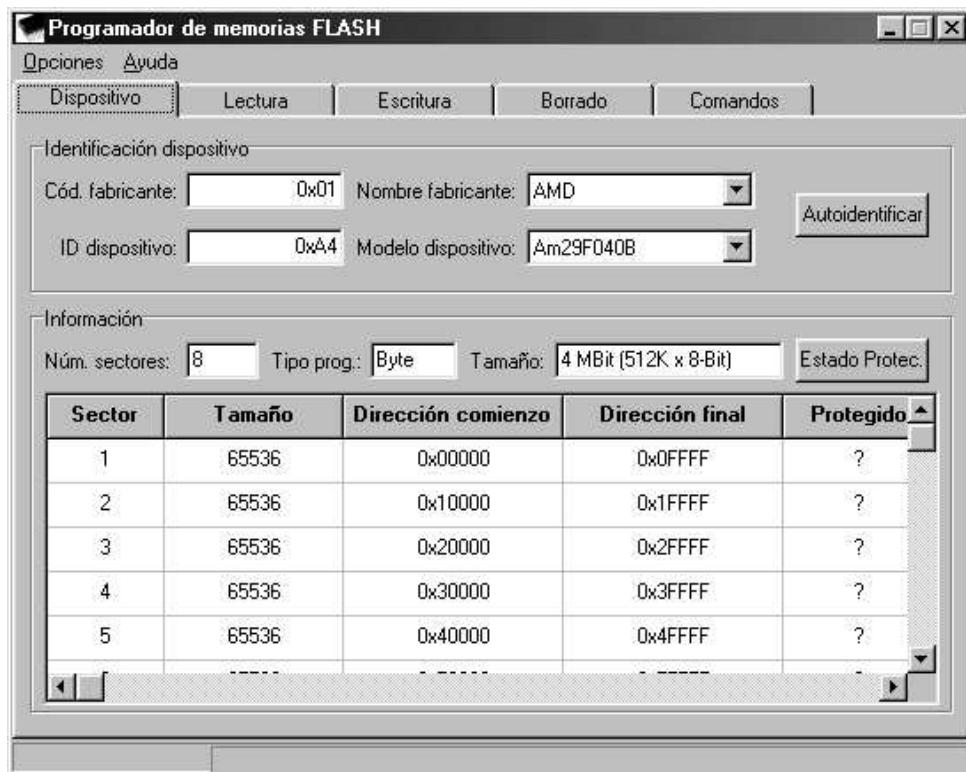


Figura A.4: Ventana de información sobre memorias

### A.4.3. Lectura de una memoria (fig. A.5)

#### Lectura de una dirección de memoria

Seleccione la opción *Byte* e introduzca en la casilla correspondiente la dirección a leer. A continuación pulse *Leer* y en *Valor* aparecerá el dato almacenado en dicha posición de memoria.

#### Lectura de un sector

Seleccione la opción *Sector* y elija en la lista desplegable la dirección de comienzo del sector a leer. Introduzca el nombre del fichero donde se van a guardar los datos almacenados en el sector y pulse *Leer*.

#### Lectura de un rango de direcciones

Seleccione la opción *Rango* e introduzca la dirección inicial y final del rango de direcciones que desea leer. Introduzca el nombre del fichero donde se van a guardar los datos y pulse *Leer*.

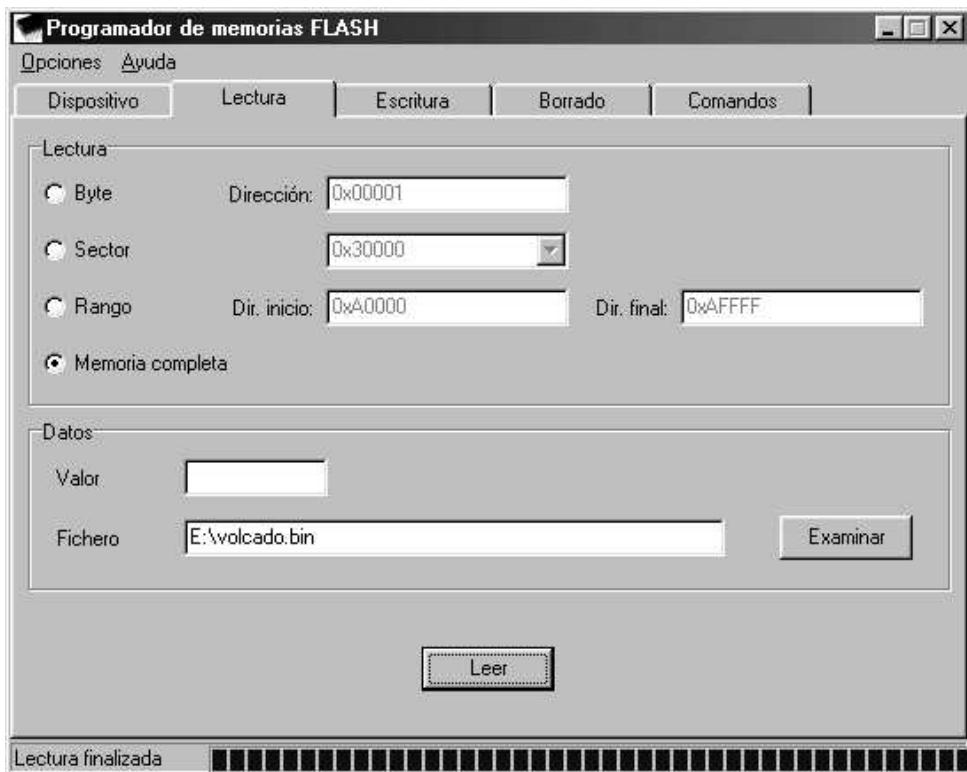


Figura A.5: Ventana de lectura de memorias

#### Lectura de la memoria completa

Seleccione la opción *Memoria*. Introduzca el nombre del fichero donde se van a guardar los datos almacenados en la memoria y pulse *Leer*.

#### A.4.4. Grabación de una memoria (fig. A.6)

Antes de grabar una memoria debe comprobar si se trata de una memoria programable por páginas o byte a byte (puede obtener esta información en la pestaña *Dispositivo*).

Las memorias programables byte a byte requieren que las direcciones que van a grabarse estén borradas (es decir, tengan un valor 0xFF<sup>1</sup>). Remítase a la sección A.4.5 en caso de necesitar borrarla. Tenga en cuenta que la unidad mínima que puede borrar es un sector completo, por lo que si sólo desea modificar una posición de memoria es posible que necesite leer el sector completo en el que se encuentra, modificar el valor en la imagen obtenida, borrar el sector y volver a grabarlo en la memoria.

En las memorias por páginas, durante la grabación, se realiza automáticamente un borrado de la/s página/s que contiene la dirección o direcciones a grabar, por lo que no es necesario

<sup>1</sup>Realmente sólo es necesario que: NOT(valor actual) AND valor a grabar == 0x00

un borrado previo. Para evitar la pérdida de datos del resto de la página al grabar un byte, nuestro software realiza internamente una lectura de la página, la modifica y la vuelve a grabar completamente, quedando por tanto las direcciones no modificadas intactas.

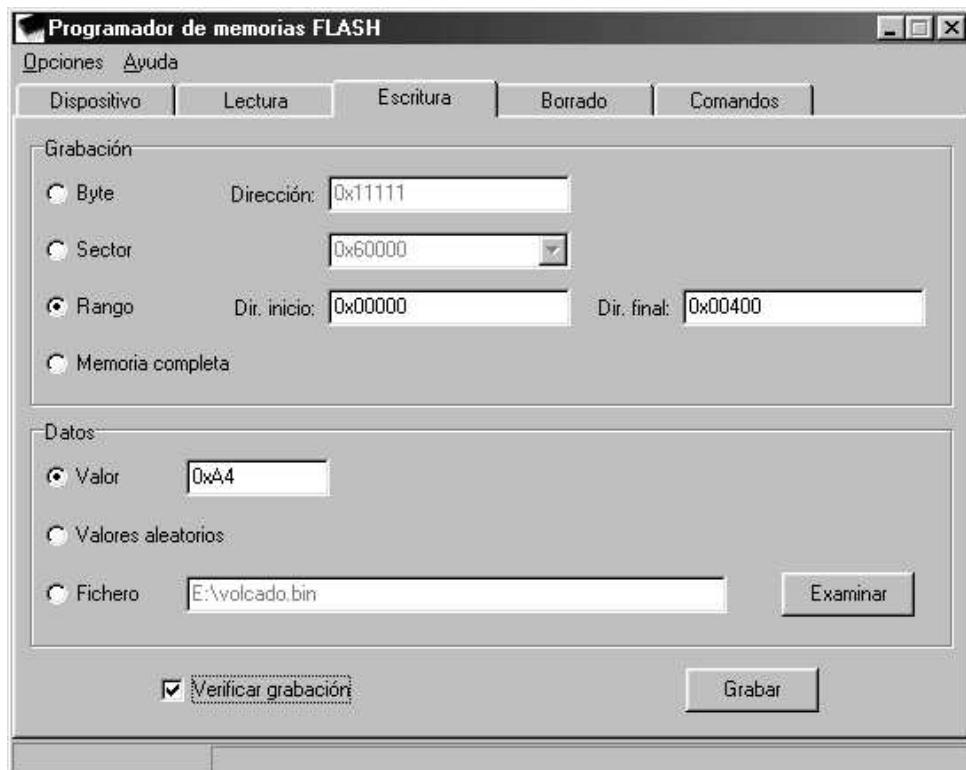


Figura A.6: Ventana de grabación de memorias

### Grabación de una dirección de memoria

Seleccione la opción *Byte* e introduzca en la casilla correspondiente la dirección a grabar. Seguidamente introduzca en la casilla *Valor* el dato a grabar. Si desea comprobar que la grabación se realiza correctamente marque la casilla *Verificar grabación*. A continuación pulse *Grabar*.

### Grabación de un sector

Seleccione la opción *Sector* y elija en la lista desplegable la dirección de comienzo del sector a grabar. Tiene tres opciones:

- Rellenar el sector con un determinado valor. Marque para ello *Valor* e introduzca el dato a grabar.

- Rellenar el sector con valores al azar. Marque *Valores aleatorios*.
- Guardar en el sector los datos contenidos en un fichero. Marque *Fichero* e introduzca el nombre.

Si desea comprobar que la grabación se realiza correctamente marque la casilla *Verificar grabación*. A continuación pulse *Grabar*.

### **Grabación de un rango de direcciones**

Seleccione la opción *Rango* e introduzca la dirección inicial y final del rango de direcciones que desea grabar. Tiene tres opciones:

- Rellenar el rango de direcciones con un determinado valor. Marque para ello *Valor* e introduzca el dato a grabar.
- Rellenar el rango de direcciones con valores al azar. Marque *Valores aleatorios*.
- Guardar en el rango de direcciones los datos contenidos en un fichero. Marque *Fichero* e introduzca el nombre.

Si desea comprobar que la grabación se realiza correctamente marque la casilla *Verificar grabación*. A continuación pulse *Grabar*.

### **Grabación de la memoria completa**

Seleccione la opción *Memoria*. Tiene tres opciones:

- Rellenar la memoria con un determinado valor. Marque para ello *Valor* e introduzca el dato a grabar.
- Rellenar la memoria con valores al azar. Marque *Valores aleatorios*.
- Guardar en la memoria los datos contenidos en un fichero. Marque *Fichero* e introduzca el nombre.

Si desea comprobar que la grabación se realiza correctamente marque la casilla *Verificar grabación*. A continuación pulse *Grabar*.

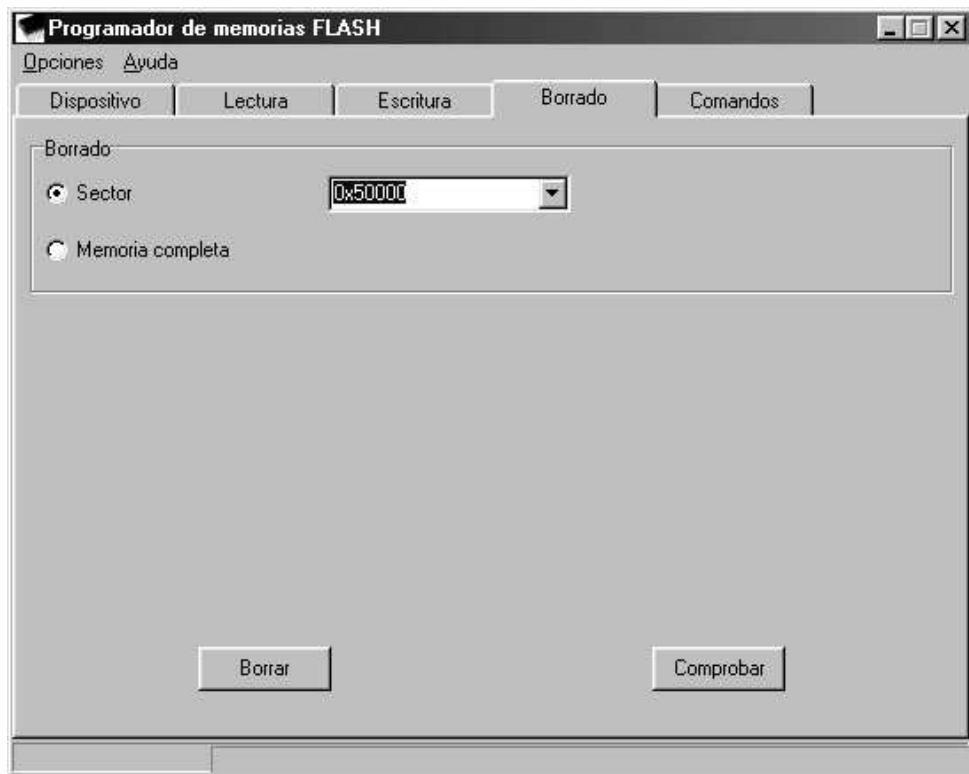


Figura A.7: Ventana de borrado de memorias

#### A.4.5. Borrado de una memoria (fig. A.7)

##### Comprobación y borrado de un sector

Seleccione la opción *Sector* y elija el sector en la lista desplegable. Pulse *Comprobar* para determinar si el sector está vacío (es decir, todas las direcciones están a *0xFF*) o *Borrar* para proceder a su borrado.

##### Comprobación y borrado de la memoria completa

Seleccione la opción *Memoria completa*. Pulse *Comprobar* para determinar si la memoria está vacía o *Borrar* para proceder a su borrado.

#### A.4.6. Ejecución de comandos (fig. A.8)

Si la memoria ofrece otros comandos aparte de los básicos para el borrado, grabación y lectura podrá elegir en la lista desplegable uno de estos comandos y ejecutarlo.

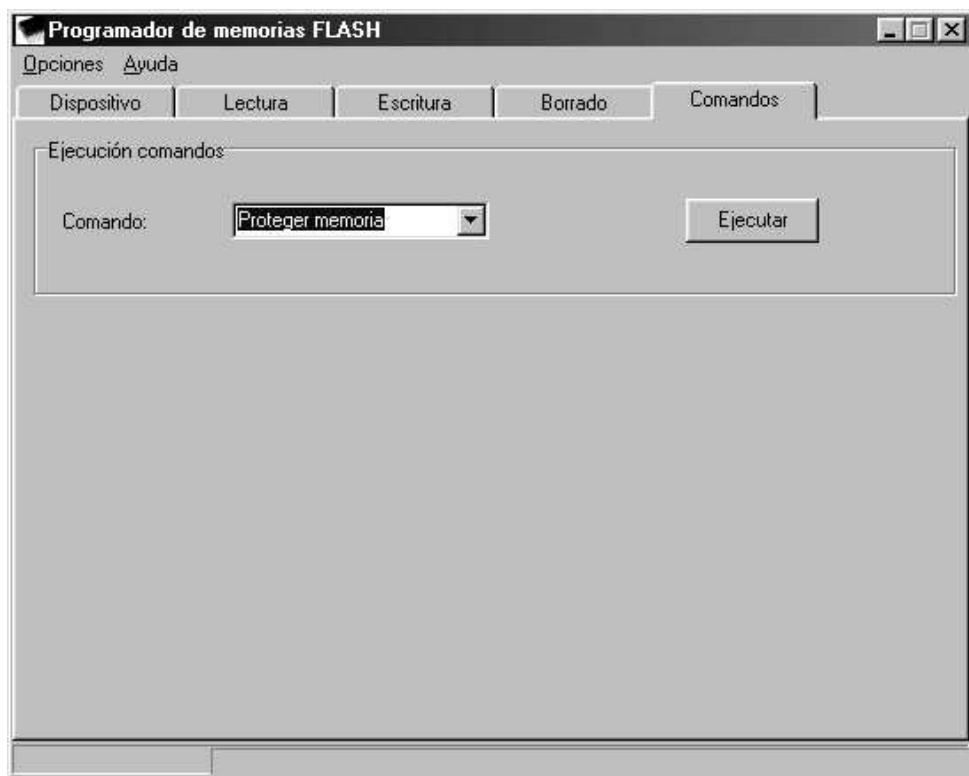


Figura A.8: Ventana de ejecución de comandos

## A.5. Gestión de la base de datos de memorias

Para acceder al gestor de la base de datos, pulse en el menú *Opciones* y elija la opción *Base de datos de memorias*.

### A.5.1. Gestión de fabricantes (fig. A.9)

Cada memoria estará asociada a un fabricante y su correspondiente código de identificación. En la ventana de la figura A.9 obtendrá una lista de los fabricantes presentes en la base de datos. Para añadir un nuevo fabricante pulse el botón *Nuevo*, aparecerá una nueva fila cuyos campos debe llenar. Para eliminar un fabricante selecciónelo y pulse *Eliminar*, a continuación confirme la operación. Tenga en cuenta que no es posible eliminar un fabricante que tenga memorias asociadas a él en la base de datos ni cambiar su código, por lo que antes tendrá que eliminar éstas.

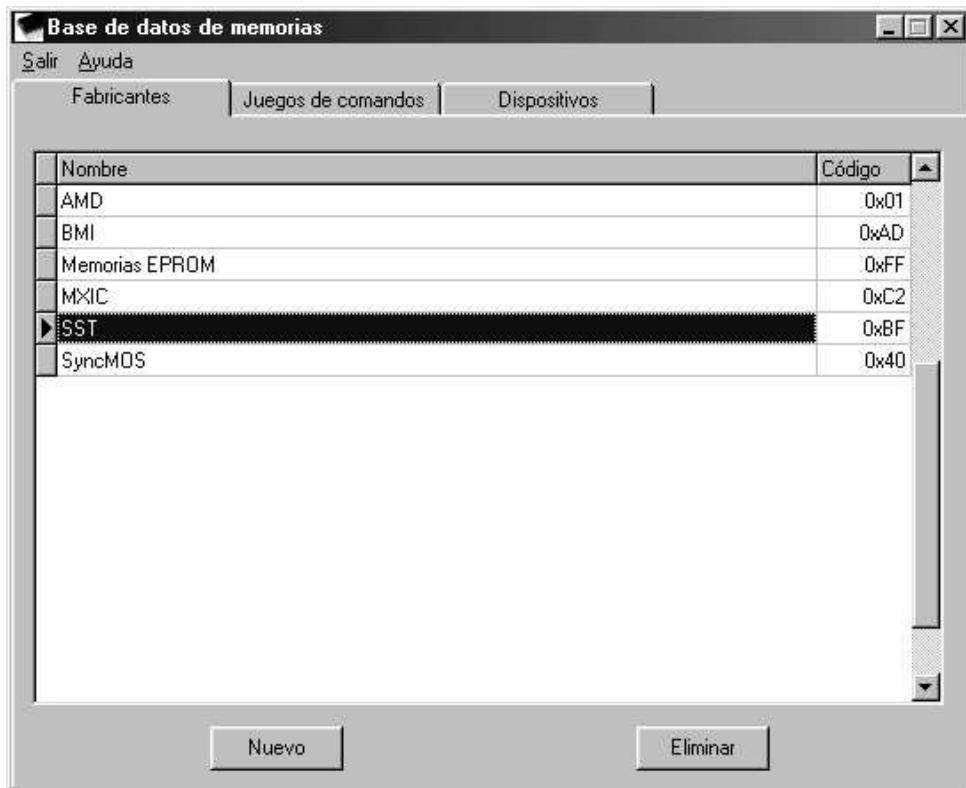


Figura A.9: Ventana de edición de fabricantes

### A.5.2. Gestión de juegos de comandos (fig. A.10)

La mayor parte de las memorias flash que se encuentran en el mercado usará uno de los dos juegos de comandos que vienen en la base de datos original de este software, etiquetados como “Comandos AMD” o “Comandos SST”, pero en caso necesario puede crear nuevos juegos de comandos o modificar los ya existentes.

La parte superior de la pantalla (figura A.10) nos permite crear, copiar y eliminar juegos de comandos, y la inferior crear, copiar, eliminar o modificar comandos concretos del juego seleccionado en la parte superior.

Cada juego de comandos consta de una serie de comandos básicos, necesarios para el funcionamiento de la aplicación (y que, por tanto, no se pueden eliminar) como son los de lectura (*Read*), programación (*Program*), borrado (*Sector erase* y *Chip Erase*), etc., y una serie de comandos extras y opcionales, que son los que se podrán ejecutar en la pestaña *Comandos* de la ventana principal del software. Cada uno de estos comandos se compone de una secuencia de pares dirección-dato. Las direcciones son valores hexadecimales (aunque también puede introducirlos en formato decimal y el programa hará automáticamente la conversión a hexadecimal) comprendidos entre 0x00000 y 0x7FFFF y los datos valores entre 0x00 y 0xFF. En el

## A.5. GESTIÓN DE LA BASE DE DATOS DE MEMORIAS

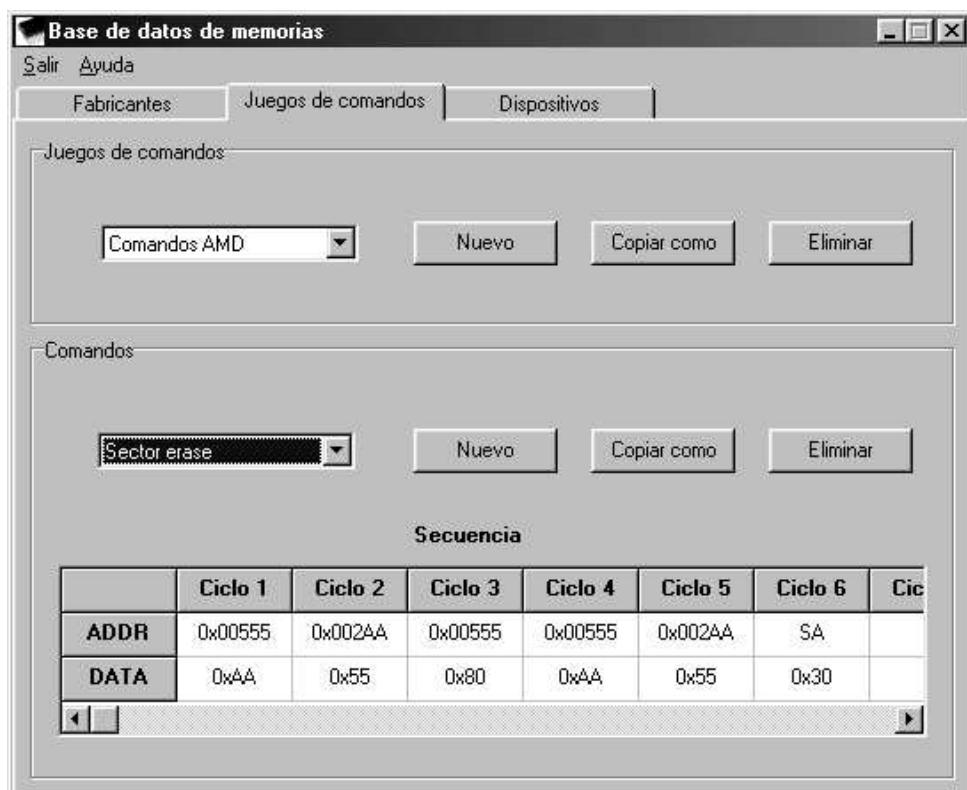


Figura A.10: Ventana de edición de juego de comandos

comando de borrado de sectores (*Sector erase*) se permite además el uso del parámetro *SA*, que será sustituido por la dirección del sector a borrar durante el envío de la secuencia a la memoria. Los cambios realizados serán guardados en la base de datos inmediatamente, sin pedirse confirmación.

### A.5.3. Gestión de dispositivos (fig. A.11)

La ventana de la figura A.11 nos permite añadir, copiar, modificar y eliminar las memorias de la base de datos.

De cada memoria se requieren los siguientes datos, que se pueden obtener de las especificaciones (*datasheet*) publicadas por el fabricante:

- Nombre y código del fabricante.
- Código identificativo del modelo.
- Nombre del modelo.
- Tamaño total de la memoria.

## A.5. GESTIÓN DE LA BASE DE DATOS DE MEMORIAS

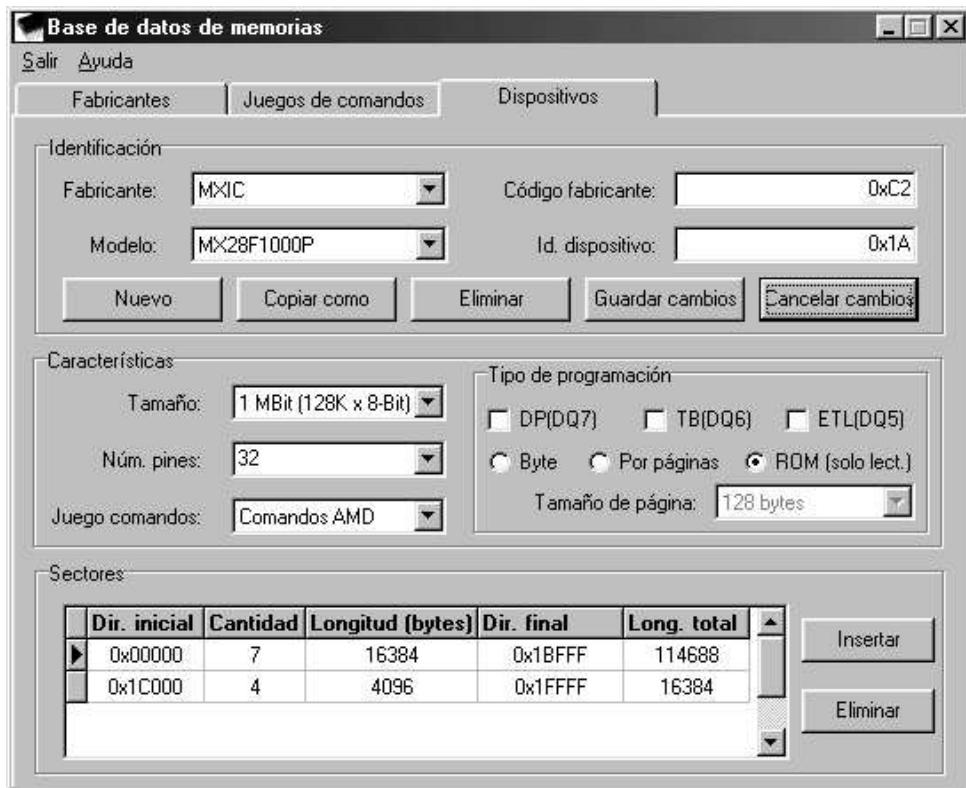


Figura A.11: Ventana de edición de memorias

- Número de pines.
- Juego de comandos que utiliza.
- Tipo de programación: programable byte a byte o por páginas o, si no es programable a 5V, *ROM*. Así como los mecanismos de señalización de estado de la grabación que soporta: *data polling (DP-DQ7)*, *toggle bit (TB-DQ6)* y/o *exceeded timing limit (ETL-DQ5)*.
- Tamaño de página, si es aplicable.
- Lista de sectores, si es que está organizada de tal forma. Para añadir un sector sitúese sobre la tabla y pulse *Insertar* o desplácese hasta la última fila y pulse el cursor-abajo; aparecerá una nueva entrada con los campos vacíos para que los rellene. Si la memoria tiene sectores consecutivos del mismo tamaño se recomienda indicar el tamaño, la dirección del primer sector y el número de sectores consecutivos, en vez de crear una entrada por cada sector. Para eliminar una fila sitúese sobre ella y pulse *Eliminar*, a continuación confirme la operación.

#### *A.5. GESTIÓN DE LA BASE DE DATOS DE MEMORIAS*

---

Los cambios realizados en esta pantalla no se harán efectivos hasta que pulse en *Guardar cambios* o conteste afirmativamente a la pregunta que se le hace al abandonar la pestaña o el gestor de la base de datos.

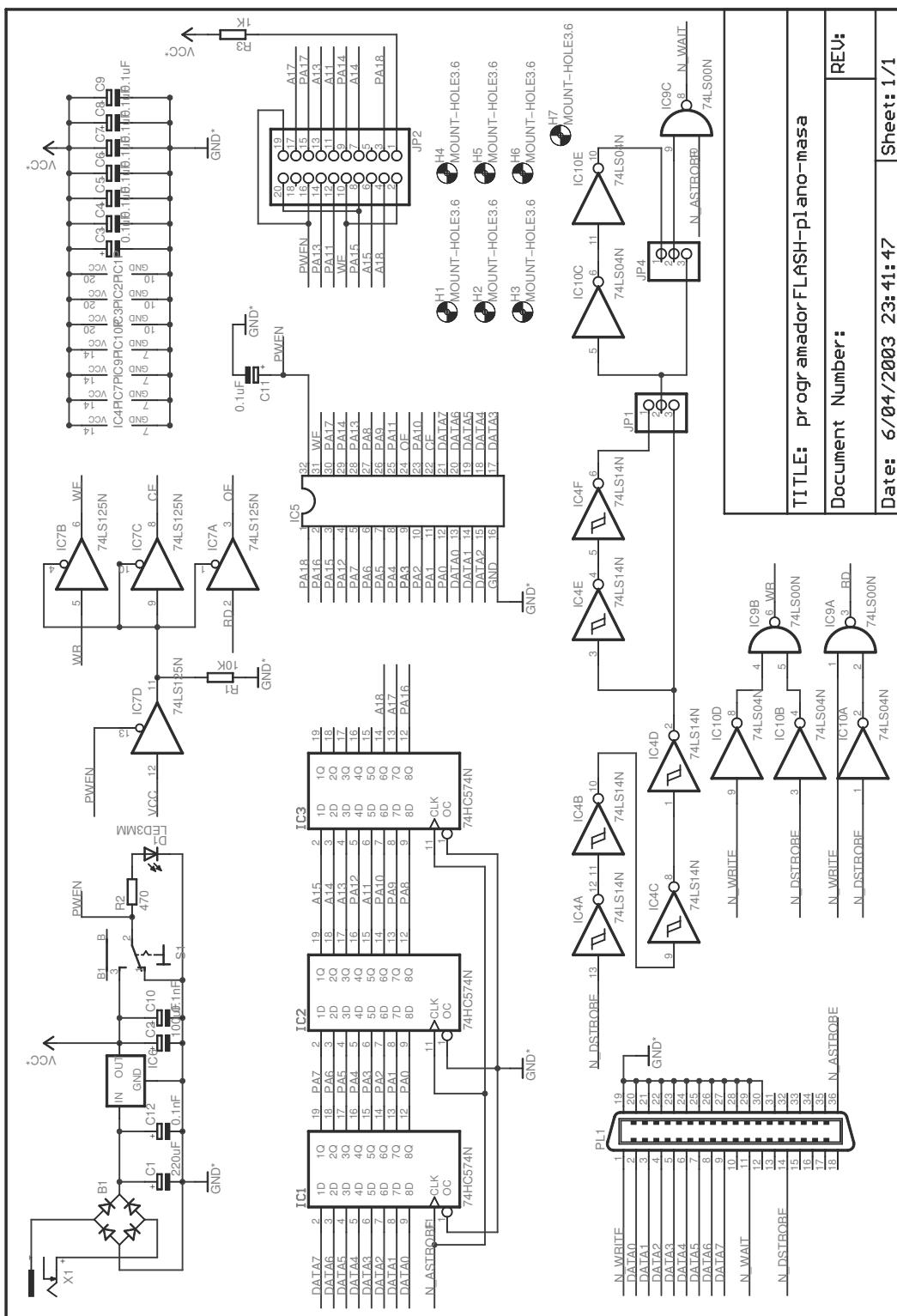
---

*A.5. GESTIÓN DE LA BASE DE DATOS DE MEMORIAS*

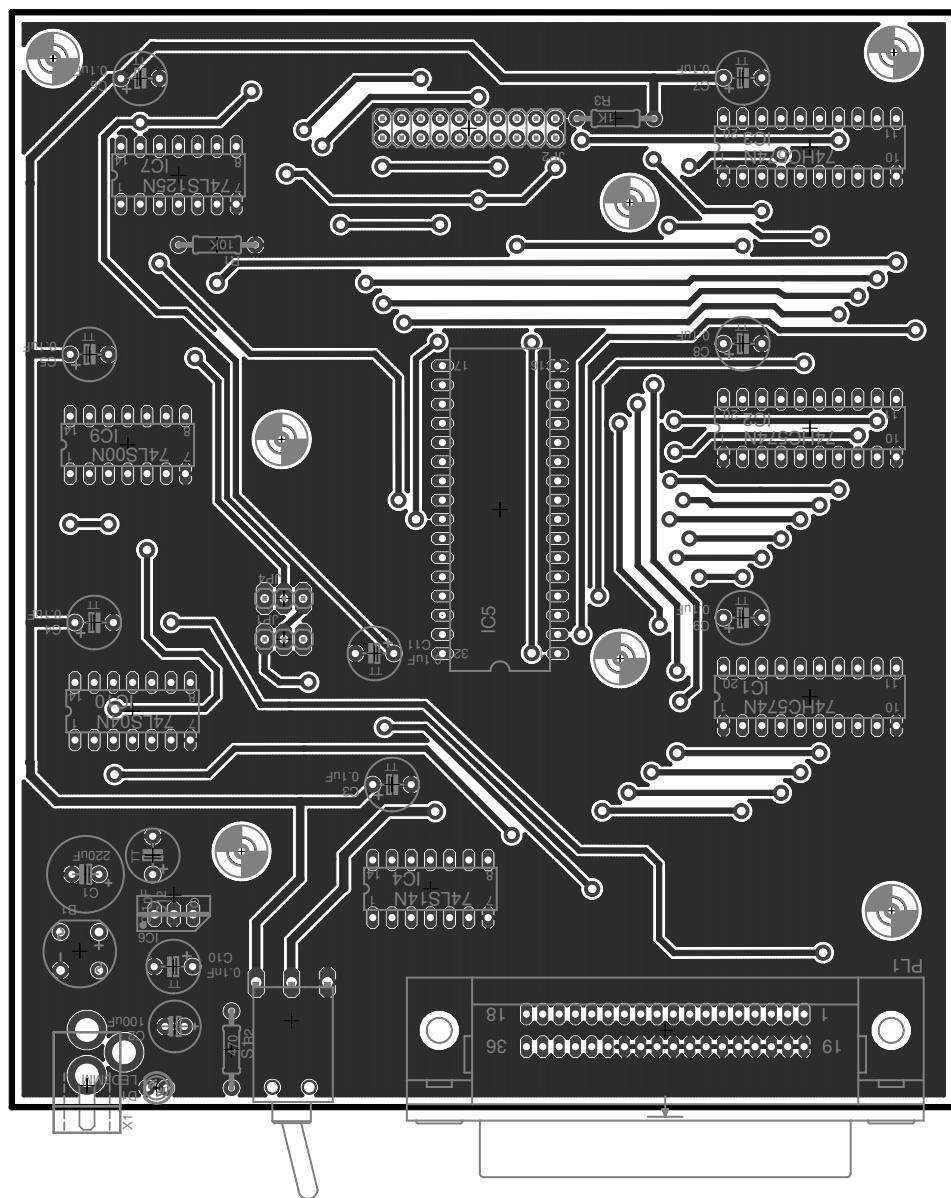
## **Capítulo B**

### **Diagramas del hardware**

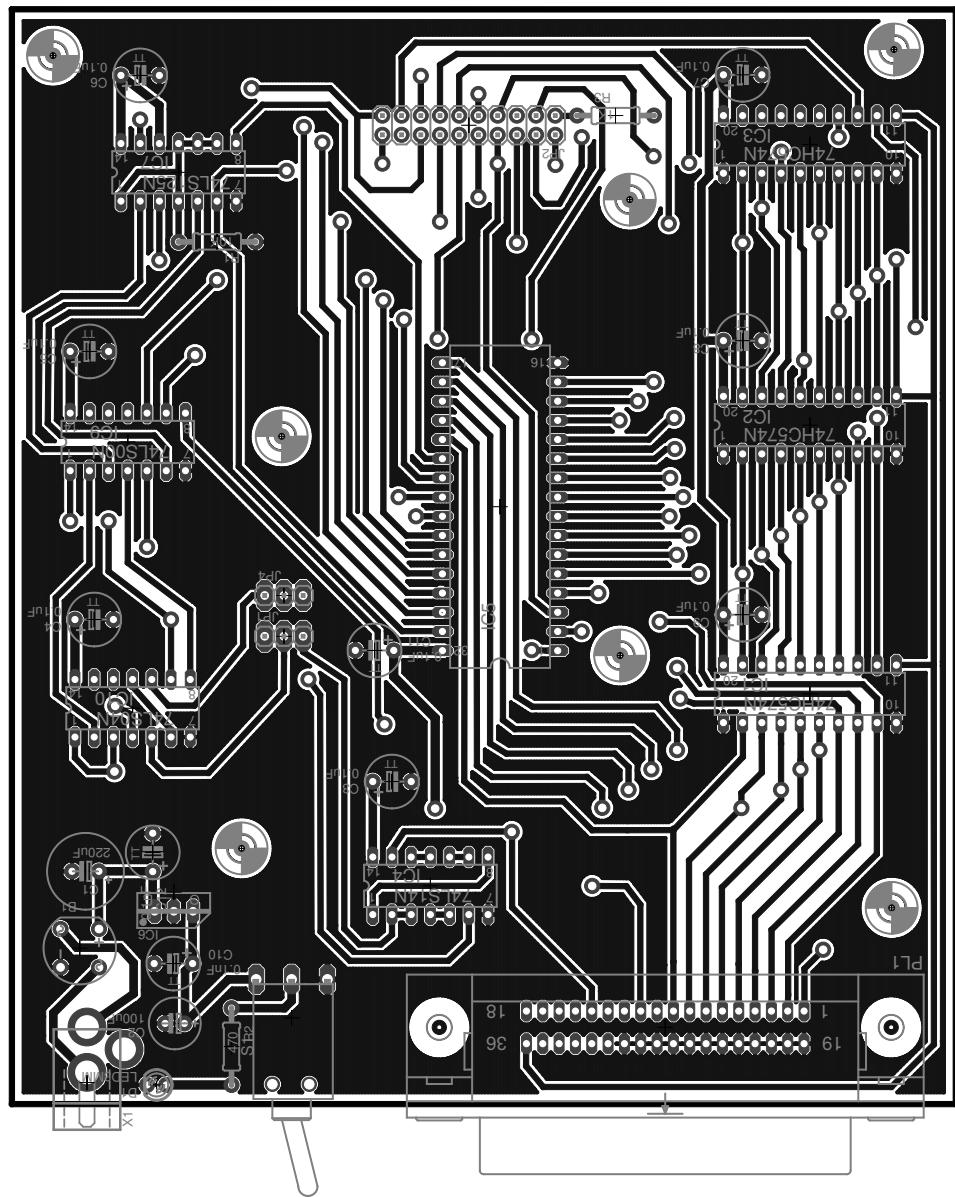
### **B.1. Diseño esquemático final**



## B.2. Trazado final placa PCB. Cara superior



### **B.3. Trazado final placa PCB. Cara inferior**

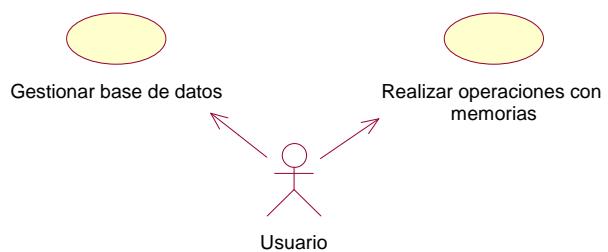


# **Capítulo C**

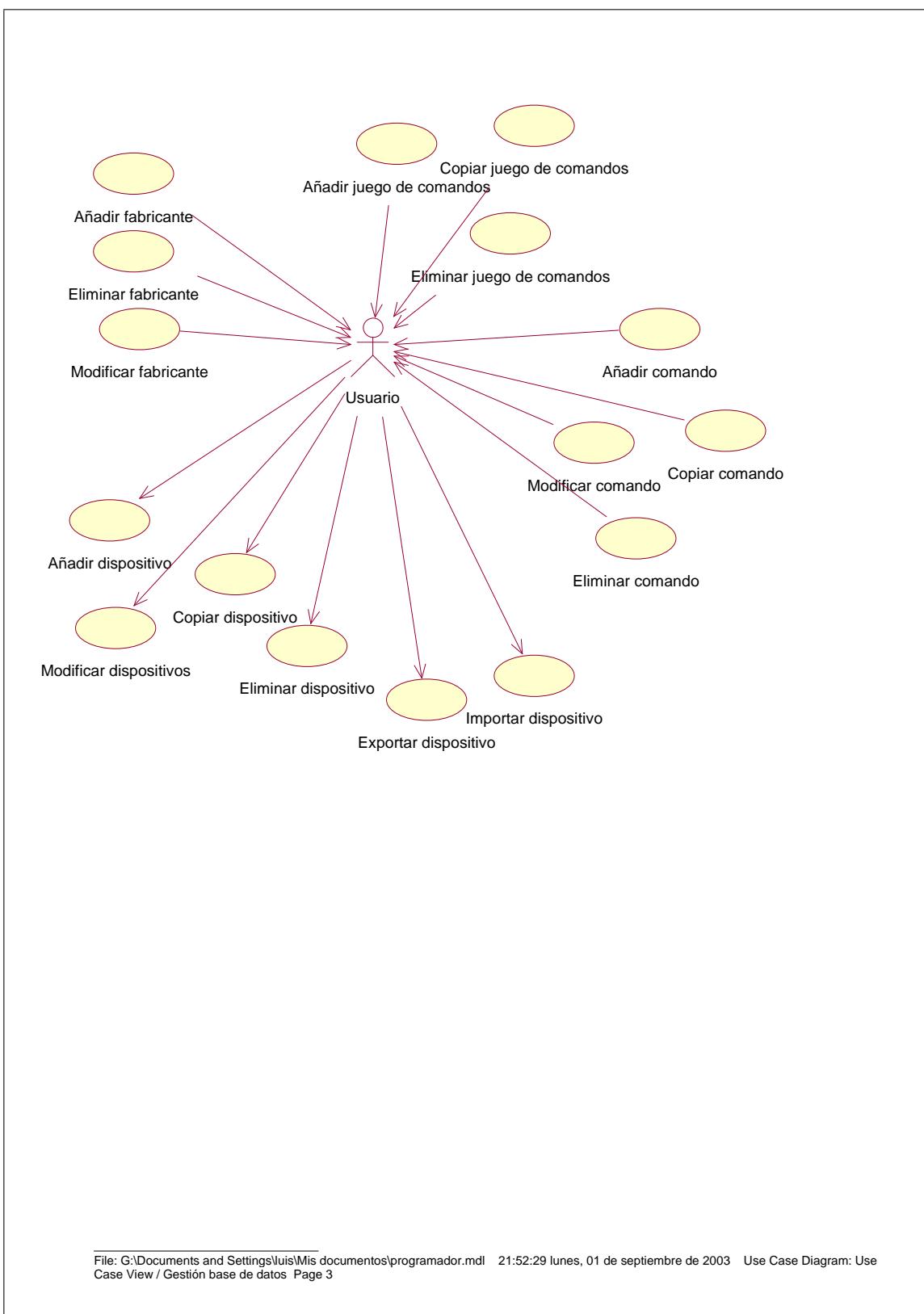
## **Diagramas de análisis del software**

### **C.1. Diagramas de casos de uso**

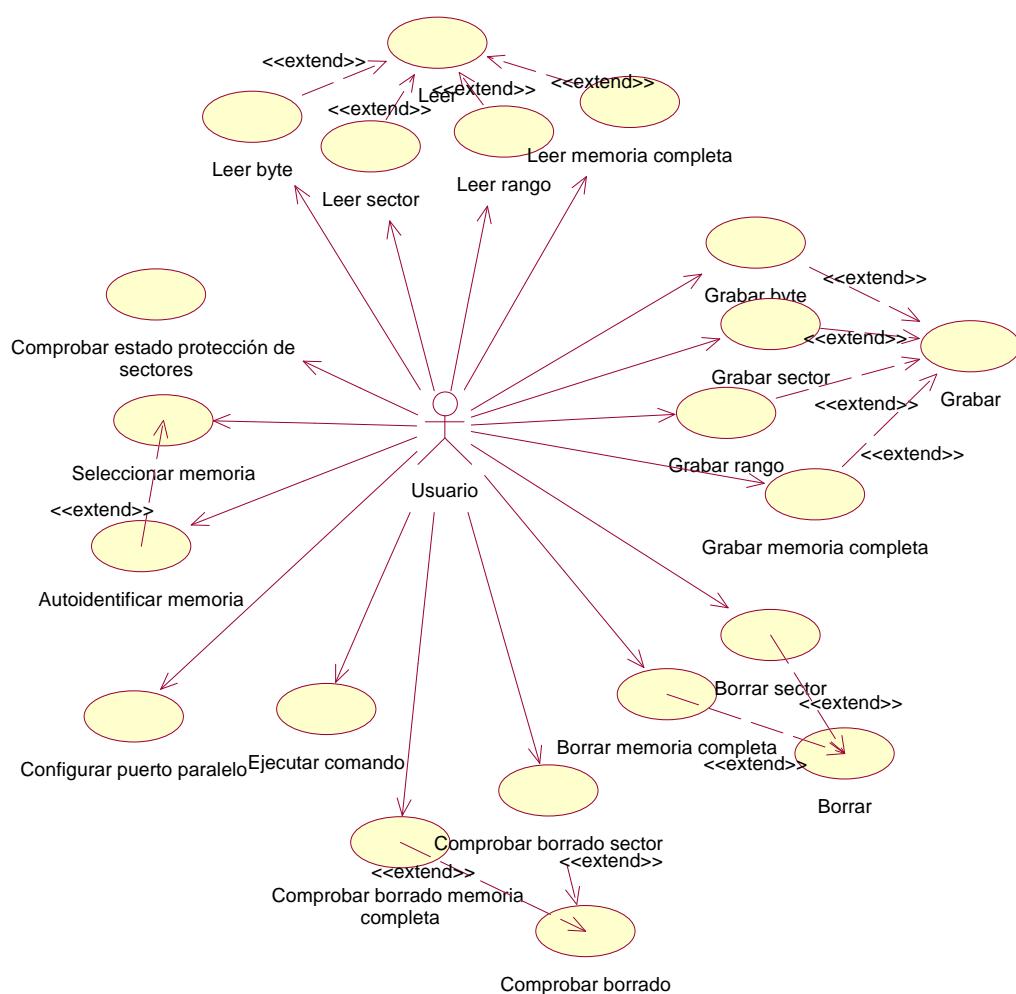
### C.1. DIAGRAMAS DE CASOS DE USO



File: G:\Documents and Settings\luis\Mis documentos\programador.mdl 21:52:29 lunes, 01 de septiembre de 2003 Use Case Diagram: Use Case View / Main Page 2



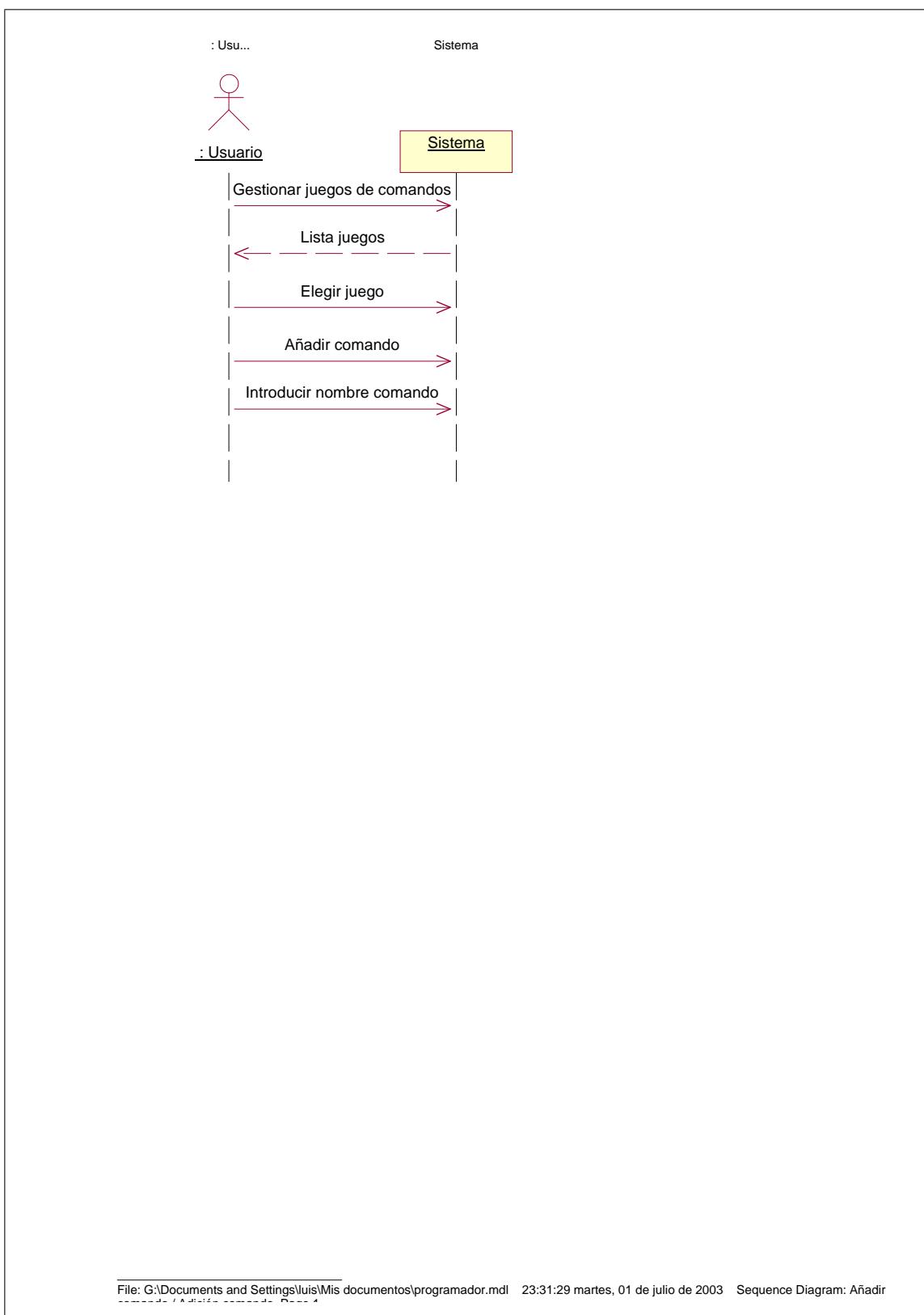
File: G:\Documents and Settings\luis\Mis documentos\programador.mdl 21:52:29 lunes, 01 de septiembre de 2003 Use Case Diagram: Use Case View / Gestión base de datos Page 3



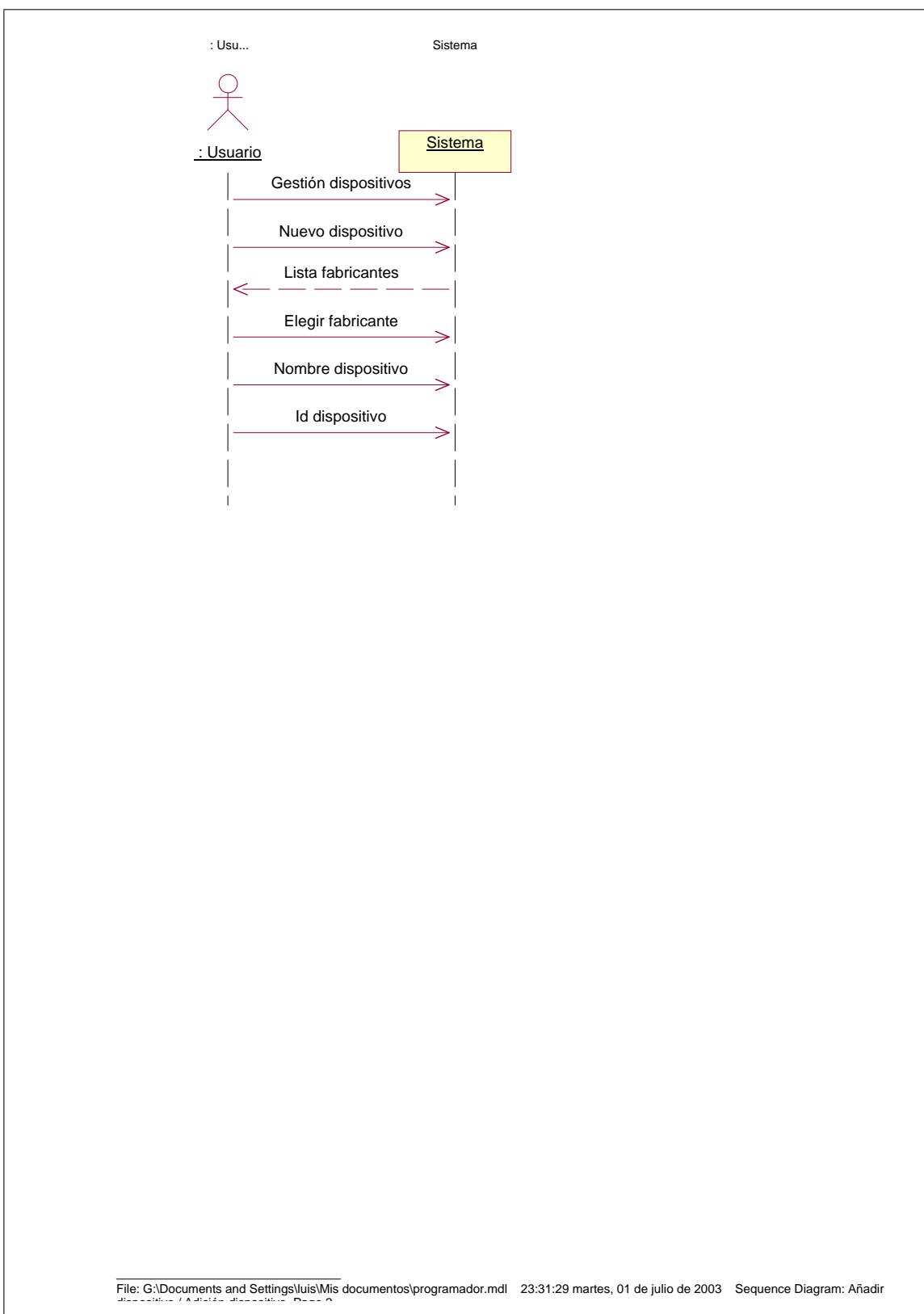
File: G:\Documents and Settings\luis\Mis documentos\programador.mdl 21:52:29 lunes, 01 de septiembre de 2003 Use Case Diagram: Use Case View / Operaciones con memorias Page 1

## **C.2. Diagramas de secuencia de casos de uso**

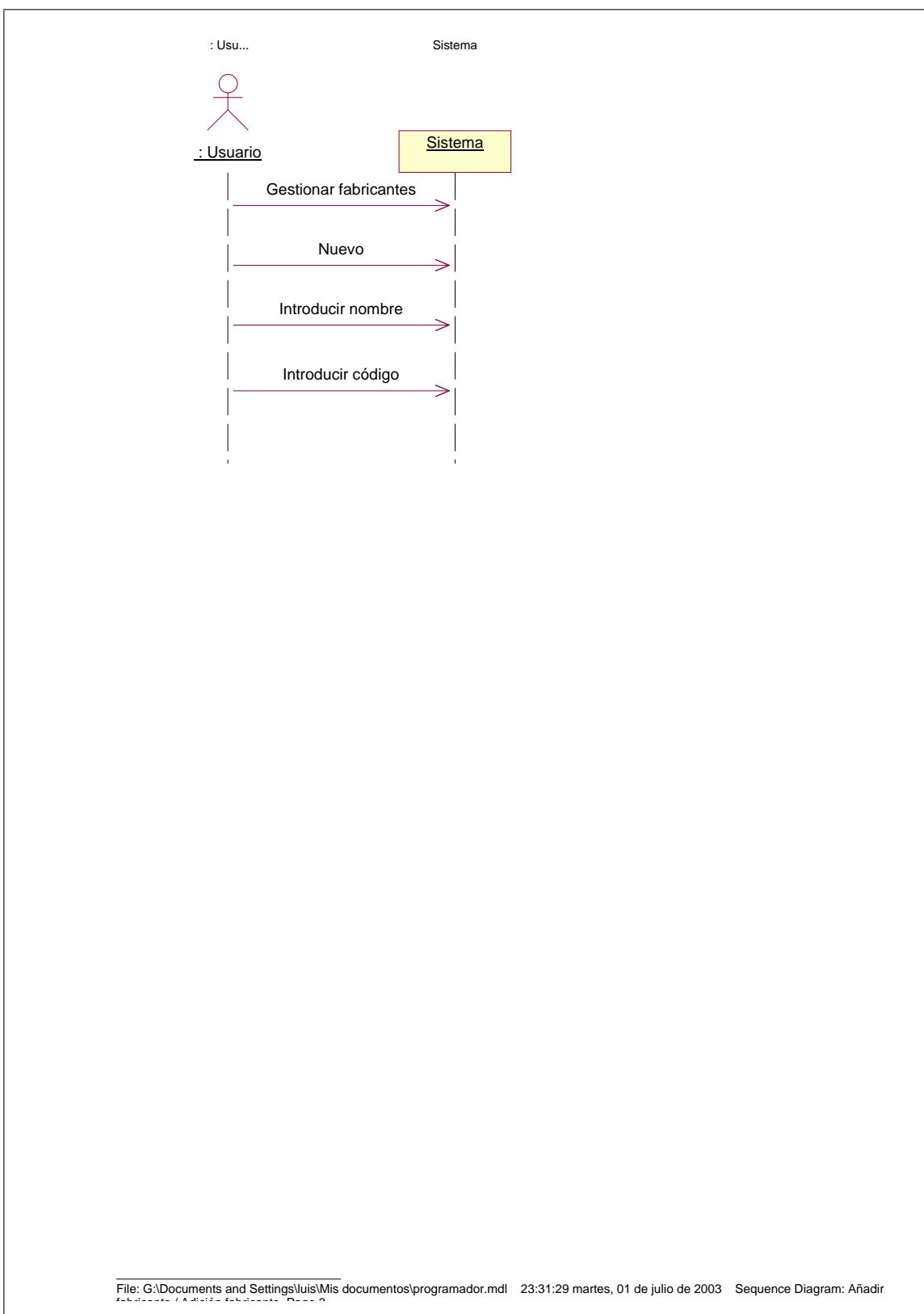
## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO



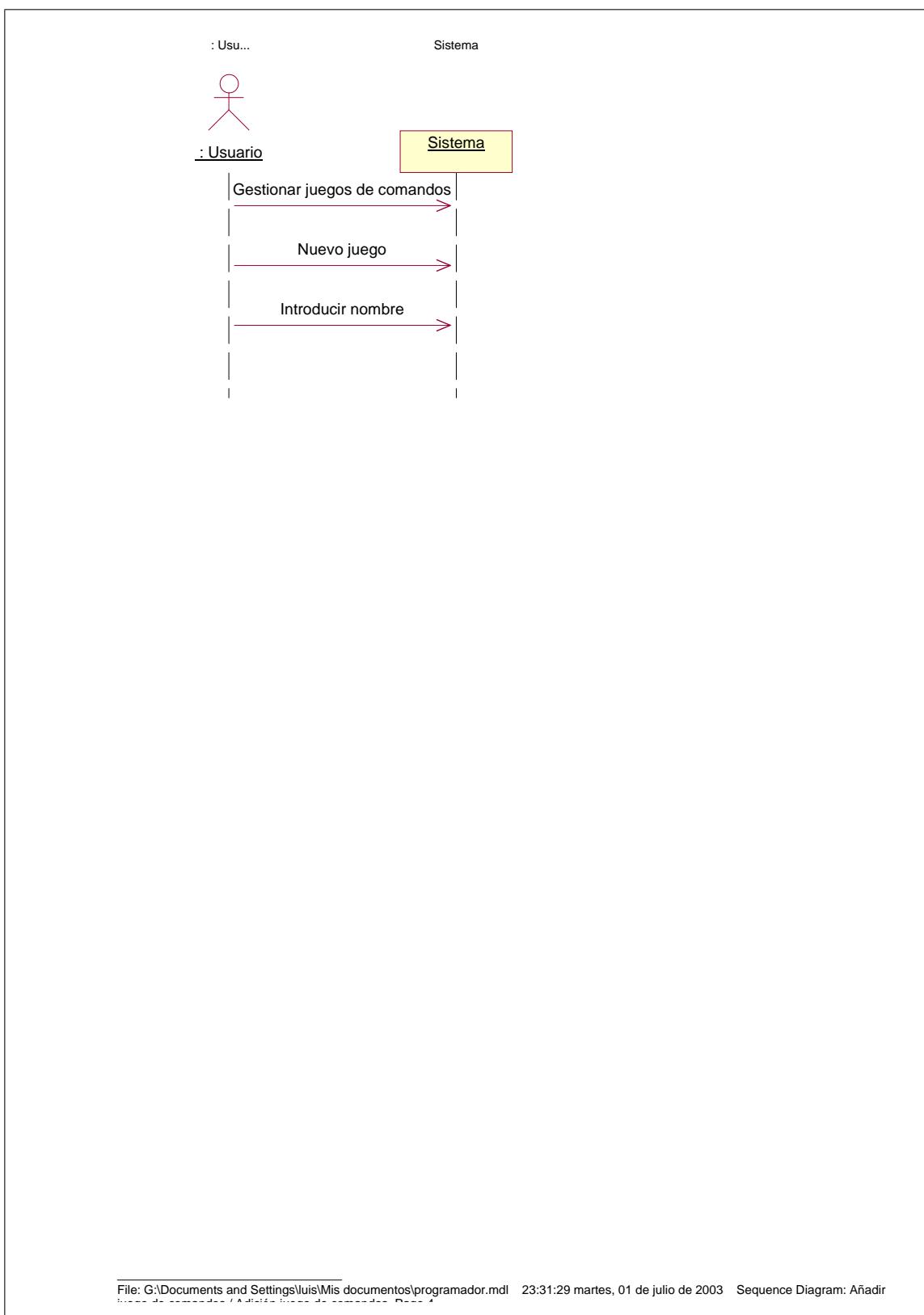
## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO



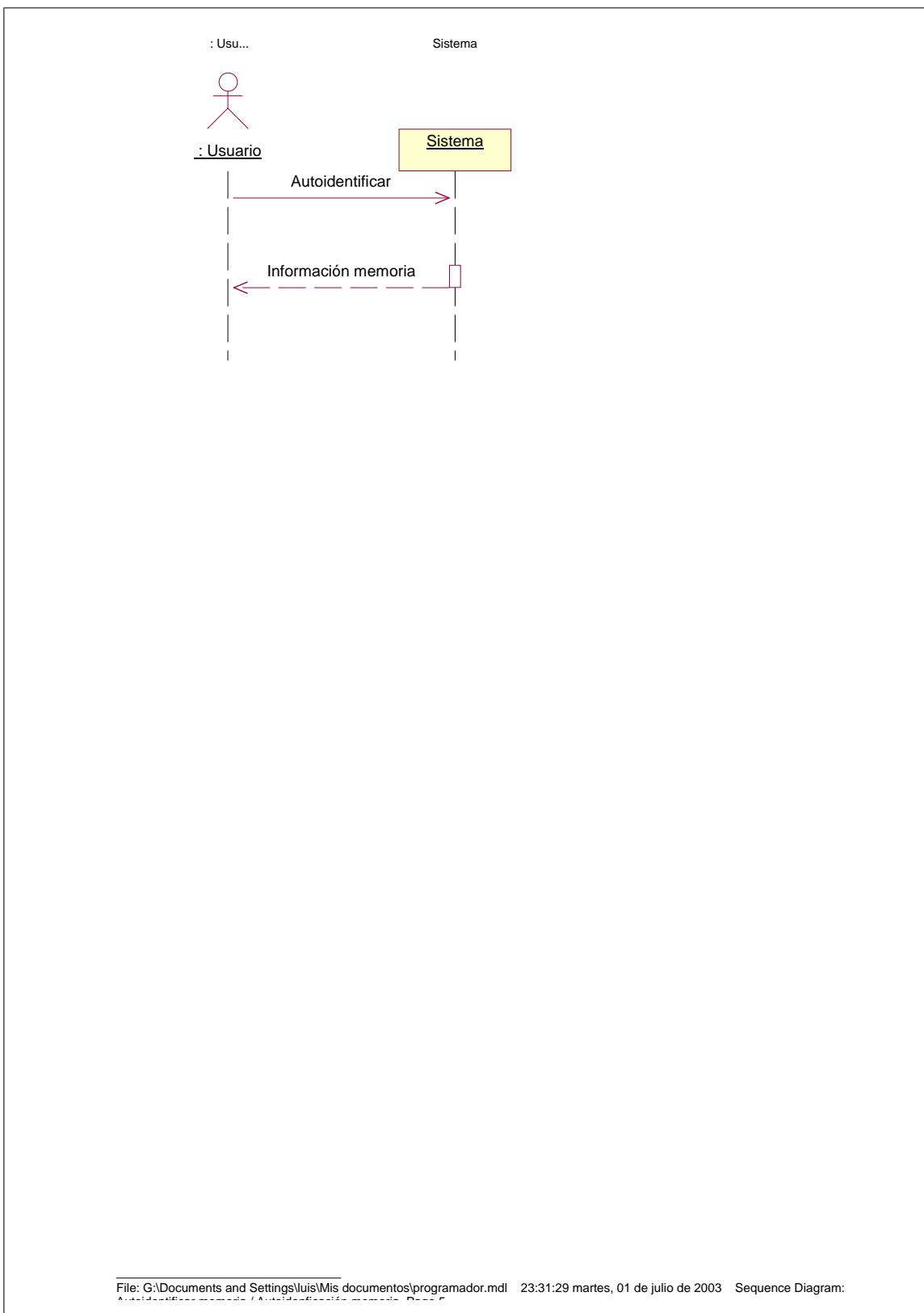
## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO



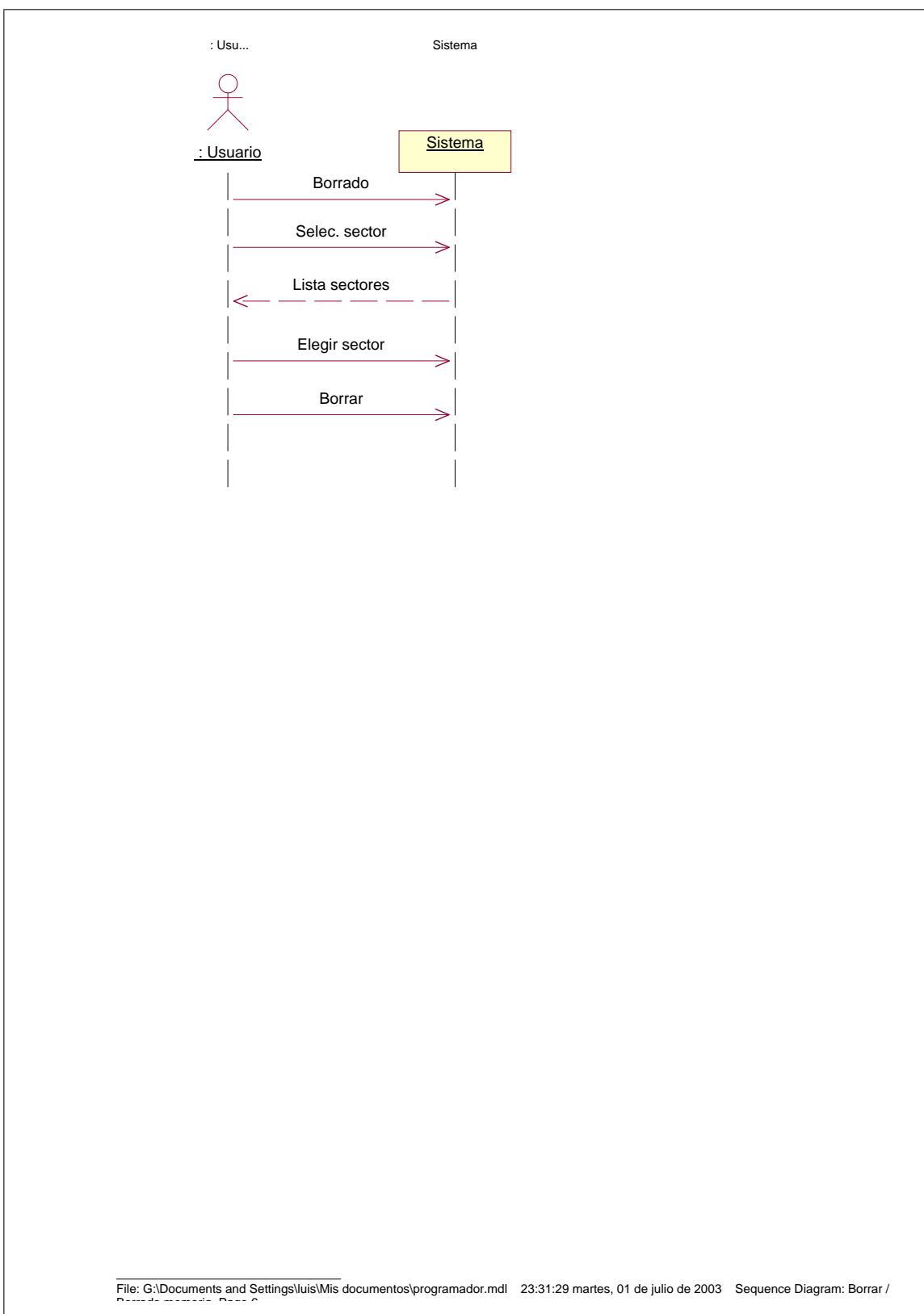
## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO



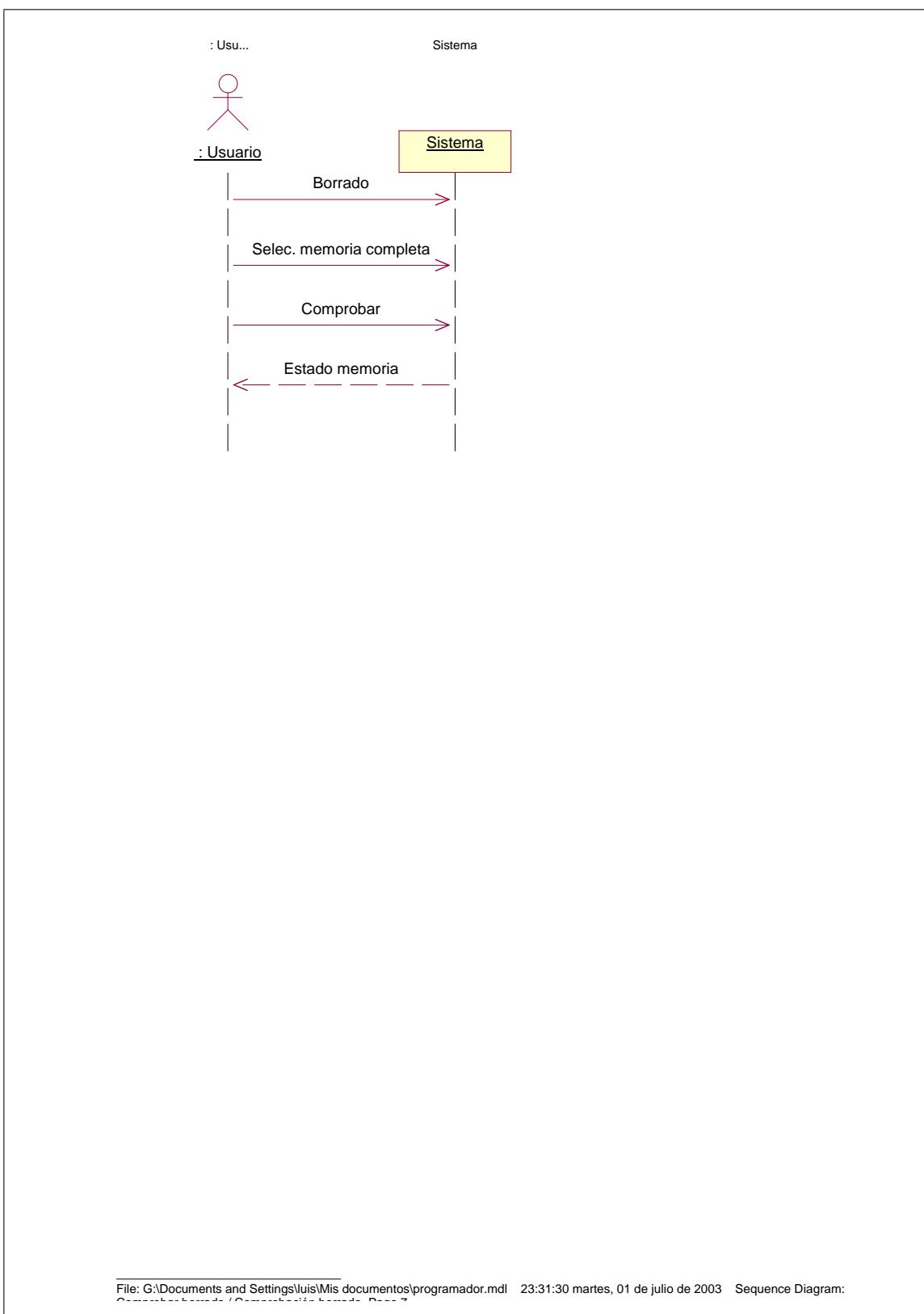
## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO



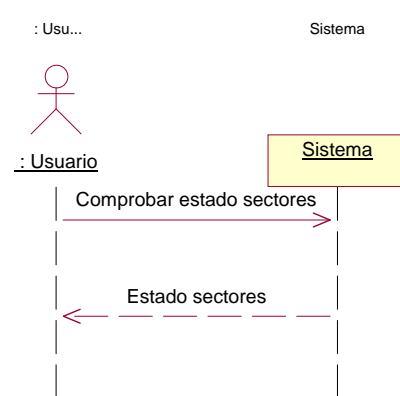
## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO



## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO

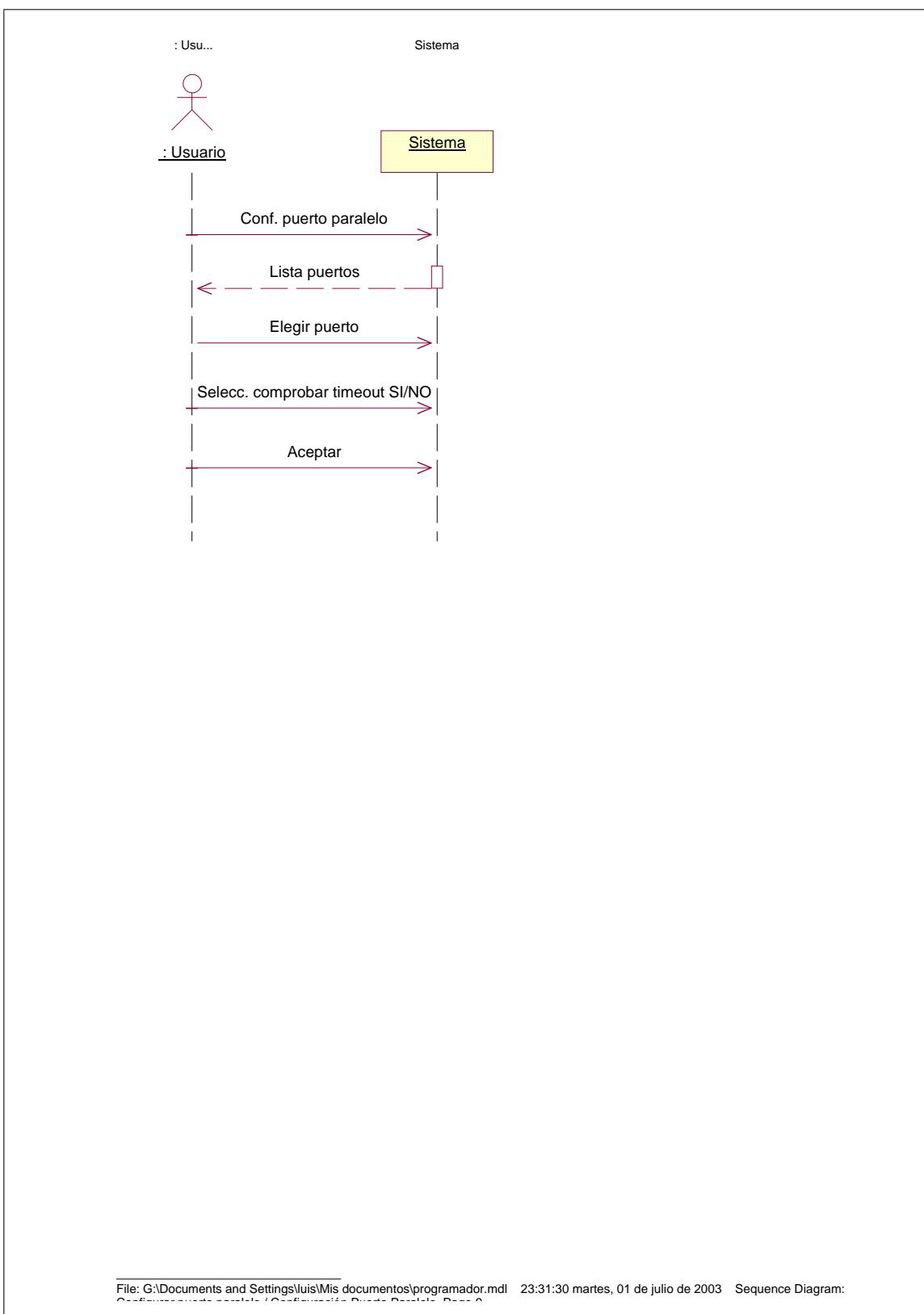


## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO

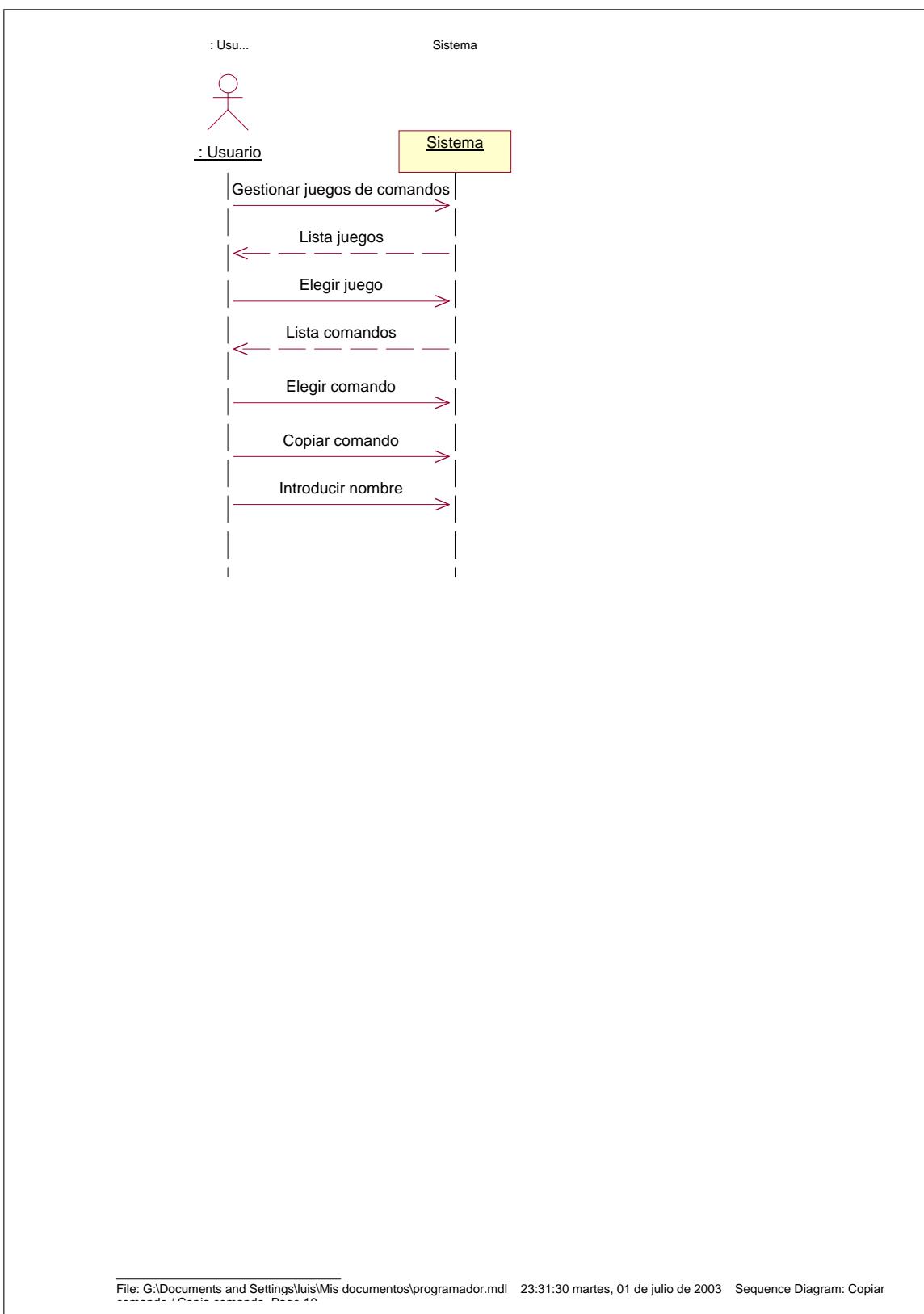


File: G:\Documents and Settings\luis\Mis documentos\programador.mdl 23:31:30 martes, 01 de julio de 2003 Sequence Diagram:

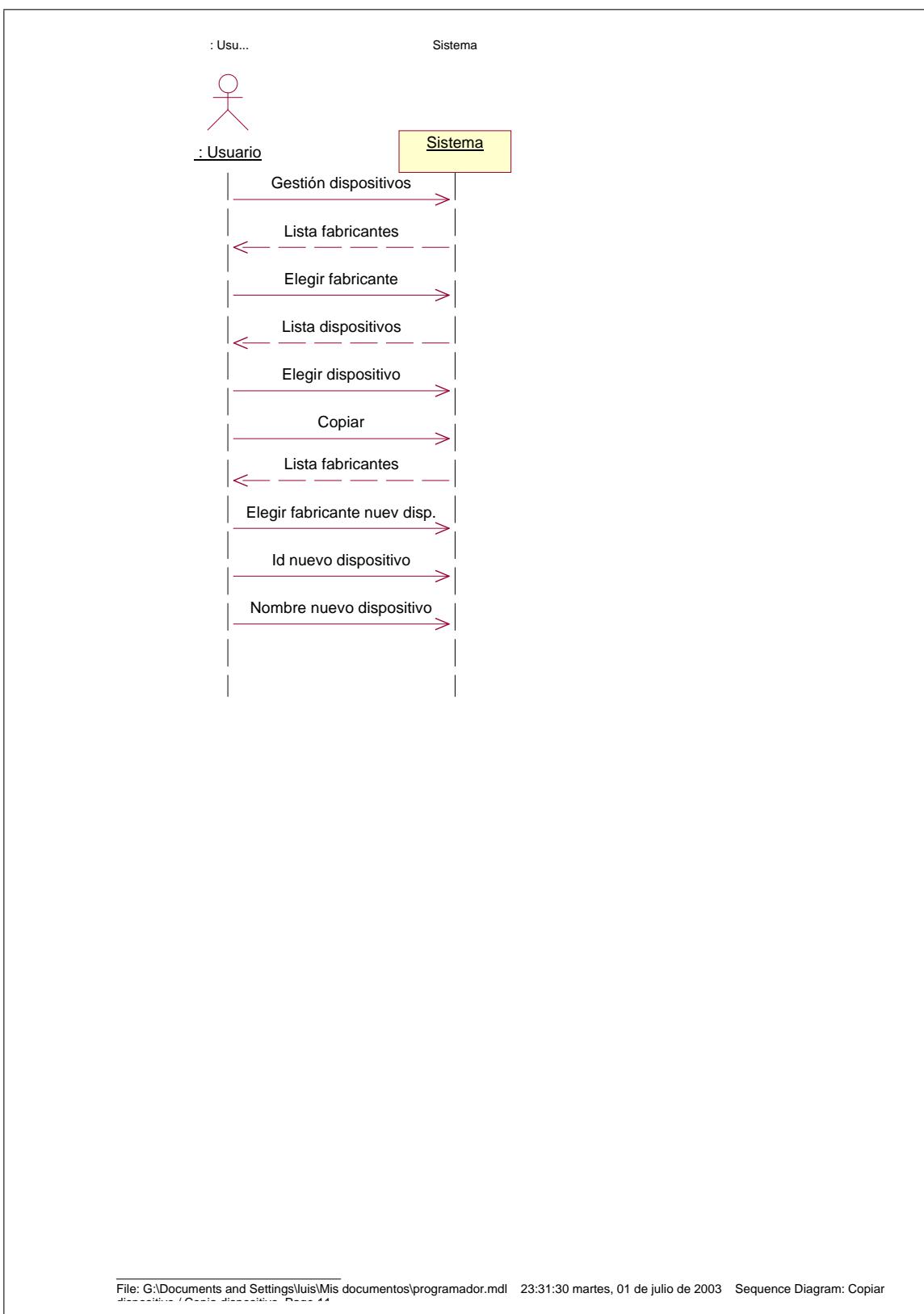
## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO



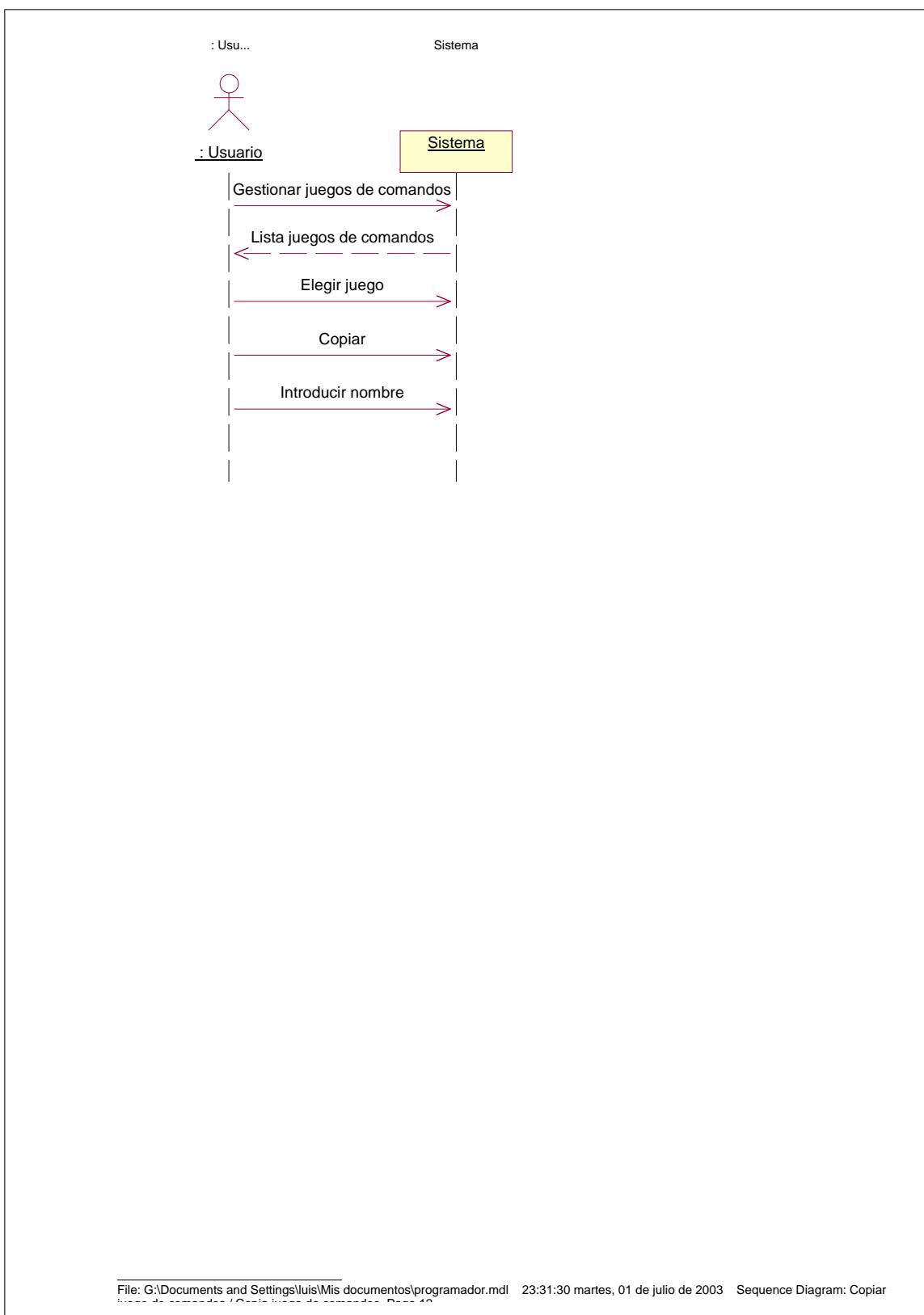
## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO



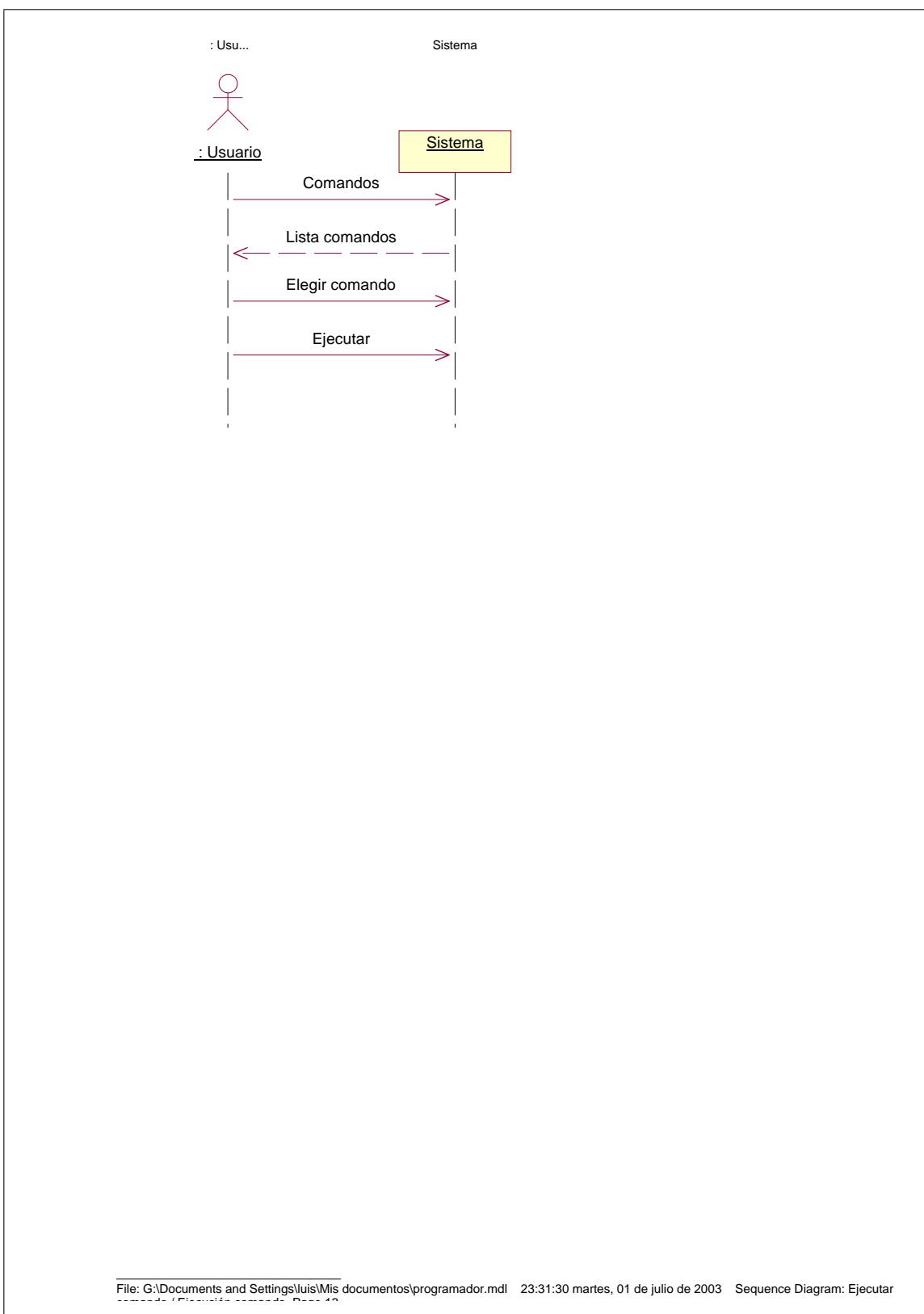
## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO



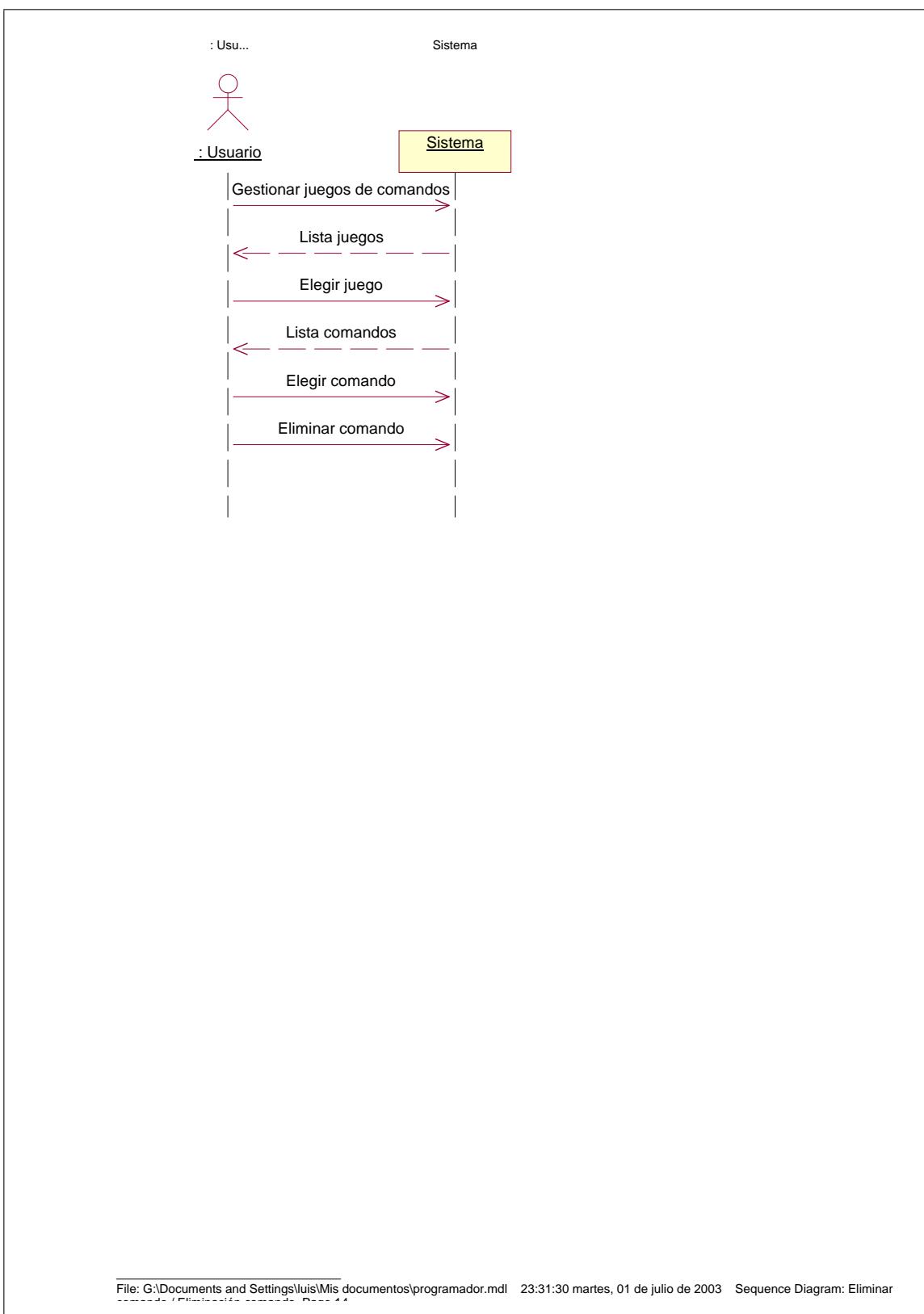
## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO



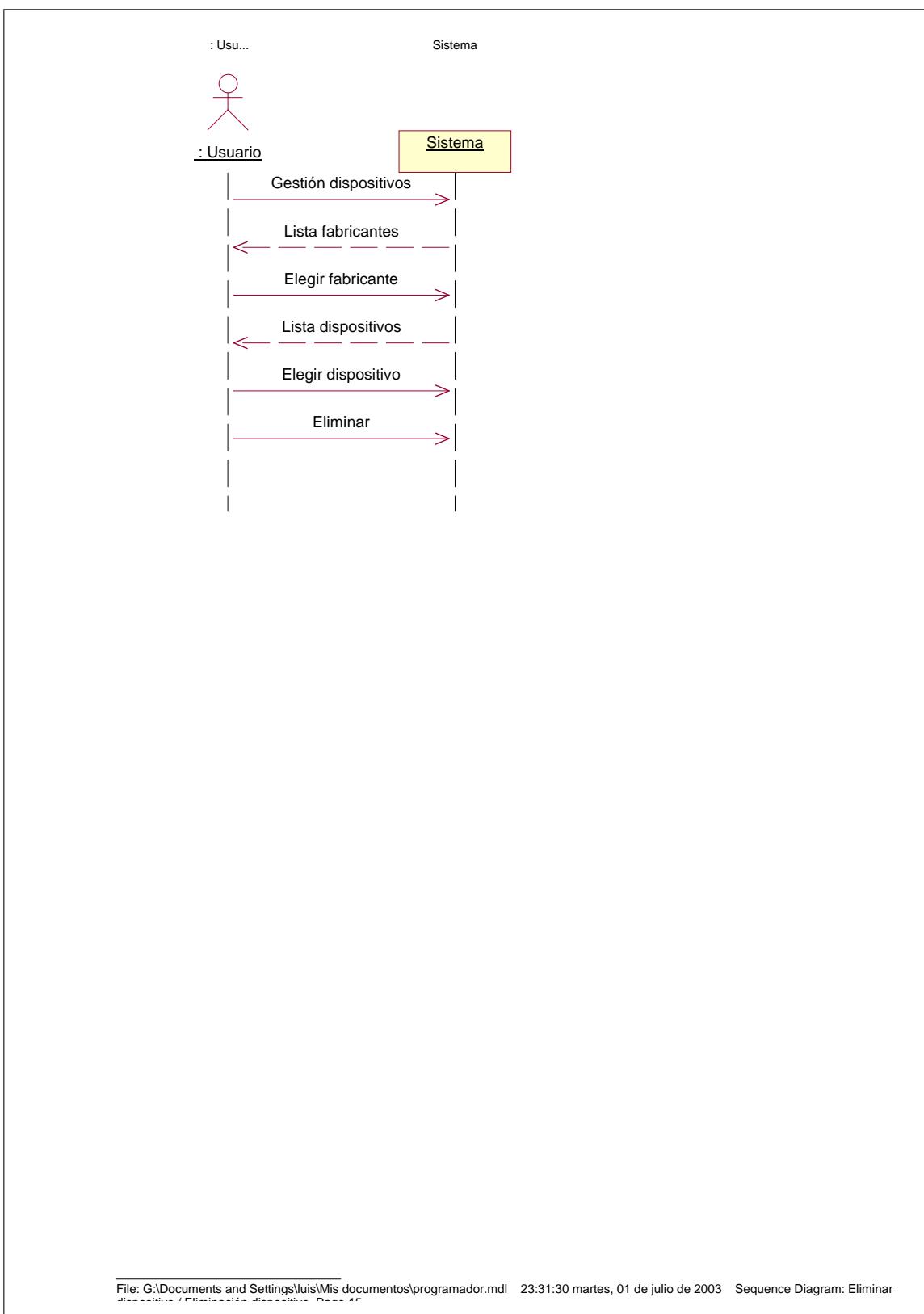
## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO



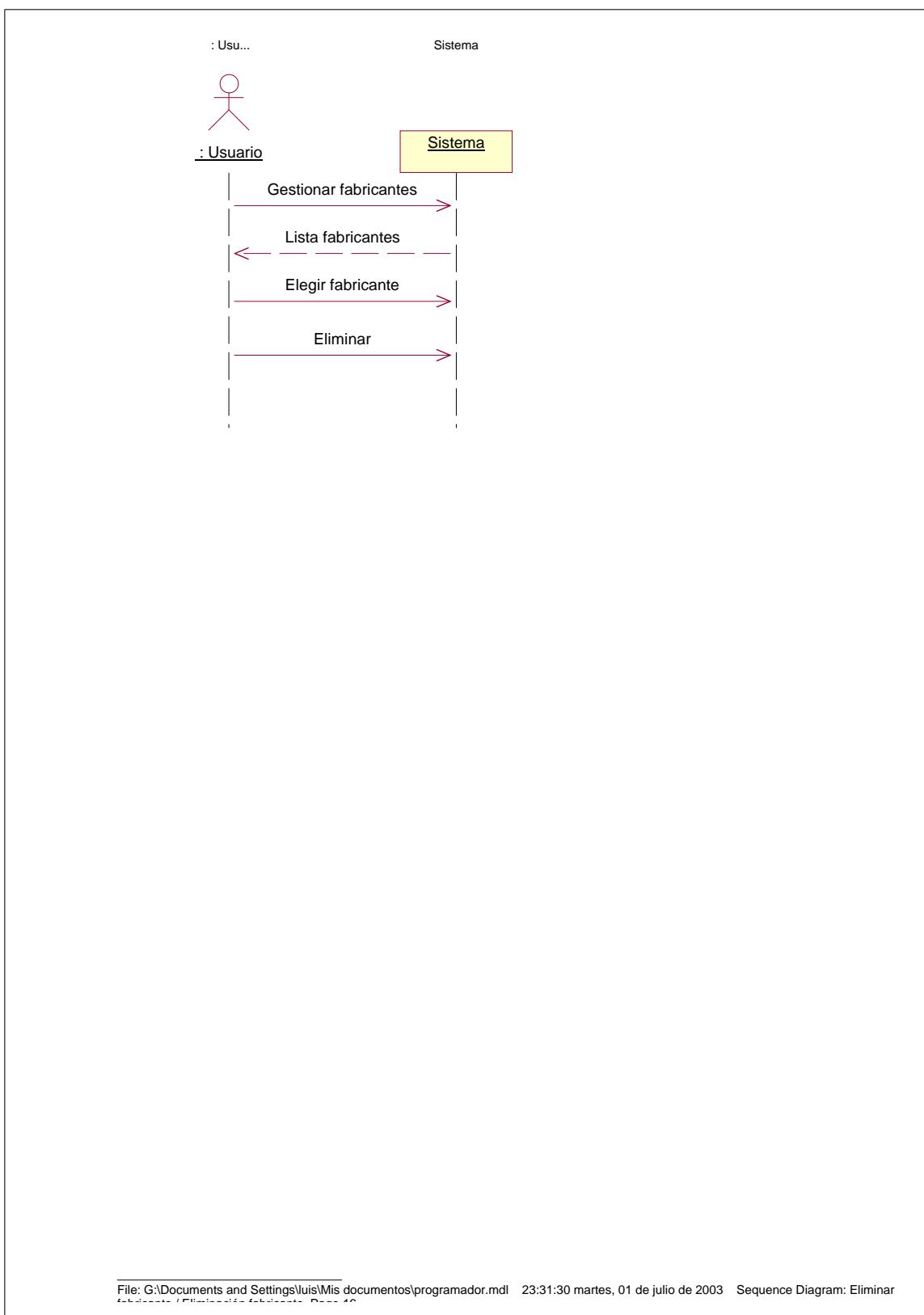
## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO



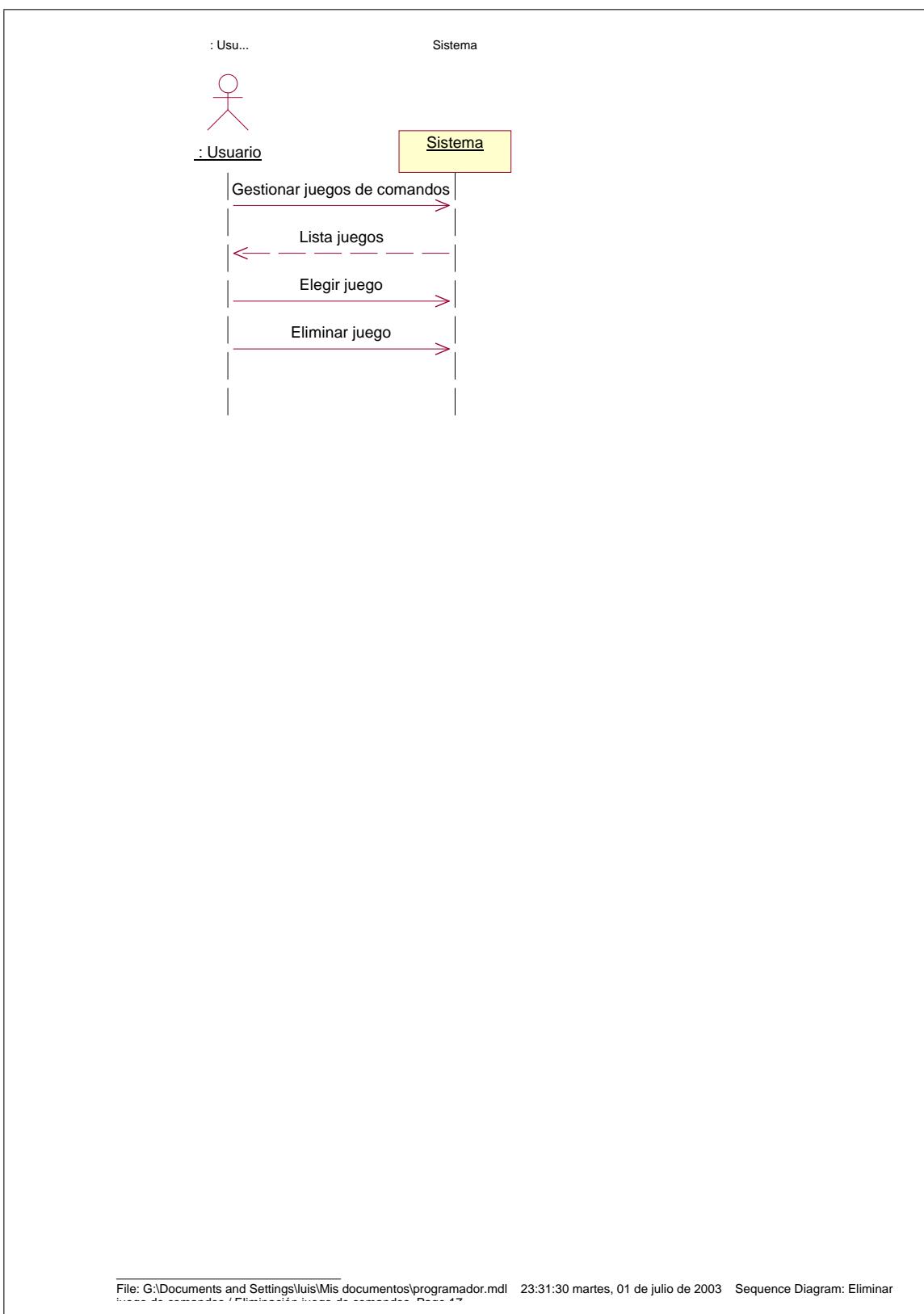
## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO



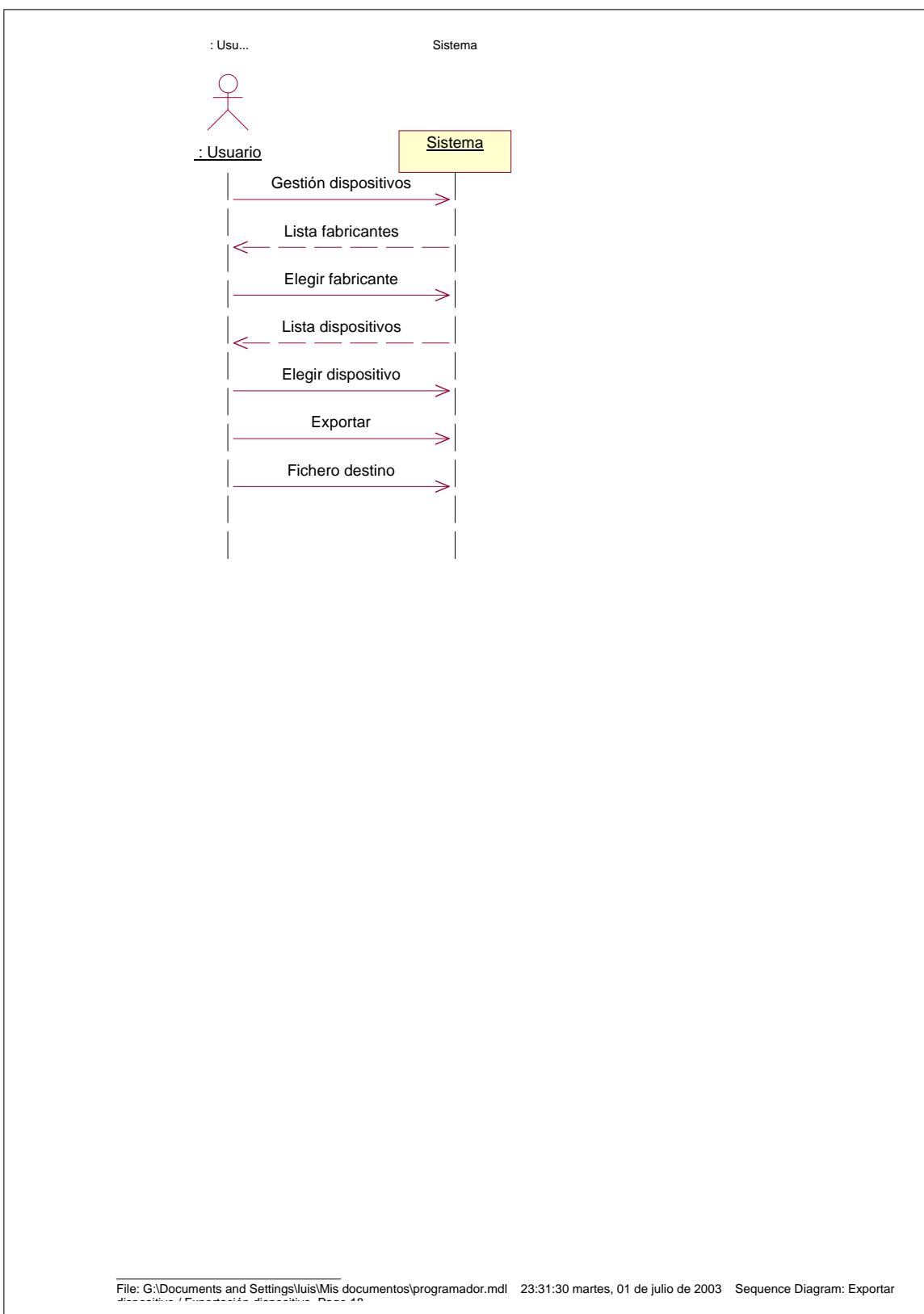
## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO



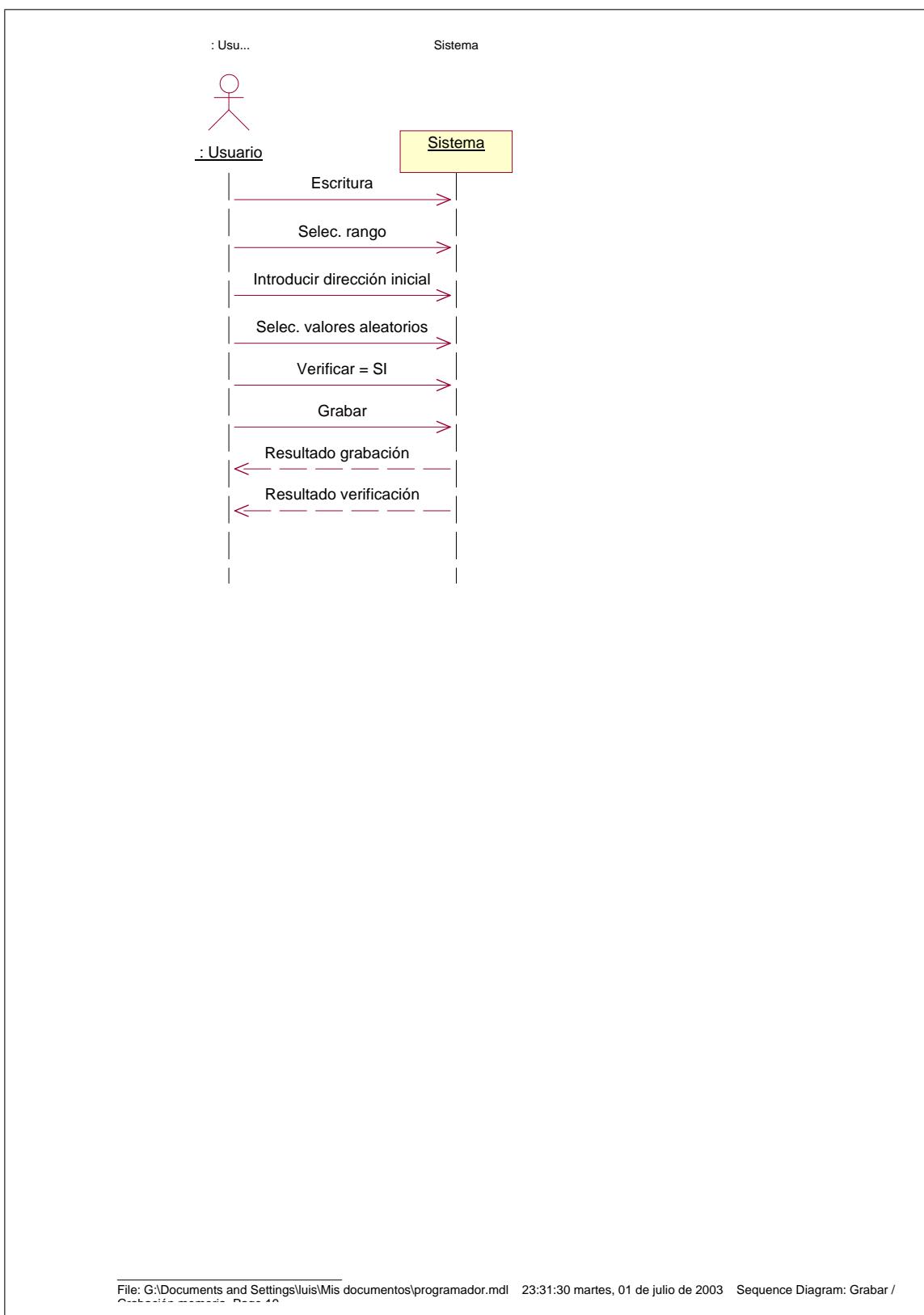
## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO



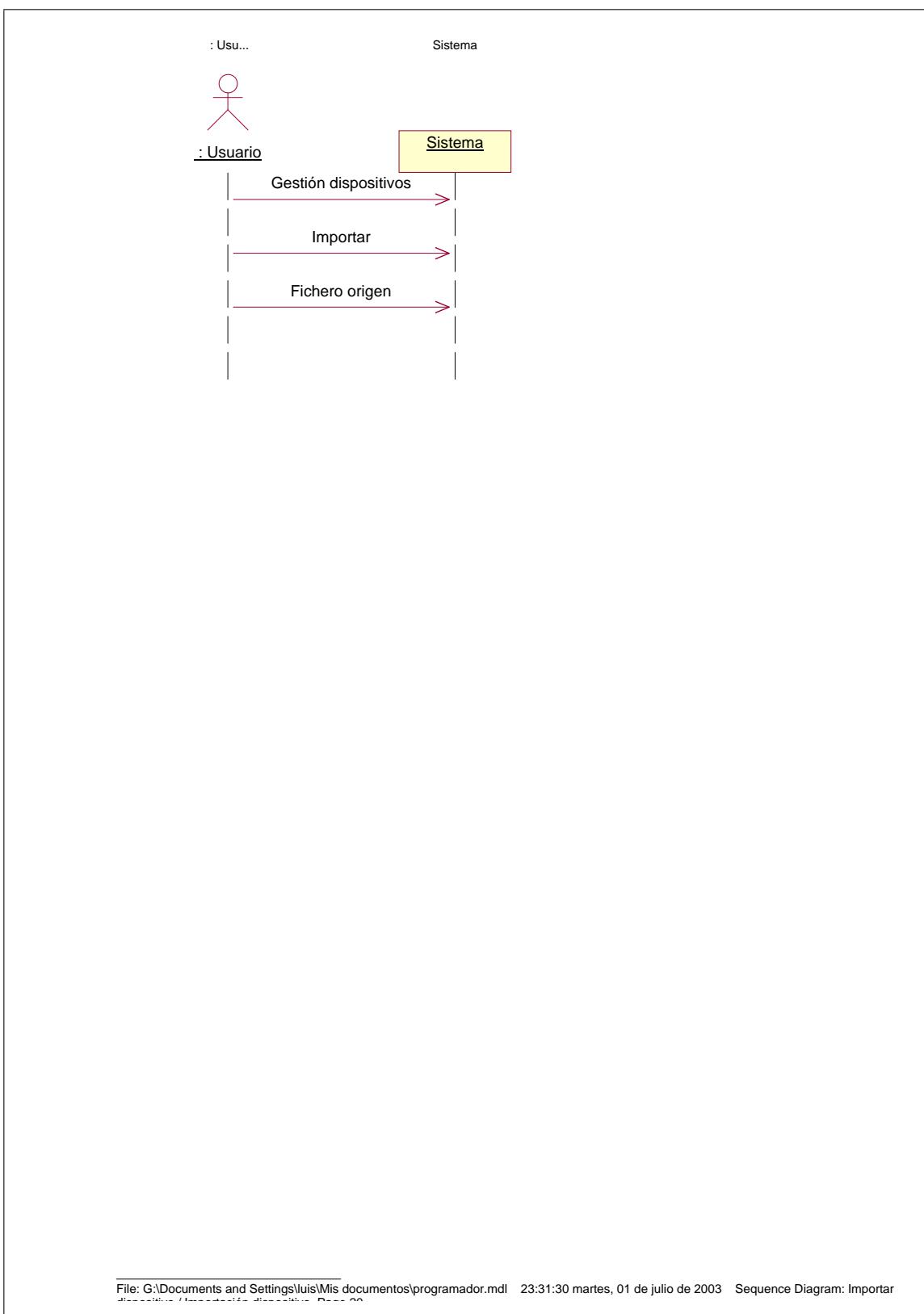
## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO



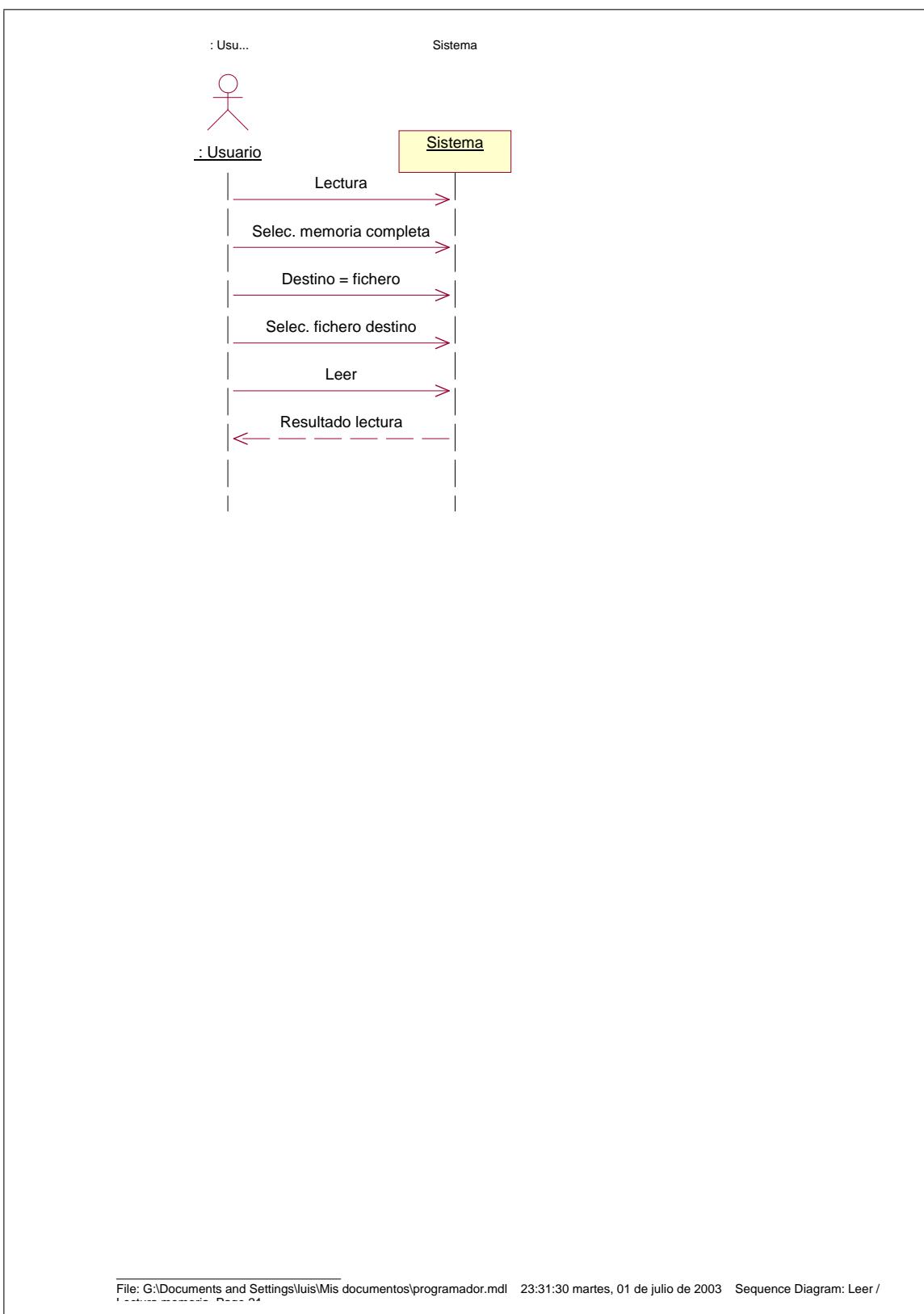
## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO



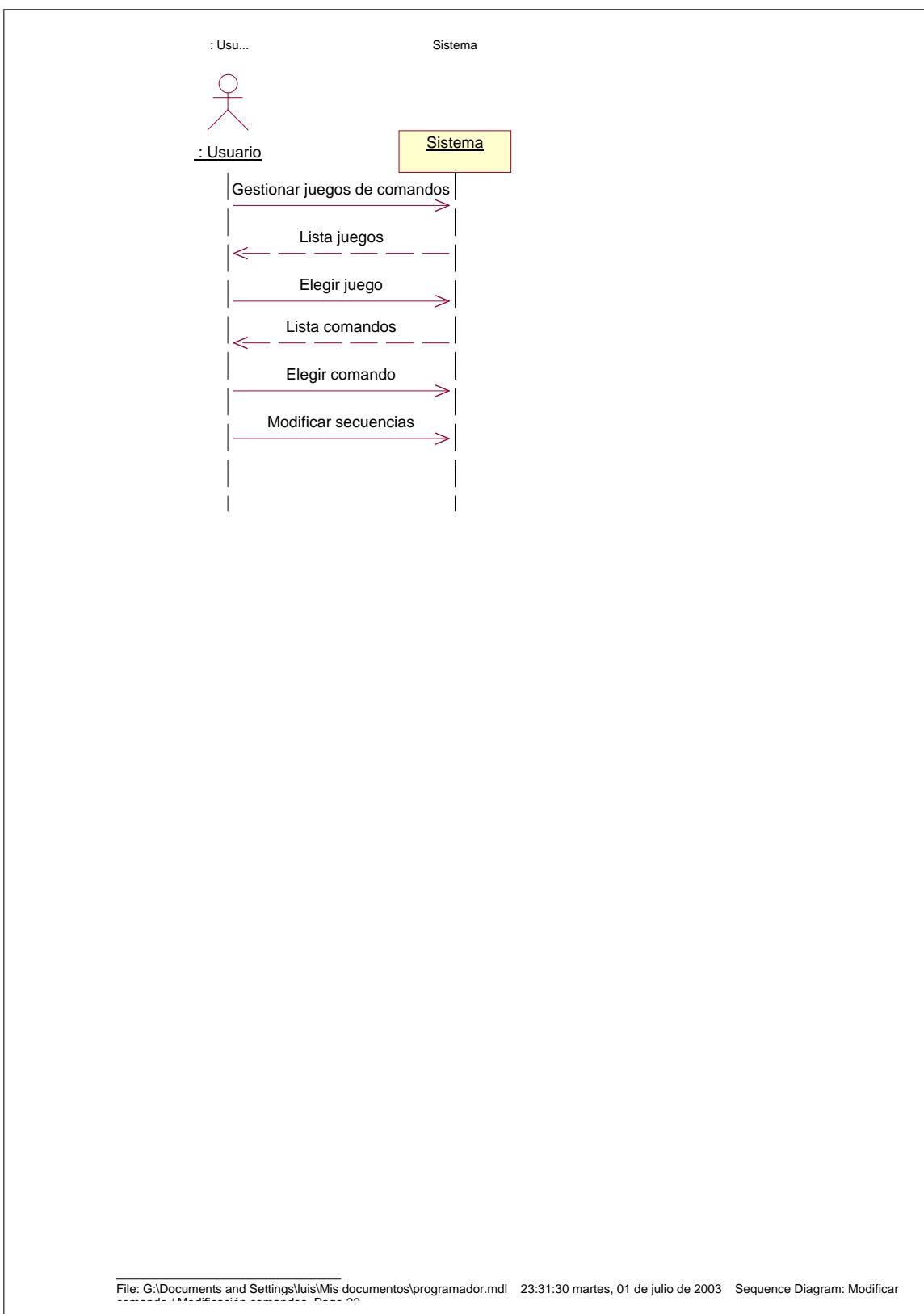
## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO



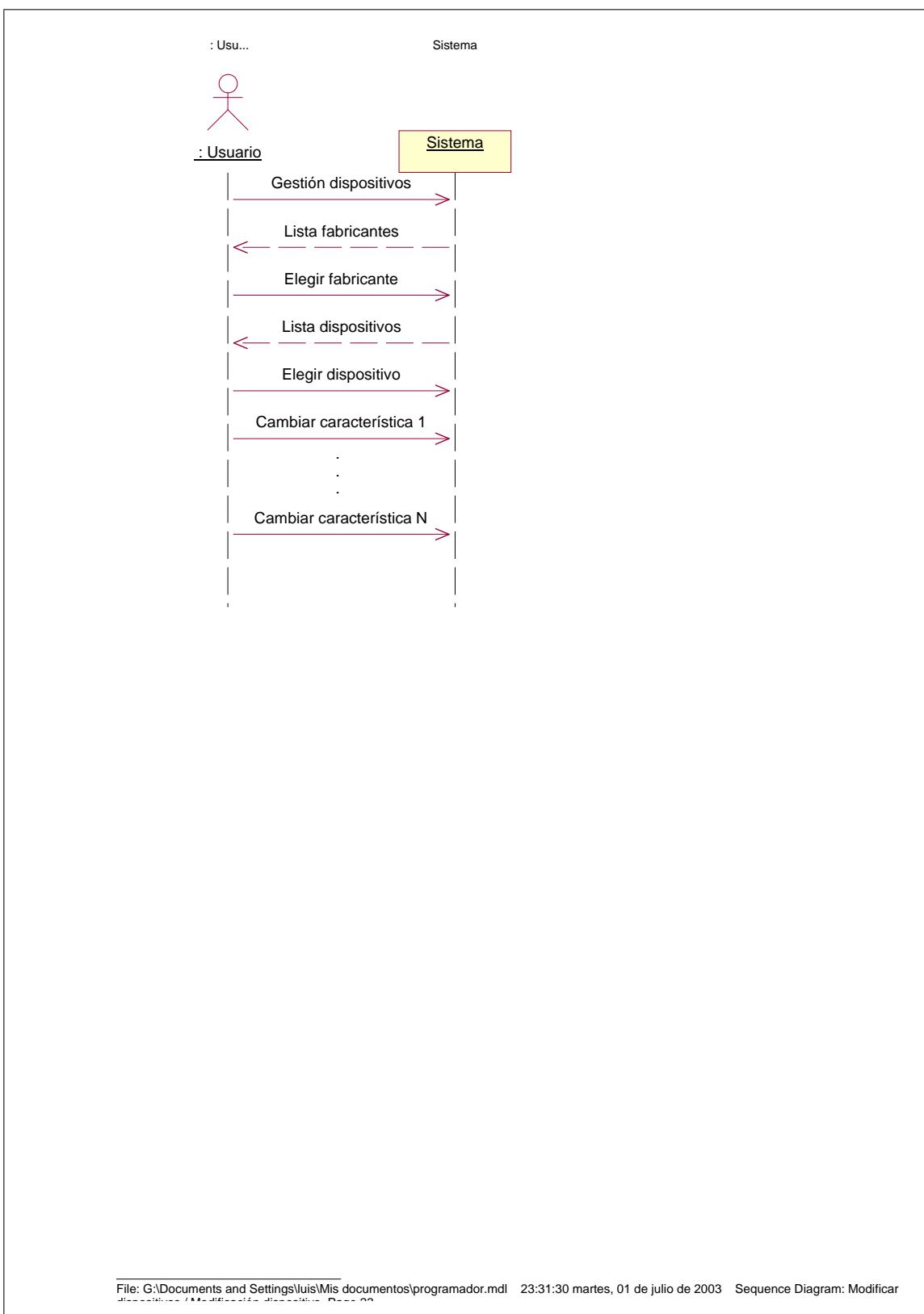
## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO



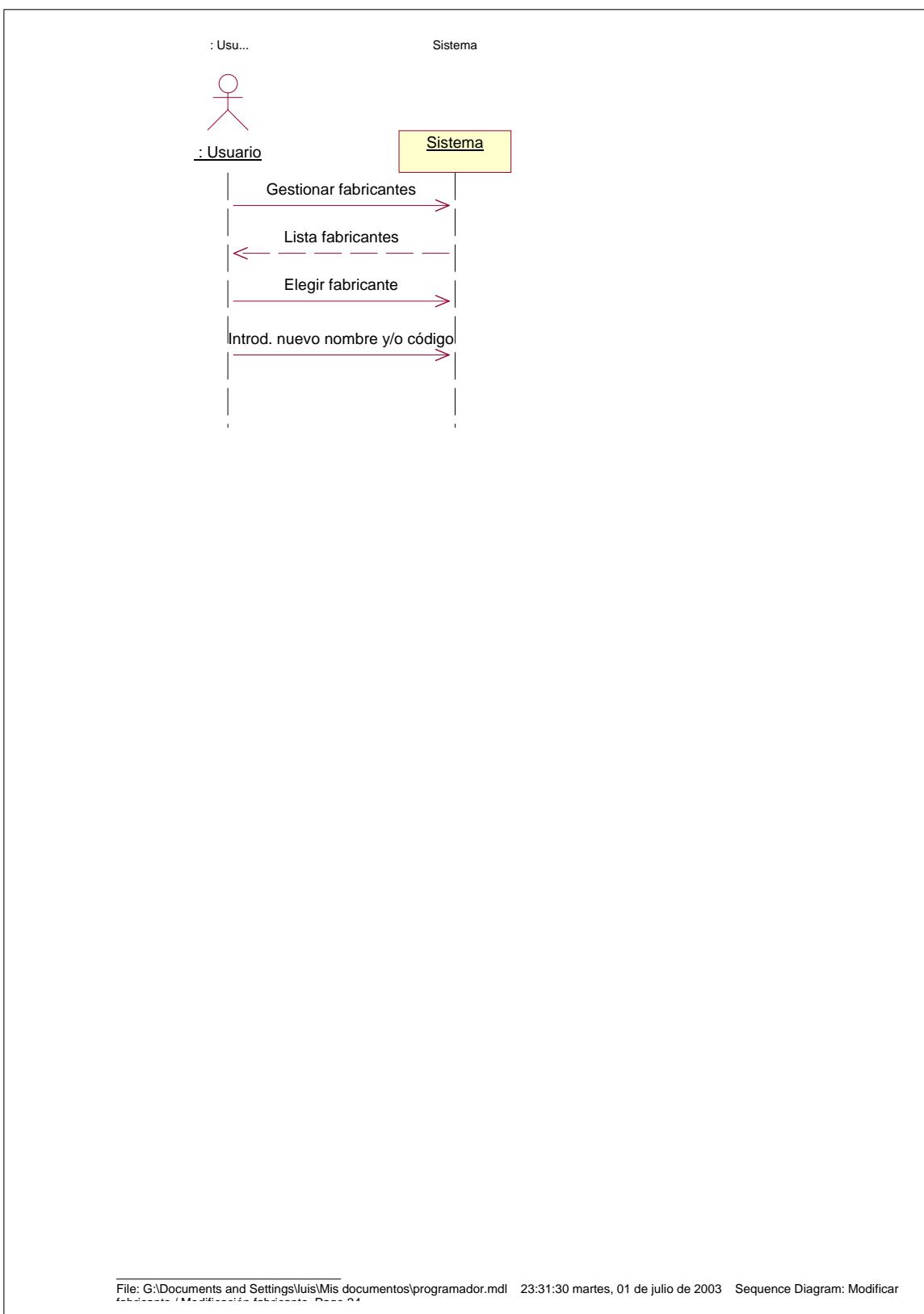
## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO



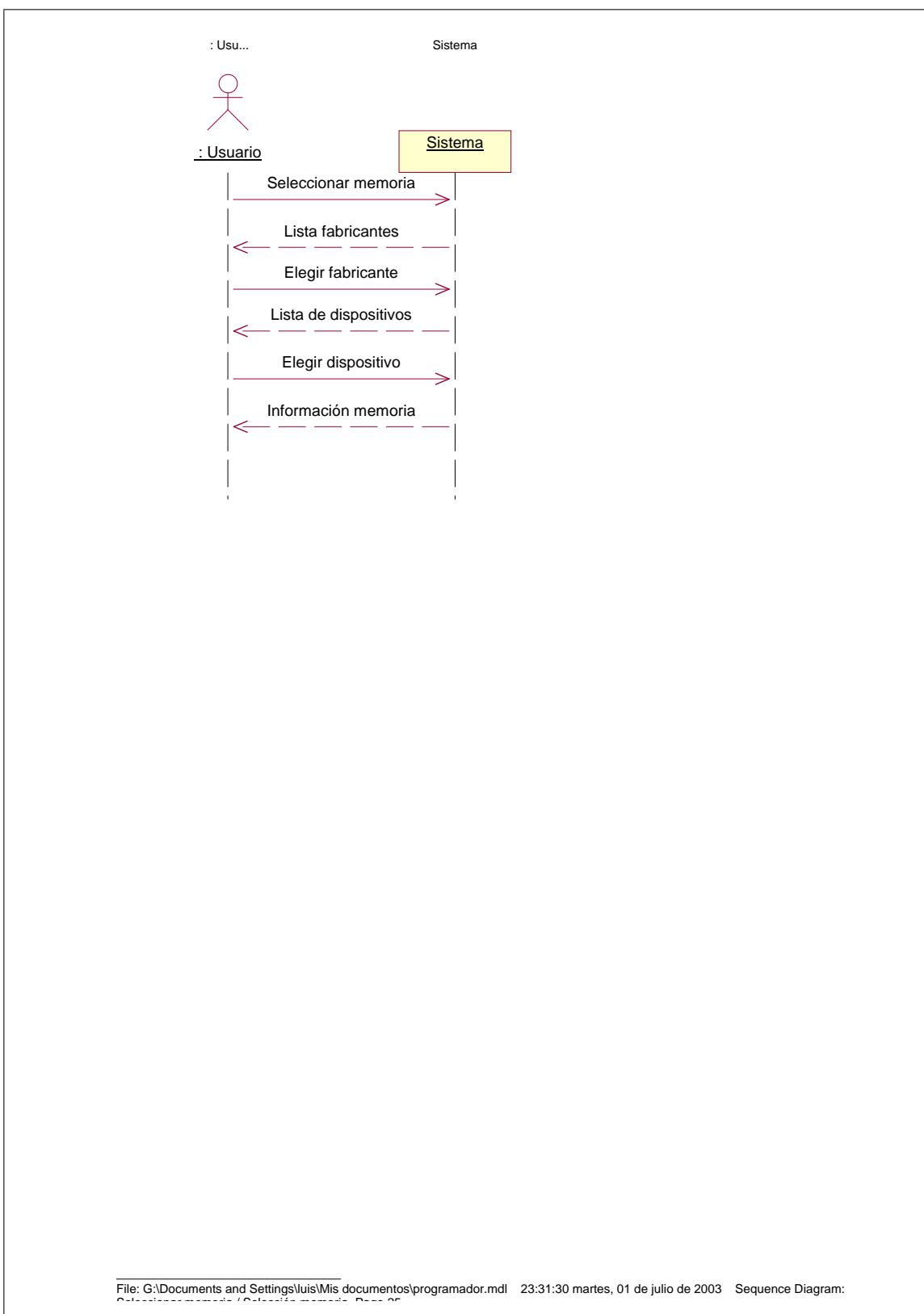
## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO



## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO

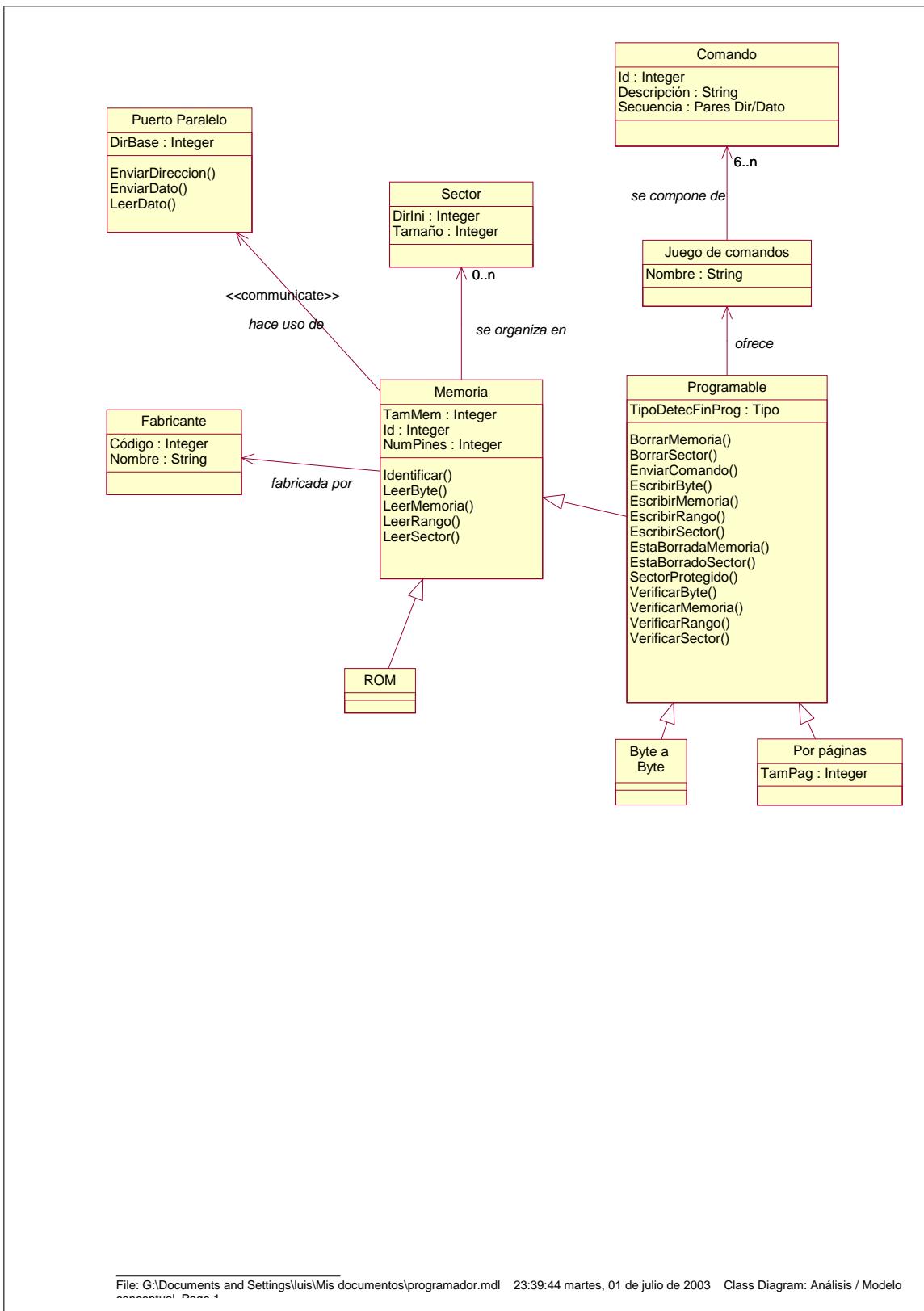


## C.2. DIAGRAMAS DE SECUENCIA DE CASOS DE USO



### **C.3. Modelo conceptual**

### C.3. MODELO CONCEPTUAL

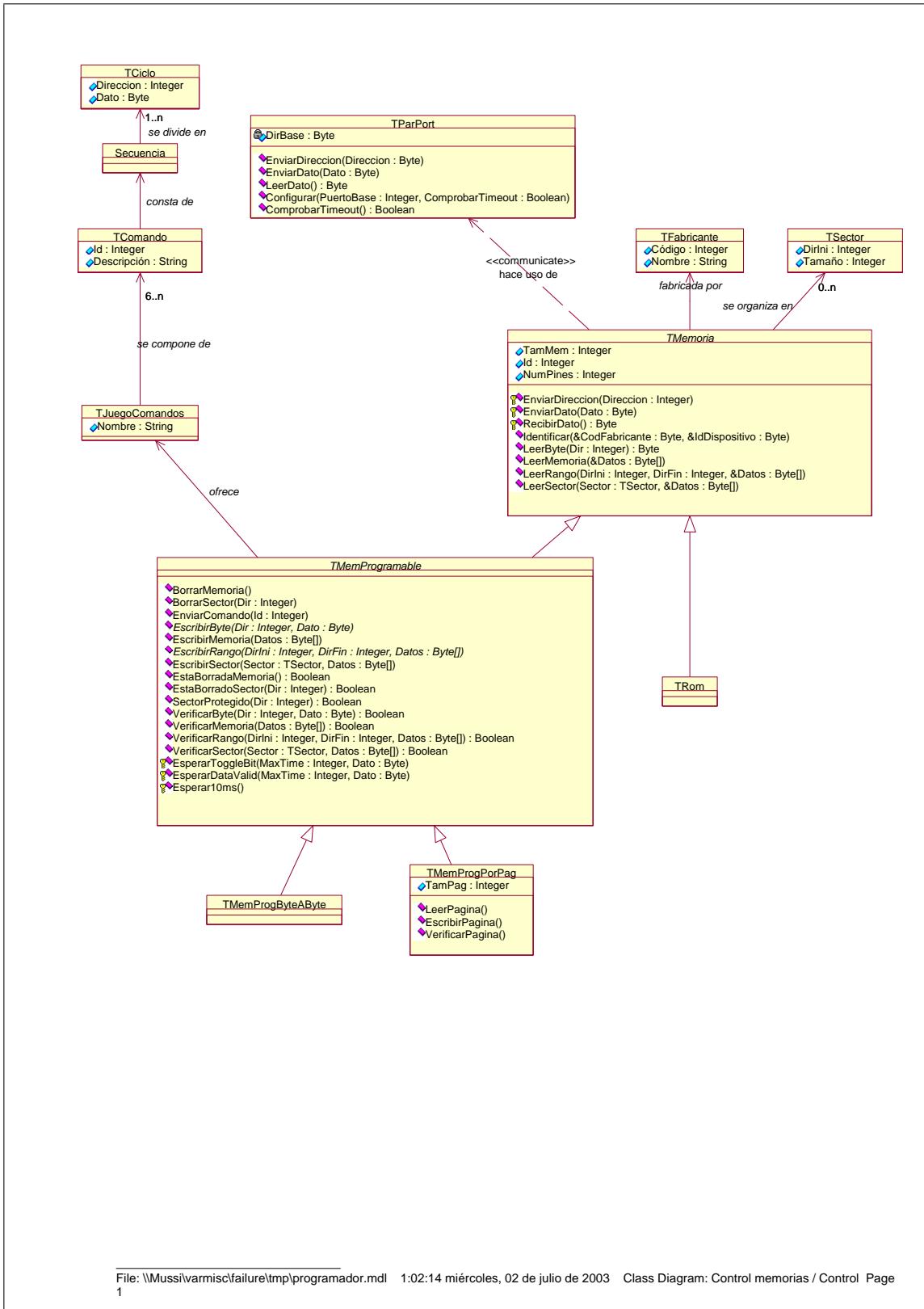


# **Capítulo D**

## **Diagramas de diseño del software**

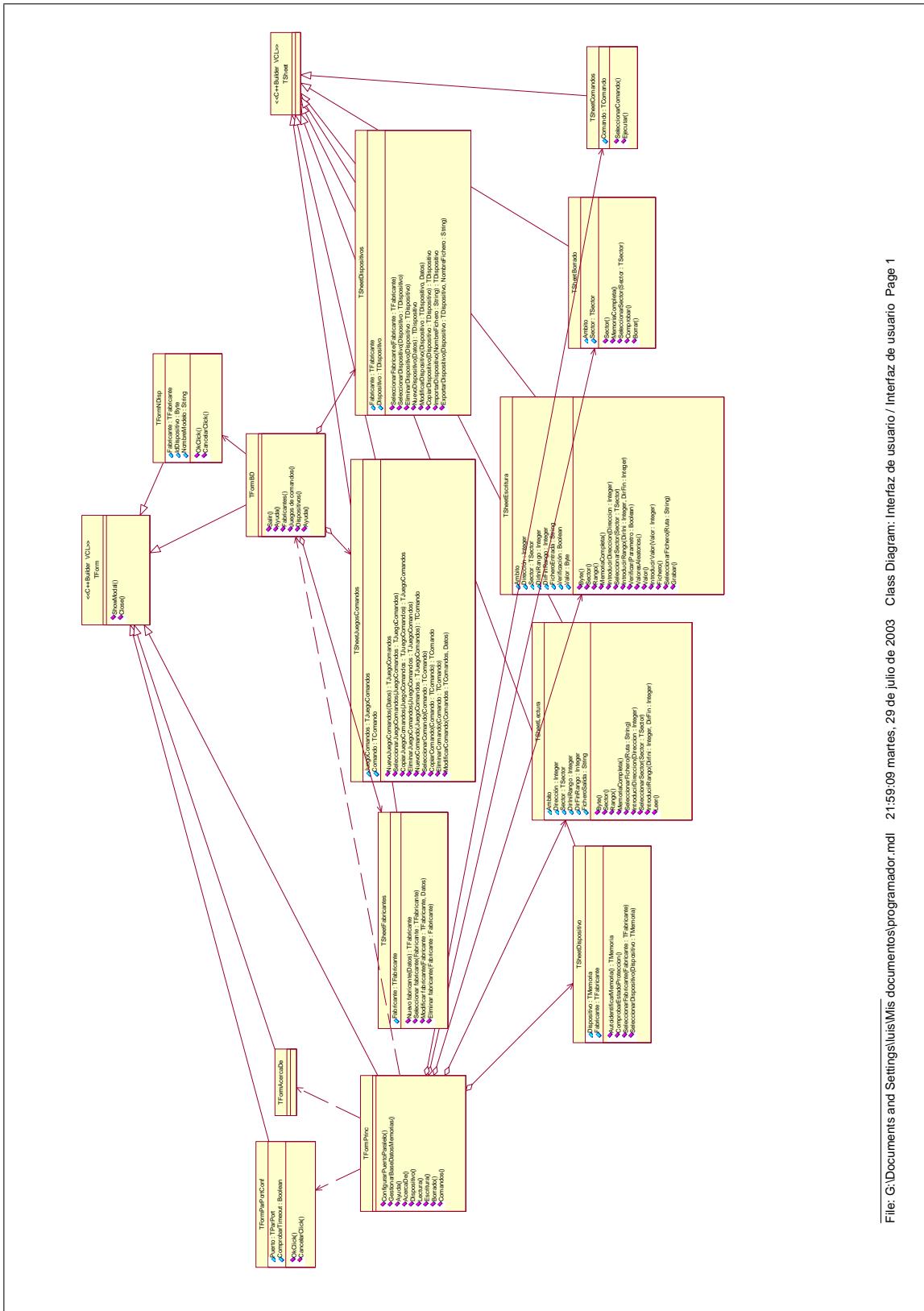
### **D.1. Diagrama de clases de control**

## D.1. DIAGRAMA DE CLASES DE CONTROL



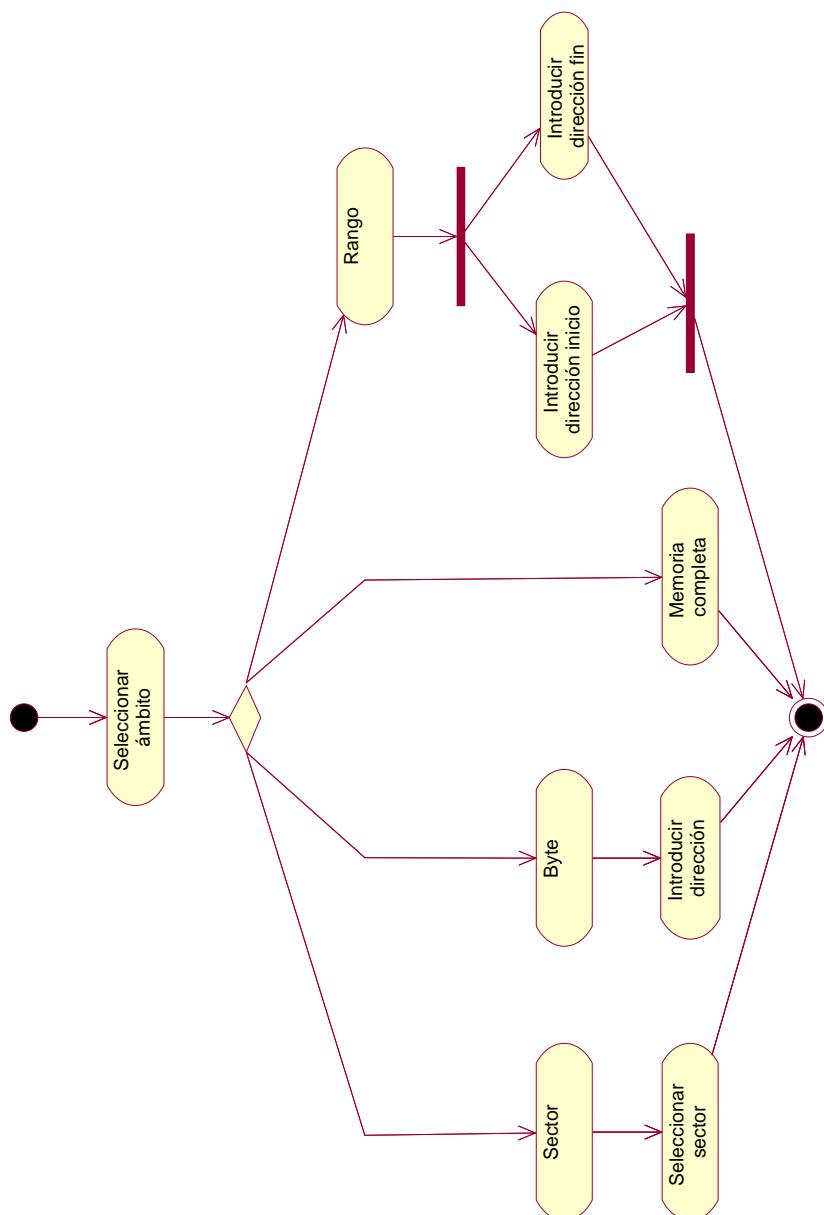
## **D.2. Diagrama de clases de interfaz**

## D.2. DIAGRAMA DE CLASES DE INTERFAZ



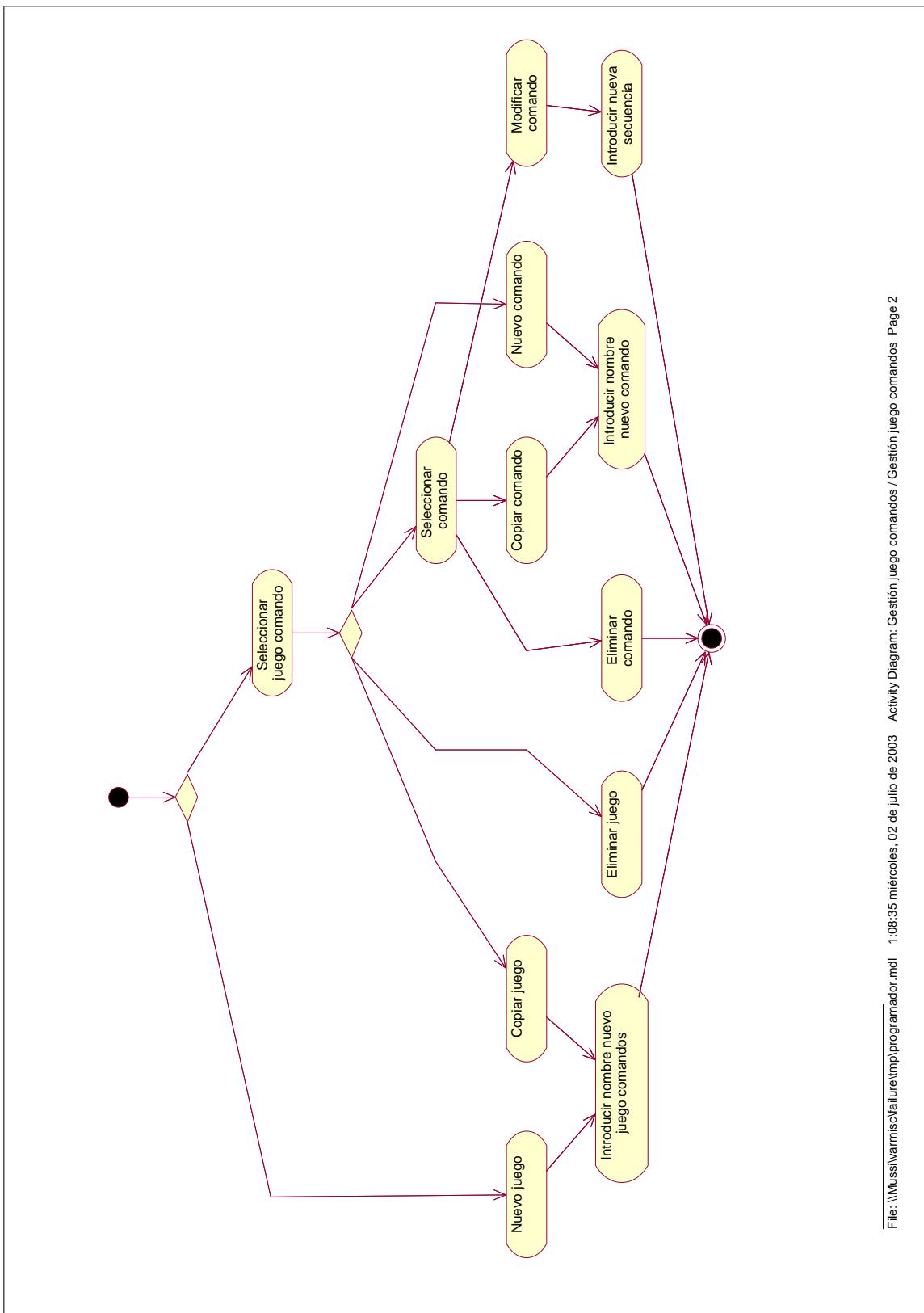
## **D.3. Especificación del comportamiento del interfaz**

### D.3. ESPECIFICACIÓN DEL COMPORTAMIENTO DEL INTERFAZ



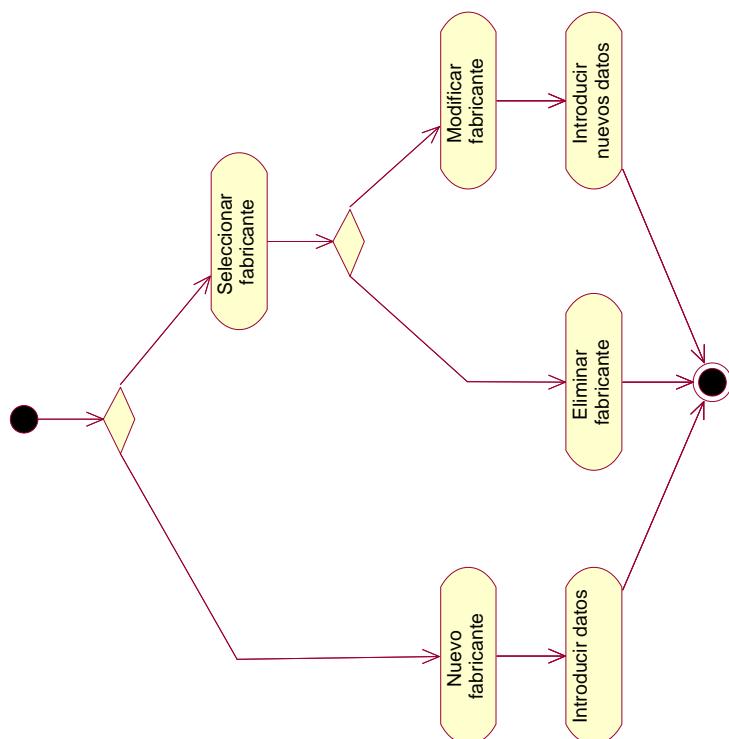
File: \WMS\varmiscafailure\tmp\programador.mdl 1:08:34 miércoles, 02 de julio de 2003 Activity Diagram: Selección ámbito operación / Selección ámbito operación Page 1

### D.3. ESPECIFICACIÓN DEL COMPORTAMIENTO DEL INTERFAZ



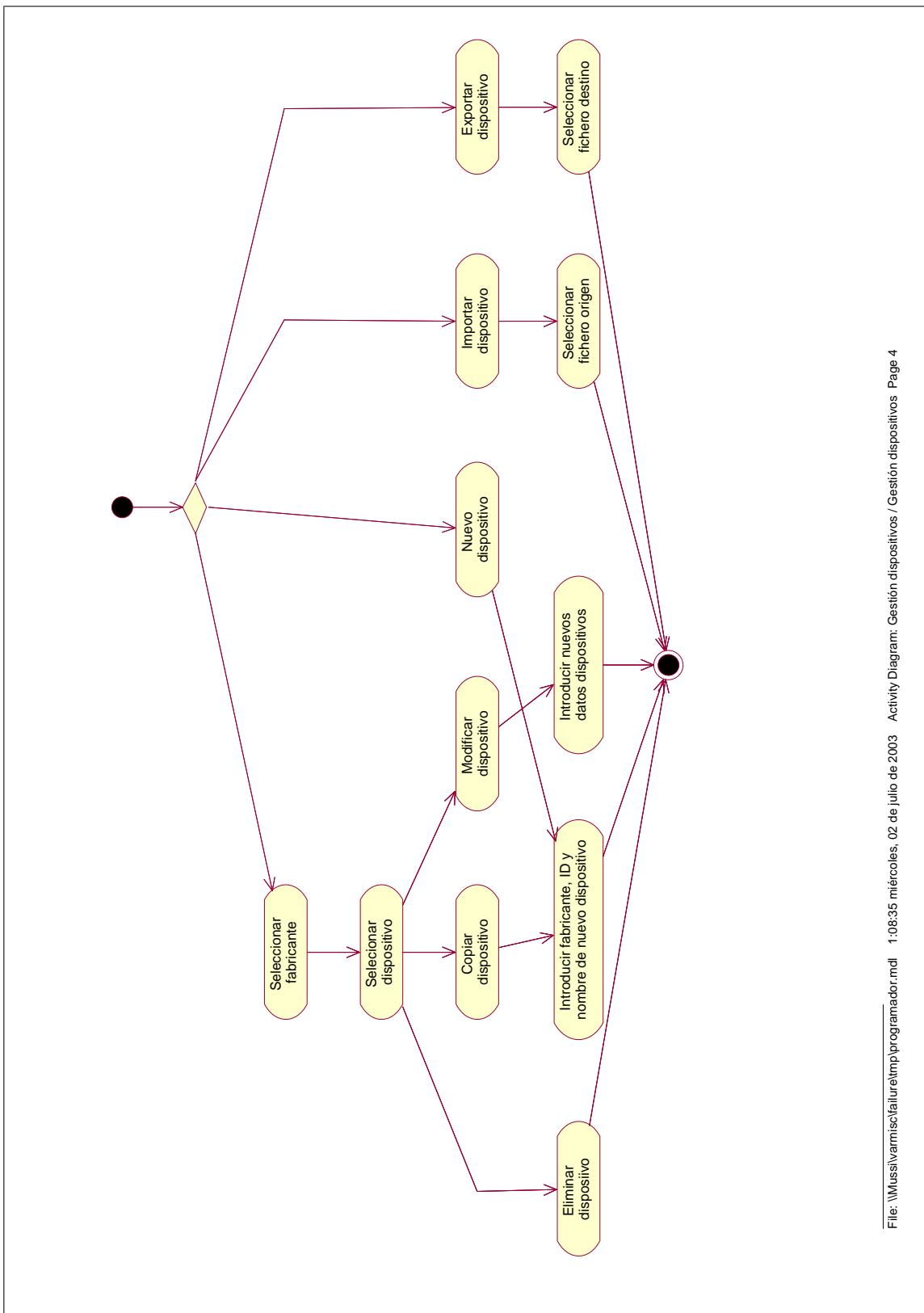
File:\Mussi\var\msc\failure\tmp\programador.mdl 1:08:35 miércoles, 02 de julio de 2003 Activity Diagram: Gestión juego comandos / Gestión juego comandos Page 2

### D.3. ESPECIFICACIÓN DEL COMPORTAMIENTO DEL INTERFAZ



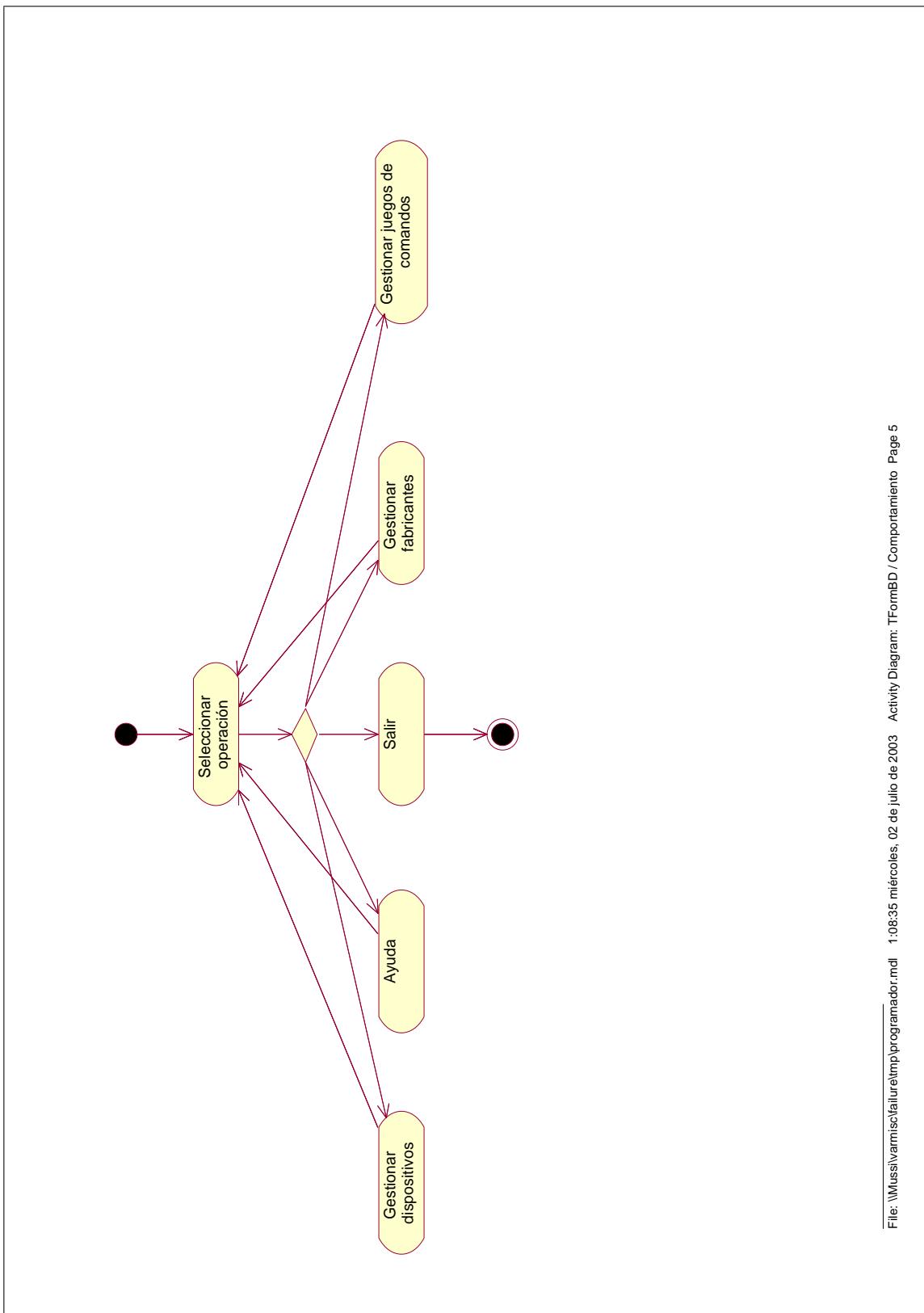
File: \Wms\varnish\failure\tmp\programador.mdl 1:08:35 miércoles, 02 de julio de 2003 Activity Diagram: Gestión fabricantes / Gestión fabricantes Page 3

### D.3. ESPECIFICACIÓN DEL COMPORTAMIENTO DEL INTERFAZ



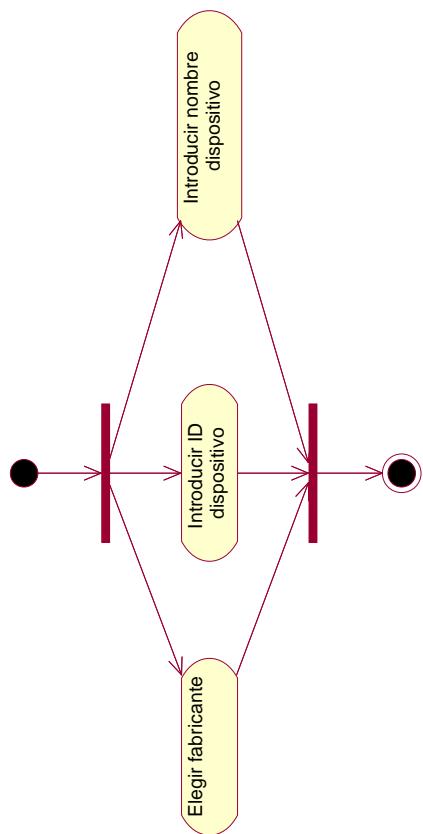
File: \\Mussi\varnish\failure\tmp\programador.mdl 1:08:35 miércoles, 02 de julio de 2003 Activity Diagram: Gestión dispositivos / Gestión dispositivos Page 4

### D.3. ESPECIFICACIÓN DEL COMPORTAMIENTO DEL INTERFAZ



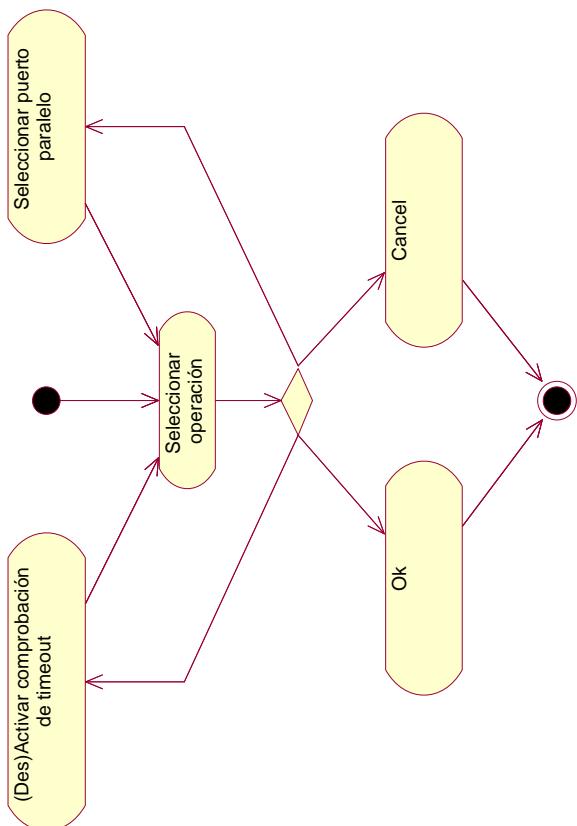
File: \Mus\varmiscafailure\temp\programador.mdl 1:08:35 miércoles, 02 de julio de 2003 Activity Diagram: TFormBD / Comportamiento Page 5

### D.3. ESPECIFICACIÓN DEL COMPORTAMIENTO DEL INTERFAZ



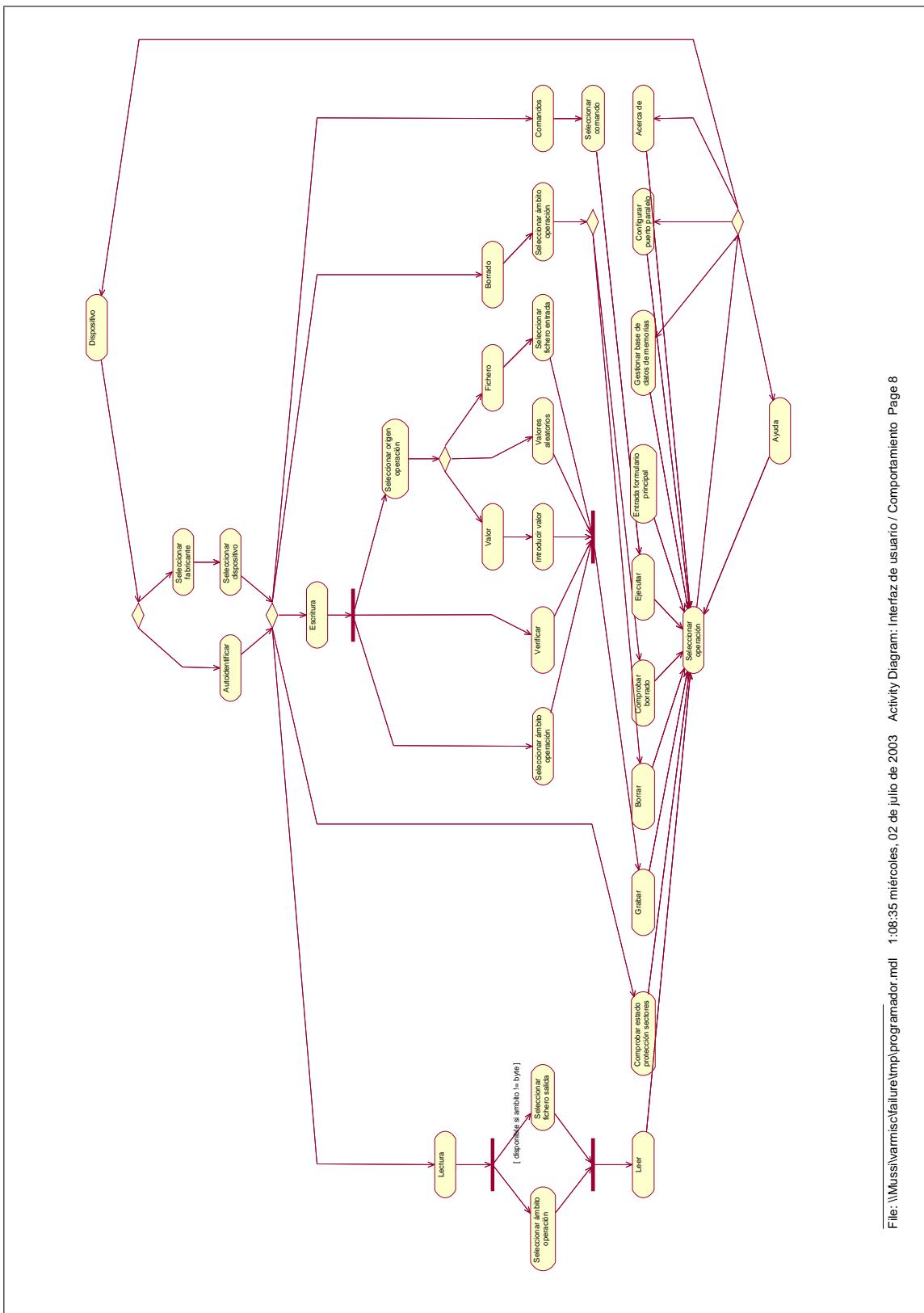
File: \Muss\varmisch\failure\tmp\programador.mdl 1:08:35 miércoles, 02 de julio de 2003 Activity Diagram: TFormNDisp / Comportamiento Page 6

### D.3. ESPECIFICACIÓN DEL COMPORTAMIENTO DEL INTERFAZ



File: \Muss\varmiscafailure\tmp\programador.mdl 1:08:35 miércoles, 02 de julio de 2003 Activity Diagram: TFormPaPortConf / Comportamiento Page 7

### D.3. ESPECIFICACIÓN DEL COMPORTAMIENTO DEL INTERFAZ



---

*D.3. ESPECIFICACIÓN DEL COMPORTAMIENTO DEL INTERFAZ*

# Bibliografía

- [1] Amic, AMD, Atmel, Bright, EON, Mxic, ST, SST, SyncMos, Winbond y Xicor.  
Datasheets de memorias EEPROM y Flash.
- [2] Philips Semiconductors, “Online Datahandbook System”.  
<http://www.philips-semiconductors.com/products/>
- [3] Jeff Frohwein, “Flash/EEPROM Programmer, Rev C”, 2001.  
<http://www.devrs.com/e/tools.php#toolprgr>
- [4] Jeff Frohwein y Sshua, “Frohwein-Sshua Flash Programmer”, 2001.  
<http://www.devrs.com/e/tools.php#toolprgr>
- [5] CadSoft Computer, Inc. “EAGLE 4.0 Tutorial”.
- [6] Daniel D. Gajski. “Principios de Diseño Digital”, Prentice Hall, 1997.
- [7] Craig Peacock. “Interfacing the Standard Parallel Port”, 1998.  
<http://www.beyondlogic.org>
- [8] Craig Peacock. “Interfacing the Enhanced Parallel Port”, 1997.  
<http://www.beyondlogic.org>
- [9] Craig Peacock. “Interfacing the Extended Capabilities Port”, 2000.  
<http://www.beyondlogic.org>
- [10] T-Tech, Inc. “Quick Circuit AMC 2500 User Manual”.
- [11] Inprise Corporation. “Borland C++ Builder 5 Developer’s Guide”.
- [12] John Pappas, “TDLPortIO version 1.2 User Manual”, 1999.  
<http://diskdude.cjb.net/software/cbuilder/index.html>
- [13] R. S. Pressman. “Ingeniería del software: un enfoque práctico”, McGrawHill, 1993, 3<sup>a</sup> edición.

---

## BIBLIOGRAFÍA

- [14] Stroustrup, Bjarne. “The C++ Programming Language”, Addison-Wesley, 1991, 2<sup>a</sup> edición.
- [15] Craig Larman, “Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process (2nd Edition)”, Prentice Hall, 2001, 2<sup>a</sup> edición.