UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA
INSTITUTO DE INVESTIGACIONES ELECTRICAS

DISEÑO Y CONSTRUCCION DE UN CIRCUITO BASADO EN MICROPROCESADOR PARA INTERFAZ ENTRE UNA RED TOKEN BUS (IEEE 802.4) Y DISPOSITIVOS INTELIGENTES DE ADQUISICION DE DATOS (Junction Box)

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO MECANICO ELECTRICISTA
(Area Eléctrica y Electrónica)

P R E S E N T A:

JUAN JOSE GAYTAN HERNANDEZ MAGRO

Director de Tesis (UNAM): Ing. Rodolfo Peters
Codirectores de Tesis (IIE): M. en C. Pedro A. Molina
M. en C. César A. Ortega

MEXICO, D. F. **1995**

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

INSTITUTO DE INVESTIGACIONES ELÉCTRICAS

DISEÑO Y CONSTRUCCIÓN DE UN CIRCUITO BASADO EN MICROPROCESADOR PARA INTERFAZ ENTRE UNA RED TOKEN BUS (IEEE 802.4) Y DISPOSITIVOS INTELIGENTES DE ADQUISICIÓN DE DATOS (Junction Box)

TESIS

que para obtener el título de:

Ingeniero Mecánico Electricista
(Área Eléctrica y Electrónica)

presenta

[redacted]

Director de Tesis (UNAM): Ing. Rodolfo Peters
Codirectores de Tesis (IIE): M. en C. Pedro A. Molina
M. en C. César A. Ortega

Agradecimientos

Al Creador

A Katherine, Caní, Coquí y Papá

A todos los maestros que colaboraron en mi formación,
especialmente al Ing. Rodolfo Peters

A todos mis compañeros,
en especial a los ingenieros Héctor Cázares y Rolando Tamayo

Agradezco al Instituto de Investigaciones Eléctricas en Cuernavaca por el apoyo brindado para realizar esta tesis. Especialmente a Pedro Molina (hardware y muchísimo aprendizaje), Pablo Ibargüengoitia (monitor de pruebas), Dalila Jiménez (recompilación del software), César Ortega (culminación) y a todo el Departamento de Electrónica.

Mención especial para los ingenieros Educardo Soto, Enrique Dávalos y Héctor Suárez, por su apoyo y amistad.

ÍNDICE

1. INTRODUCCIÓN

Un sistema de producción está formado por diferentes procesos. Cada proceso se encarga de una tarea específica, que puede realizar en forma autónoma o dependiente de otros procesos. Al conjuntar los procesos se debe obtener, como resultado, un producto o servicio disponible al mercado. Por ejemplo, supongamos que se desea producir autos. El *sistema* de producción de autos puede dividirse en diferentes *procesos:* uno para obtener el chasis, otro para el motor, otro para los interiores etc. Algunos de esos procesos dependen, a su vez, de otros sub-procesos. Por ejemplo, la fabricación del motor depende de los sub-procesos de obtención de todas las partes que lo forman: bujías, filtro, carburador, inyector de gasolina etc. Al conjuntar todos los procesos debe obtenerse, finalmente, un automóvil completo.

El desempeño de un sistema de producción depende de la calidad, conexión e interrelación de los procesos que lo conforman. Es decir, para nuestro ejemplo, que el desempeño de nuestra fábrica de autos depende de la calidad que se tenga en la fabricación de cada parte del automóvil, de la relación que exista entre ellos –fabricar las cantidades necesarias de cada elemento: un motor, cuatro asientos etc.– y de la posibilidad de ensamblarlas sin problemas –tener disponibles las partes necesarias y que todas sean compatibles–.

El control de cada uno de dichos procesos y de sus relaciones es lo que permite la puesta en marcha de una planta de producción, es decir, del sistema total. Todo sistema tiene variables que pueden ser sensadas y medidas para generar reportes del estado de cada proceso o dar alarmas. Esta información necesita ser recolectada, almacenada y procesada desde el nivel más bajo del sistema, para que después sea diseminada y comunicada logrando tener el control total del sistema de producción.

Anteriormente, se contaba con muchos operarios en los procesos, que realizaban las acciones de control por sí mismos. La toma de decisiones era muy lenta pues los humanos tenemos una velocidad límite de respuesta. Además, los procesos se han tornado cada vez más complejos, dificultando la labor del control para los operadores. Debido a las limitaciones en velocidad y capacidad cuando el control es complejo, los operadores han sido desplazados por computadoras y sistemas digitales. De esta manera surgieron los *sistemas digitales de control.*

Durante la década de los 80's los procesos de control eran centralizados, es decir, una sola computadora maestra se encargaba de todo el trabajo de control de un sistema. Esta tendencia ha cambiado a Sistemas de Control Distribuido –SCD–

gracias a la capacidad de añadir *inteligencia* a diversos dispositivos del sistema, por lo que se dice que el sistema tiene *inteligencia* distribuida. Los SCD están formados por subsistemas de control que trabajan en paralelo, de este modo la carga de trabajo que sufría la computadora central se reparte y permite la división de trabajo, cuyo resultado es un sistema controlado de forma más robusta, más rápida y de manera muy eficiente. Entiéndase *inteligencia* en un dispositivo como su capacidad de procesar información manejada en forma digital.

En su nivel más bajo un sistema de control debe adquirir datos provenientes de los procesos, mediante *dispositivos inteligentes de adquisición de datos* que se encuentran conectados en red. Los datos son información obtenida del adecuamiento de señales que pueden ser medidas. Una señal puede entenderse como cualquier cantidad física por la cual un sistema o elemento de éste influye sobre otros sistemas o elementos del mismo. En el ejemplo de los autos, tal vez nos interese saber la potencia eléctrica que consumen los motores de las máquinas de producción para controlar el factor de potencia de la planta, y utilicemos dispositivos de adquisición de datos para los medidores de potencia.

La información recolectada en el nivel más bajo es transportada por la red de adquisición –o bus de campo– hacia sistemas digitales de control. Desgraciadamente, las empresas que fabrican dispositivos inteligentes de adquisición de datos no han seguido ningún estándar en el manejo de la información, sino que cada una la maneja con un formato propio. De esta forma, si se tienen dispositivos de adquisición de datos de diversas compañías la transmisión de la información hacia niveles superiores de control debe hacerse por diferentes redes –dada la incompatibilidad de las señales–, y resulta imposible suministrar al sistema de control entradas para todos los tipos de dispositivos del mercado. Si el sistema de control cuenta con un solo medio de transmisión se descubre un problema de cautividad de mercado, pues la selección de dispositivos se limita a los de una compañía en particular. Entonces la capacidad de desarrollo, el mejoramiento de los sistemas de control y la adquisición de nueva tecnología se ven limitadas pues dependen exclusivamente del avance tecnológico de dicha compañía.

Para superar estas deficiencias se ha creado el concepto de sistemas abiertos en los que es posible la interconexión de diversos dispositivos inteligentes. En los sistemas abiertos la cantidad de información es elevada y su manejo tiene gran complejidad, por lo que es necesario desarrollar sistemas de comunicación que permitan el intercambio de información entre subsistemas de control.

En esta tesis se describe el desarrollo de un subsistema de comunicaciones para sistemas abiertos de control, que permite la conexión de diversos dispositivos de adquisición de datos y soluciona los problemas de transmisión de la información en el nivel más bajo de un sistema de control.

Formalmente, esta tesis consiste en el desarrollo de un subsistema de comunicaciones para una arquitectura de control distribuido en sistemas abiertos, que satisface los requerimientos del modelo de interconexión de sistemas abiertos –OSI– para el nivel de ligado de datos (data link layer).

En el segundo capítulo se proporciona información sobre los SCD, se describe el modelo OSI (Open Systems Interconnection) para sistemas de control y se logra ubicar al sistema desarrollado en la tesis en el contexto de un SCD, indicándose los objetivos a alcanzar según los requerimientos expuestos.

En el tercer capítulo se expone la metodología de diseño empleada, se describen los principios de diseño y se describe al sistema como un diagrama de bloques.

En los capítulos cuarto y quinto se describen el *hardware* y el *software* del sistema en forma técnica.

En el capítulo sexto se evalúa el sistema en forma global y se presentan resultados y conclusiones finales.

2. PLANTEAMIENTO DEL PROBLEMA

2.1 SISTEMAS DE CONTROL DISTRIBUIDO (SCD)

Un sistema de control distribuido es aquel que se forma por varios subsistemas de control que cuentan con inteligencia propia. Esta división en subsistemas se basa en el hecho de que cualquier proceso de control puede subdividirse en unidades básicas de proceso[1]. Dichos subsistemas poseen controladores que son capaces de efectuar el control de un procedimiento específico –una unidad de proceso– en forma independiente, de tal forma que satisfacen totalmente su parte en el funcionamiento del sistema global. El empleo de subsistemas de control permite que la tarea de procesamiento y control en el sistema total sea descentralizada. En la figura 2.1 se muestra un modelo gráfico de SCD.

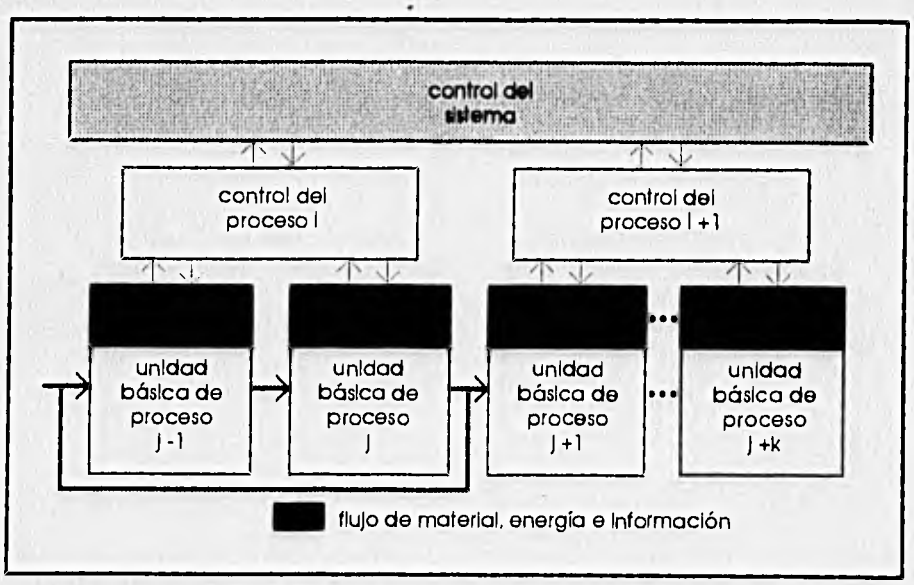


Fig. 2.1 Sistema de control distribuido (SCD)

Un sistema de control distribuido necesita, a cambio, una robusta estructura de comunicaciones para coordinar y compartir la información procesada entre subsistemas. Esta estructura debe desempeñar las siguientes acciones: adquisición de

[1]Postulados de Z. Kehler, 1975

datos, ejecución de comandos de corte intempestivo (shutdown commands), arbitrio para la puesta en marcha del sistema y presentación del estado de los procesos a operadores.

Desde un punto de vista informático, es posible definir un modelo piramidal en el que se divide a un sistema de control en niveles de acuerdo a la cantidad y la calidad de la información[2]. Este modelo, propuesto por la Instrument Society of America –ISA–, se representa en la figura 2.2.

Los niveles del sistema de control son el empresarial, el de planta, el de supervisión, el de la unidad de control y el de proceso; ordenados de acuerdo a la cantidad de información que se maneja en cada uno, de mayor a menor.

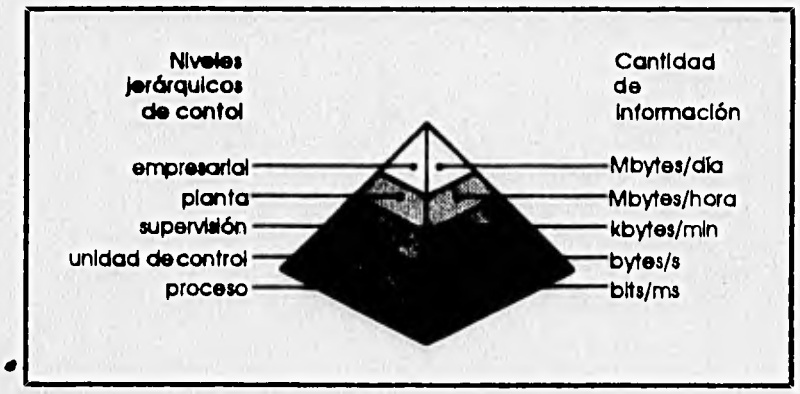


Fig. 2.2 Modelo piramidal de control

- Nivel empresarial (5): en este nivel se planean las estrategias y se evalúa la operación del sistema como un todo. Es equivalente a una junta de directivos en una empresa. La información procede de varias fuentes. El esquema de comunicaciones incluye una amplia red que interconecta diferentes computadoras, terminales y medios de almacenamiento de datos que pueden ser accesados alrededor del país o aún del mundo. La información se colecta por medio de una red de área amplia o WAN (Wide Area Network), cuyo volumen de datos es del orden de MBytes/día. Las características funcionales son:
  - conexión hacia el nivel planta y de supervisión mediante WAN

[2] *An Integral Approach for the field, control and supervisory levels of a DCS.* Documento para la Industrial Computing Society Conference ICS/93, por Pedro A. Molina, Pablo H. Ibargüengoitia y Jorge Hermosillo del Instituto de Investigaciones Eléctricas. México, 1993

- conexión hacia *displays* para presentación compleja de datos
- emisión de comandos para subordinar los niveles de control inferiores, de modo que se logre la coordinación y el control total del desempeño del sistema
- cálculo de predicciones, planeación y desarrollo del sistema
- manejo del sistema

- Nivel de planta (4): en este nivel se maneja información obtenida de los distintos subsistemas del sistema total. Los subsistemas equivalen a distintos departamentos de una empresa, y generalmente están ubicados en la misma instalación. Por ello, para colectar la información se utiliza una red de área local o LAN (Local Area Network), con estaciones de trabajo (workstations) y minicomputadoras, que maneja un volumen de datos de MBytes/hora. Las características funcionales son:
  - conexión entre los niveles empresarial y de supervisión mediante LAN
  - supervisión y coordinación de variables de procesos
  - modelado del control de procesos
  - optimación del control de procesos

- Nivel de supervisión (3): en este nivel se maneja la información para optimar el proceso, producir y manejar alarmas y manejar otros eventos críticos. Las acciones son realizadas por un operador que trabaja con una estación de trabajo con interfaz tipo hombre-máquina que contiene información recogida de los niveles más bajos. El tiempo típico de recolección de datos es de milisegundos y se realiza en un área pequeña, por lo que también se utiliza una pequeña LAN. Las características funcionales son:
  - conexión hacia consolas de supervisión
  - recepción de datos procedientes del proceso
  - emisión de comandos y señales de control hacia los procesos y controladores
  - conexión hacia el nivel de proceso mediante comunicación serial o pequeñas LAN
  - conexión hacia el nivel de control mediante LAN

- Nivel de la unidad de control (2): en este nivel se manejan dispositivos en tiempo real y se aplican algoritmos de control. Los dispositivos empleados varían según la aplicación, pero un ejemplo clásico es el uso de computadoras lógicas programables o PLCs. La información se obtiene de los niveles más bajos en milisegundos y se realiza mediante una LAN pequeña o conexiones punto a punto. Las características funcionales son:
  - recepción de señales desde transductores o convertidores
  - emisión de señales de control y comandos hacia actuadores
  - recepción de comandos provenientes del nivel de proceso y emisión de comandos provenientes del nivel de supervisión
  - control de encendido/apagado, protección, funciones aritmético-lógicas para control de procesos
  - indicación del estado del proceso hacia el nivel de supervisión

- Nivel de procesamiento (1): este nivel está formado por las interfaces con los dispositivos de campo o dispositivos inteligentes de campo que procesan información generada en unos cuantos milisegundos o incluso microsegundos. Las características funcionales son:
  - recepción de señales desde el nivel de control que van dirigidas a transductores o convertidores
  - emisión de señales de control hacia actuadores
  - emisión de señales procesadas hacia los niveles de control y supervisión
  - recepción de comandos provenientes de los niveles de control y supervisión
  - cálculo complejo de datos del proceso
  - indicación del estado de las señales del proceso
  - regulación de funciones

Para ilustrar el modelo piramidal, supóngase que se desea controlar la generación de electricidad. A nivel de procesamiento se encuentran sensores en válvulas y turbinas –diseminadas en toda la planta– que proporcionan mediciones de presión, temperatura etc. Estas mediciones son recolectadas, en una conexión punto a punto, a nivel de unidad de control por controladores que permiten la apertura y cierre de válvulas, el aumento o disminución de la presión etc. Los datos son transmitidos mediante una red hacia el nivel de supervisión, que monitorea el estado de los dispositivos en estaciones de trabajo dedicadas a proporcionar el estado de la

planta. Gracias al conocimiento del estado de cada planta generadora proveniente del nivel de supervisión, se coordina la generación de electricidad a nivel planta. Por último, recolectando la información obtenida a nivel planta, es posible coordinar en el nivel empresarial la generación de electricidad de cada centro y sus vías de transmisión para todo el país, en una labor semejante a la del CENACE.

2.2 INTERCONEXIÓN DE SISTEMAS ABIERTOS DE CONTROL (OSI)

Las empresas que fabrican dispositivos inteligentes de adquisición de datos ofrecen al mercado sistemas que no poseen ningún estándar en su plataforma de comunicaciones, por lo que no existe compatibilidad con otros dispositivos inteligentes excepto con los fabricados por la misma empresa. Si se utilizan única y exclusivamente sistemas inteligentes de una sola compañía se elimina el problema de compatibilidad, pero se descubren dos problemas importantes. El primero es la limitación en la selección de dispositivos de control, pues el mercado está limitado a una sola empresa y es muy probable que ésta no posea la variedad deseada en una cierta aplicación. El segundo es que la capacidad de desarrollo, el mejoramiento de los sistemas de control y la adquisición de nueva tecnología dependen exclusivamente del avance tecnológico de la empresa de control a la que se esté sujeto.

La necesidad de superar las desventajas de cautividad de mercado y supeditación del desarrollo tecnológico al utilizar SCD obligó el análisis de los sistemas abiertos de control (Open Systems). El punto clave es resolver la *interconexión* entre subsistemas provistos por diversas empresas, tanto en lo referente a hardware como a software.

La International Standard Organization –ISO–, con objeto de resolver el problema de interconexión para sistemas de control, propuso un modelo llamado OSI (Open Systems Interconnection) en el que se definen siete niveles conceptuales que abarcan todos los aspectos de la comunicación. Este modelo tomó como base la definición de los estándares MAP (Manufacturing Automation Protocol) que fueron propuestos por General Motors con el apoyo de más de 2000 compañías de control. El modelo OSI se describe en forma de tabla. En adelante, al citar el modelo OSI se estará haciendo referencia a la especificación MAP de GM.

Nivel	*Protocolo MAP*	*Funciones y servicios*
7. Aplicación (application layer)	Usuario ISO–CASE ISO File Transfer MAT Messaging MAP Directory Service MAP Network Managing Realización por software	Funciones: – control del sistema y de la aplicación Servicios: – identificación de socios en comunicaciones – definición de disponibilidad – autorización – revisión de validaciones – acuerdo de los recursos disponibles – aceptación de servicios – sincronización de aplicaciones – responsabilidad en eliminación de errores – selección del tipo de diálogo – revisión de integridad de datos – adherencia a la sintaxis de los datos
6. Presentación (presentation layer)	No existente Realización por software	Funciones: – solicitud de inicio y desarrollo de sesiones – transferencia de datos – coordinación y conversión de sintaxis – coordinación y conversión del perfil de la presentación Servicios: – conversión de la sintaxis de los datos – formateo de datos – selección de sintaxis – selección del perfil de la presentación
5. Sesión (session layer)	ISO Session Kernel Realización por software	Funciones: – coordinación de la sesión y conexión del transporte – control del flujo de datos en la conexión de la sesión – intercambio acelerado de datos – restablecimiento de la conexión de la sesión – manejo del nivel de sesión – desconexión de la sesión Servicios: – inicio y fin de la conexión de una sesión – transferencia de datos estándar y acelerada – control del diálogo – sincronización de la conexión de la sesión – reparación de errores
4. Transporte (transport layer)	ISO Transport Class Realización por software	Funciones: – inicio de las conexiones del transporte – transferencia de datos – desconexión del transporte Servicios: – inicialización – transferencia de datos – fin de los servicios de transporte

3. Red (network layer)	ISO CLNS Realización por software	Funciones: – establecimiento de rutas y rutas alternativas – conexiones de red y multiplexaje – segmentación y división por bloques – reconocimiento y corrección de errores – control de secuencia y flujo – transferencia de datos a alta velocidad – reinicio de conexiones de la red – selección de servicios – manejo del nivel de red Servicios: – conexiones y direccionamiento de la red – identificación y conexión de puntos terminales – transferencia de las unidades de datos del servicio de la red – parámetros de la calidad del servicio – reporte de errores – control de flujo y secuencia de datos – transferencia de datos a alta velocidad – reinicialización – desconexión de la red
2. Ligado de datos (data link layer)	IEEE 802.2 LLC IEEE 802.4 Token Bus ISO 8802.4 Token Bus Realización en hardware (MAC), software (LLC) y firmware	Funciones: – inicio y fin de la liga de transmisión – sincronización y cuadratura de mensajes – control de flujo y secuencia – reconocimiento y corrección de errores – intercambio de parámetros e identificaciones – monitoreo de la conexión física – manejo del nivel de ligado de datos Servicios: – sección de transmisión – ligado de las unidades de datos del servicio – conexión del identificador de punto terminal en el ligado de datos – control de flujo y secuencia – reporte de errores – parámetros de la calidad del servicio
1. Físico (physical layer)	IEEE 802.4 a 10 Mbits/s Broadband/ Carrier band Realización por hardware	Funciones: – activación y desactivación de la conexión física – transferencia de bits – manejo del nivel de transferencia de bits Servicios: – conexiones físicas – unidades de datos del servicio de conexión física – puntos terminales de la conexión física – reconocimiento del nivel de conexión – control de secuencia – indicación de errores – parámetros de la calidad del servicio

El problema de conexión de dispositivos inteligentes de adquisición de datos con un sistema abierto está ubicado en el nivel 2 del modelo OSI. El ligado de datos debe realizarse para lograr la comunicación entre el nivel físico y el de red, mediante el manejo de las unidades de datos: cuadratura, corrección de errores, establecimiento de la comunicación, control de flujo y demás funciones y servicios descritos en la tabla.

Con respecto al modelo piramidal de ISA, el problema citado corresponde a la liga de datos entre los niveles de proceso y unidad de control, es decir, a la comunicación entre los dispositivos inteligentes de adquisición de datos de campo y la unidad controladora del proceso. Debido a que el objetivo de esta tesis es desarrollar un sistema que permita la comunicación entre dos niveles de un modelo de control, al sistema se le denomina *junction box* para referencia rápida, pues es una caja negra que interconecta o une dos niveles de control (junction box: caja de unión). Se pide una disculpa a los puristas por haberlo bautizado en inglés.

2.3 UBICACIÓN DEL JUNCTION BOX EN UN SCD

La concepción del junction box se debe a que en el Instituto de Investigaciones Eléctricas (IIE) en Cuernavaca se lleva a cabo el desarrollo de un sistema para el nivel de control llamado CAM (Controlador de Acceso Múltiple). El CAM es un sistema digital para el nivel de control basado en un microprocesador Intel 80386. El CAM es un sistema abierto con gran poder de cómputo que maneja buses de campo y redes de supervisión, y cuenta con interfaces para Ethernet, Token Bus y RS-232. Ya que el CAM sólo tiene una interfaz para los buses de campo es imposible conectarle diferentes tipos de dispositivos de adquisición de datos –figura 2.3 a)–. Se desea que el CAM maneje su propio protocolo de comunicaciones, por eso fue necesario desarrollar una tarjeta que le permitiera conectar cualquier red de los dispositivos de adquisición de datos y adecuarlos al protocolo propio, manejado en una red Token Bus –figura 2.3 b)–. El junction box debía ser un subsistema de CAM.

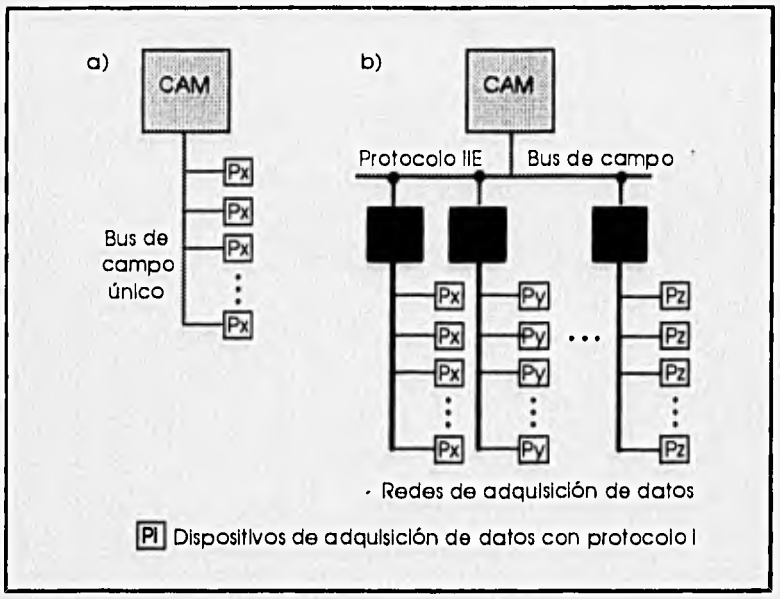


Fig. 2.3 CAM y Junction Box

En la figura 2.4 se da una representación gráfica de la trayectoria de la información en una planta, conceptualizando a los dispositivos inteligentes de adquisición de campo, al junction box, al CAM y a un tablero de supervisión como cajas negras o entes. En ella se observa que el junction box recibe las señales proporcionadas por dispositivos digitales ubicados en el nivel de proceso y las adapta para transmitirlas al controlador de los procesos de producción (CAM), que realiza las acciones de control (nivel de control) y se conecta al nivel de supervisión de la planta. Tomando como base al modelo piramidal, se puede definir al junction box como un sistema que permite el enlace entre los niveles de proceso y de control.

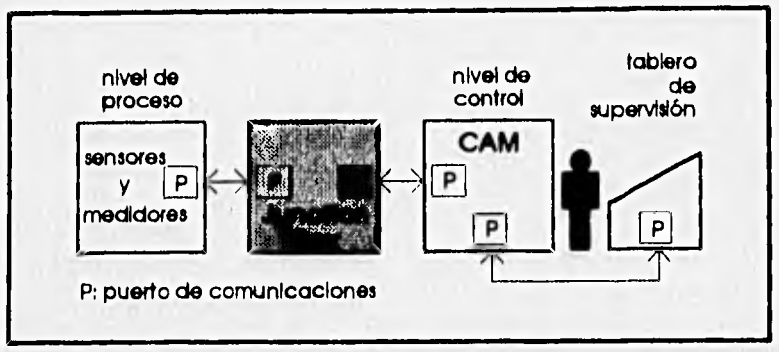


Fig. 2.4 Junction Box

Con base al modelo OSI, el junction box se concibió como un sistema que satisfaga los requerimientos del nivel de ligado de datos (data link layer), expuestos con anterioridad en la tabla de síntesis de dicho modelo.

2.4 CARACTERÍSTICAS DE LAS REDES DE COMUNICACIÓN Y SU RELACIÓN CON EL MODELO PIRAMIDAL DE CONTROL

El junction box necesita dos puertos de comunicación para el ligado de datos. La ISO ha sugerido utilizar Token Bus para la transmisión de datos en el nivel de control y Broadband/Carrier band para el nivel de proceso. La justificación de dicha propuesta se presenta a continuación.

2.4.1 Token Bus IEEE 802.4

Las características funcionales de las redes de comunicaciones varían para cada nivel de control definido por el modelo piramidal de ISA. Para el nivel de supervisión se busca maximizar el ancho de banda y minimizar el tiempo de acceso, sacrificando eficiencia cuando se eleva el tráfico de datos. En los niveles inferiores (de control y procesamiento), un incremento en el tráfico de datos indica que existe una condición anormal en el proceso que necesita ser controlada, por ello es necesario asegurar alta eficiencia en el flujo de datos para poder ejecutar la acción de control apropiada. En las redes instaladas en los niveles inferiores –llamadas *buses de campo*– se busca lograr un tiempo de respuesta uniforme ante cualquier tráfico de datos. Dadas las diferencias referentes al tráfico de datos entre niveles de control, es lógico

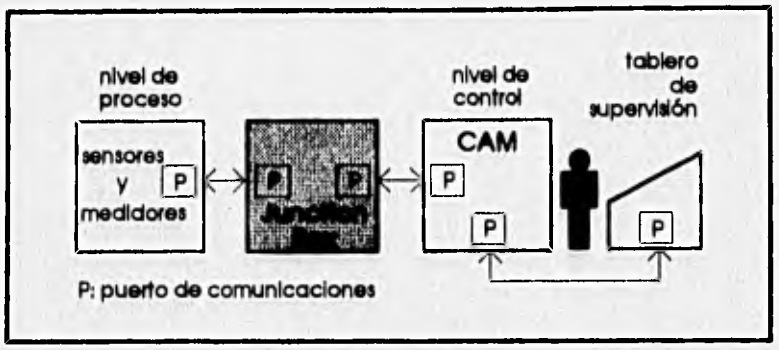


Fig. 2.4 Junction Box

Con base al modelo OSI, el junction box se concibió como un sistema que satisfaga los requerimientos del nivel de ligada de datos (data link layer), expuestos con anterioridad en la tabla de síntesis de dicho modelo.

2.4 CARACTERÍSTICAS DE LAS REDES DE COMUNICACIÓN Y SU RELACIÓN CON EL MODELO PIRAMIDAL DE CONTROL

El junction box necesita dos puertos de comunicación para el ligado de datos. La ISO ha sugerido utilizar Token Bus para la transmisión de datos en el nivel de control y Broadband/Carrier band para el nivel de proceso. La justificación de dicha propuesta se presenta a continuación.

2.4.1 Token Bus IEEE 8O2.4

Las características funcionales de las redes de comunicaciones varían para cada nivel de control definido por el modelo piramidal de ISA. Para el nivel de supervisión se busca maximizar el ancho de banda y minimizar el tiempo de acceso, sacrificando eficiencia cuando se eleva el tráfico de datos. En los niveles inferiores (de control y procesamiento), un incremento en el tráfico de datos indica que existe una condición anormal en el proceso que necesita ser controlada, por ello es necesario asegurar alta eficiencia en el flujo de datos para poder ejecutar la acción de control apropiada. En las redes instaladas en los niveles inferiores –llamadas *buses de campo*– se busca lograr un tiempo de respuesta uniforme ante cualquier tráfico de datos. Dadas las diferencias referentes al tráfico de datos entre niveles de control, es lógico

pensar que se utilizan redes de protocolos diferentes para cada uno, utilizando el que mejor se adapte a sus particularidades para lograr la máxima eficiencia en la operación.

Los parámetros más importantes para medir la eficiencia de una red son el tiempo medio y el *tiempo máximo garantizado* para transmitir un bloque de información. Este último cobra especial importancia en los procesos de automatización, pues los procesos se manejan en tiempo real y existe un límite del tiempo para reportar el estado del proceso al controlador llamado frontera superior determinística. En la mayoría de las redes de cómputo el tiempo máximo garantizado es estadístico –no es constante–, lo que impide su utilización como buses de campo.

Dado que la variable principal es el tráfico de datos, el tiempo de respuesta depende primordialmente del tipo de acceso a la red. Existen varios tipos de acceso, pero tres han ganado especial preferencia en las aplicaciones actuales: Acceso Múltiple con Sensado de Portadora y Detección de Colisiones (Carrier Sense Multiple Access with Collision Detection, CSMA/CD) o Ethernet, Token Bus y Token Ring.

En una red Token Ring una falla en un solo nodo acarrea la falla de la totalidad de la red a menos que exista un doble anillo. Puesto que se desea alta confiabilidad en el flujo de datos de la red, la característica anterior descarta a Token Ring como configuración para aplicaciones de control. Las redes de tipo Token Ring se utilizan principalmente para ambientes de oficina.

Las configuraciones restantes son CSMA/CD y Token Bus. En las siguientes gráficas (figura 2.5) se presenta una comparación de eficiencias entre una red tipo CSMA/CD (Ethernet) y Token Bus.

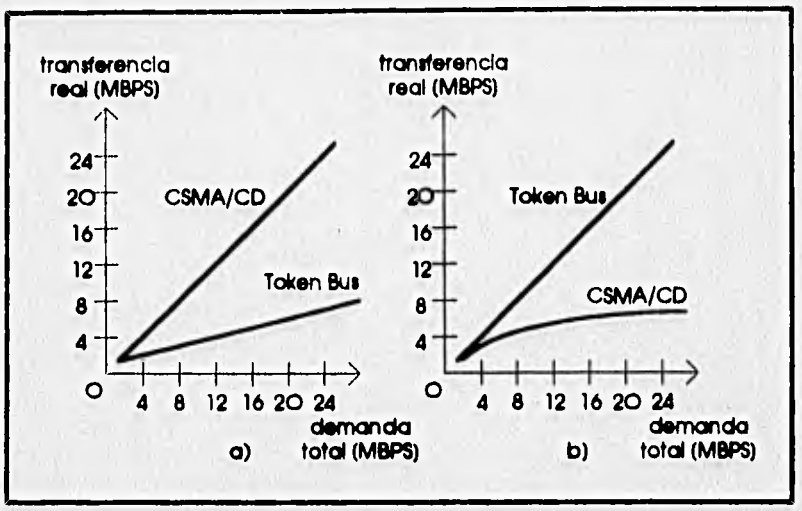


Fig. 2.5 Máxima capacidad potencial para LANs CSMA/CD y Token Bus (a) una estación activa de 100 (b) 100 estaciones activas de 100

Las gráficas consideran un flujo de datos de paquetes de 2000 bits.

En las gráficas se observa que cuando el tráfico de datos es bajo –gráfica (a)–, una red CSMA/CD tiene alta eficiencia debido a que la frecuencia de colisiones es baja. Por el contrario, la red Token Bus gasta mucho tiempo en el paso del *token* (el *token* es una seña o "ficha" que permite transmitir, es un campo dentro del cuadro o *frame* de transmisión) entre estaciones que no tienen algo que transmitir y su eficiencia disminuye. Cuando el tráfico en la red aumenta –gráfica (b)– la eficiencia de la red Token Bus aumenta porque aumenta el número de estaciones que utilizan el *token*, mientras que en la red CSMA/CD la eficiencia disminuye por el aumento en la frecuencia de las colisiones.

En la comunicación control-proceso, se requiere una red que maneje grandes cantidades de información, que dicha información sea accesible en un intervalo de tiempo definido, y que tenga alta eficiencia cuando los procesos son automatizados.

La red CSMA/CD tiene el problema de que el tiempo promedio de transmisión es no determinístico. La probabilidad de que se presente una colisión en la red es

$$P(I)=\sum_{n=1}^{\infty}(1-A)^{n}A,$$

donde I es el número de intervalos de tiempo en los que se puede transmitir y A es la probabilidad de que una sola terminal desee transmitir y es diferente de cero. La

expresión anterior es una serie geométrica convergente pues $|(1-A)|<1$ para $0<A\leq 1$ –que son los valores que puede tomar A–. La suma de la serie es $\frac{A}{1-(1-A)}=1$ y quiere decir que para cualquier valor de I siempre hay probabilidad de colisiones, por lo que es imposible obtener un tiempo máximo de transmisión. En consecuencia, una red de tipo CSMA/CD o Ethernet se descarta para aplicaciones de control.

La red Token Bus, a pesar de tener un tiempo promedio muy malo, tiene gran eficiencia cuando el tráfico de datos es alto. Además, el tiempo máximo de respuesta garantizado tiene un valor finito, por lo que soporta aplicaciones de automatización de procesos en tiempo real. Sea Tt el tiempo requerido para pasar el *token* entre estaciones, Tp el tiempo requerido para transmitir un paquete de datos, P el número máximo permitido de paquetes por transmitir para cada estación y N el número de estaciones en la red. Todos estos parámetros son constantes. El tiempo máximo garantizado es

$$Tm=(PTp+Tt)N,$$

y Tm es un parámetro determinístico pues depende de constantes.

Desde hace diez años a la fecha, Instrument Society of America (ISA) está trabajando, en conjunto con compañías dedicadas al control, a la creación de un estándar para la comunicación entre los niveles de proceso y de control. Este estándar, llamado SP–50 Field Bus (Bus de Campo), está basado en el estándar Token Bus IEEE 802.4 y se cree que tendrá gran aceptación.

En resumen, para el sistema desarrollado en esta tesis se eligió a la red Token Bus como protocolo para la comunicación con el nivel de unidad de control del modelo piramidal de ISA, debido a su alta eficiencia en el manejo de datos, su confiabilidad ante el tráfico de datos elevado y por ser una red cuyo tiempo de respuesta garantizado es determinístico –lo que le permite soportar procesos de automatización–. Otra razón para escoger Token Bus es facilitar la actualización del sistema cuando se lance el estándar SP–50.

2.4.2 RS–485

La mayoría de los dispositivos de control digitales cuentan con interfaces para comunicación serial. Los protocolos más conocidos son los protocolos RS–232, RS–422–A y RS–485. El protocolo RS–485 es una versión mejorada de RS–422–A, que permite la transmisión de información digital hacia varios componentes del sistema y periféricos en forma balanceada a través de líneas de distribución de bastante longitud,

mediante un arreglo de transmisores y receptores en un solo cable del tipo par trenzado. El protocolo RS-485, a diferencia de RS-422-A y RS-232, permite una comunicación multipunto por medio de un solo par de cables; esto lo hace muy deseable en aplicaciones donde se desea recolectar datos desde diversos puntos sin gastar demasiado dinero en el cableado y en el control de la red. En la figura 2.6 se muestra un esquema del tipo de red que se obtiene al utilizar el estándar RS-485.

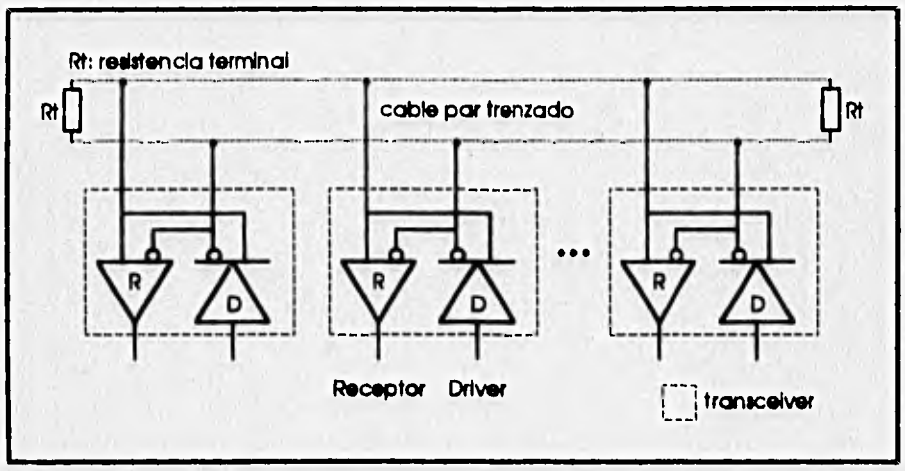


Fig. 2.6 RS-485 en aplicación multipunto

Las características del protocolo RS-485 son:

- Un transmisor puede manejar hasta 32 unidades de carga y una resistencia terminal de la línea de 60Ω o más. Una unidad de carga está definida como la carga que exige 1 mA de corriente ante un voltaje en modo común de 12V como máximo.
- El valor pico de la corriente de salida para el estado lógico bajo (off-state), no debe exceder 100 μA ante cualquier voltaje en el rango de -7V a 7V.
- El transmisor debe ser capaz de proveer un voltaje diferencial de 1.5V a 5V con voltajes de modo común entre -7V y 12V.
- El transmisor debe tener protección interna contra contenciones de bus, es decir, no debe sufrir daño si sus salidas están conectadas a una fuente de voltaje cuyo valor esté entre -7V y 12V mientras su estado de salida es un nivel lógico alto, bajo o pasivo.
- La impedancia de entrada de los receptores debe ser de 12 kΩ mínimo.

- El rango de entrada en modo común para un receptor debe oscilar entre -7V y 12V.
- La sensibilidad ante entradas diferenciales debe ser de 200 mV en un rango de -7V a 12V en modo común.

BIBLIOGRAFÍA DE REFERENCIA

Pedro A. Molina, Pablo H. Ibargüengoitia, Jorge Hermosillo
An Integral Approach for the Field, Control and Supervisory Levels of a SCD
Documento para la Industrial Computing Society Conference ISC/93
México, 1993

Gilbert Held
Understanding Data Communications
John Wiley & Sons Ltd.
Baffins, Chichester, England 1991

Fred Halsall
Data Communications, Computer Networks and Open Systems
Addison Wesley Publishing Company
USA, 1992

3. DESCRIPCIÓN GENERAL DEL SISTEMA

3.1 METODOLOGÍA DE DISEÑO

El junction box es una tarjeta basada en microprocesador porque debe ser capaz de transmitir y recibir datos en forma digital –desde y hacia sistemas basados en microprocesadores– y desempeñar funciones elementales de control basadas en algoritmos. Debido a que se proporcionan las condiciones de diseño, las características deseadas y el entorno del sistema, se utiliza una metodología de diseño funcional descendente o *top-down*. Esta metodología consiste en dividir el sistema total en subsistemas y desarrollarlos por separado, disminuyendo su complejidad inicial; para culminar con la integración de todos los módulos en un sistema total. La metodología a seguir se describirá en seguida.

DISEÑO DEL HARDWARE	DISEÑO DEL SOFTWARE
Especificación del hardware	Especificación del software
Partición del sistema en módulos funcionales	Partición en bloques o rutinas
Selección de la familia de componentes	Selección del lenguaje
Diagrama a bloques de cada módulo	Descomposición del sistema en rutinas funcionales
Desarrollo de cada módulo en detalle	Desarrollo de cada algoritmo en detalle
Alambrado y prueba de cada bloque	Codificación y prueba de cada rutina
Conjunción de bloques en cada módulo	Conjunción de rutinas para cada módulo
Conjunción de módulos en un sistema final	Conjunción de subsistemas en un sistema final
Prueba del sistema	

Las tareas especificadas de hardware y de software se realizan preferentemente en paralelo, aunque a veces es necesario adelantar el desarrollo de alguna de ellas.

3.2 PRINCIPIOS DE DISEÑO

Durante el diseño se cuidaron cuatro principios fundamentales

1. Modularidad: consiste en vigilar que el sistema total esté formado por módulos y pueda ser, en algún momento, subsistema de otro sistema más general.
2. Regularidad: consiste en establecer semejanzas entre los módulos del sistema o de éste con otros sistemas.
3. Localidad: consiste en vigilar la ubicación espacial de cada módulo del sistema y de éste en su ambiente de operación. Una especificación del diseño fue lograr un sistema de pequeñas dimensiones.
4. Conectividad: consiste en vigilar la forma de comunicar los módulos del sistema y de éste con otros sistemas.

3.3 ESPECIFICACIÓN DEL DISEÑO

Las características globales de los sistemas para el nivel de proceso son las siguientes[1]:

- memoria RAM y ROM para asegurar máxima confiabilidad
- procesamiento en paralelo
- capacidad del hardware para hacer rápidamente cambios de programación
- diseño de hardware del tipo de sistemas basados en microprocesador
- protección contra errores

Y las tareas que deben realizarse en dicho nivel son:

- funciones lógicas y de protección
- coordinación de algunas tareas básicas de control
- comunicación con terminales remotas

Las especificaciones de diseño estipularon los siguientes requerimientos:

- un módulo de comunicación con Token Bus

[1] Fred Halsall
Data Communications, Computer Networks and Open Systems
Addison Wesley Publishing Company
USA, 1992

- un módulo de comunicación con dispositivos inteligentes de adquisición de datos que utilizan el estándar RS-485
- una memoria ROM que pudiera ser borrada eléctricamente para poder actualizar programas operativos vía red
- *watchdog timer* (temporizador 'guardián') para permitir la operación continua del sistema (protección contra errores)
- capacidad de procesar la información adquirida mediante cualquier módulo de comunicaciones para reconocer protocolos de comunicación, actuar en presencia de alarmas y corregir errores en las comunicaciones

3.4 DIAGRAMA DE BLOQUES DEL JUNCTION BOX

Las especificaciones anteriormente descritas permiten esbozar el diagrama de bloques mostrado en la figura 3.1.

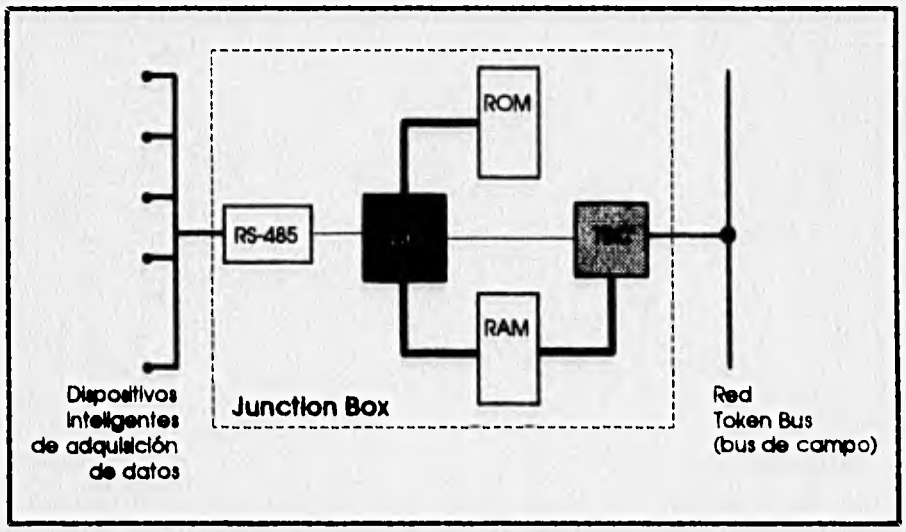


Fig. 3.1 Diagrama de bloques del junction box

3.5 CARACTERÍSTICAS GENERALES DEL JUNCTION BOX

EL junction box tiene un microprocesador para poder recibir, clasificar, procesar y transmitir información obtenida de los niveles de control y proceso, con objeto de permitir la liga de datos entre dichos niveles y además descargar trabajo al nivel de control proporcionándole solamente información útil. Sin embargo, este manejo de la información no debe menguar las capacidades de comunicaciones del sistema. Por ello se utiliza una arquitectura multiprocesador-multiprocesamiento, es decir, procesamiento en paralelo.

El multiprocesamiento consiste en utilizar varios procesadores que trabajan cooperativamente. Se pueden traslapar cualquier tipo de operaciones en cualquier instante porque los procesadores ejecutan diferentes tareas al mismo tiempo. Los procesadores comparten memoria y periféricos, controlados por un mismo sistema operativo.

La técnica de paralelismo se refiere a la ejecución simultánea, ya sea en tiempo o espacio, de tareas en el sistema. Existen tres tipos de paralelismo:

1. Por procesador. Se utilizan varios procesadores que ejecutan la misma tarea al mismo tiempo. Se utiliza para sistemas que requieren alta confiabilidad y en el diseño de computadoras tolerantes a fallas. La simultaneidad es espacial.
2. Por instrucción. Se traslapan dos instrucciones que se ejecutan en diferentes lugares del procesador. El procesamiento es simultáneo en el tiempo. Se utiliza en mainframes y sistemas de redes. Se le conoce también como *pipeline*.
3. Aritmético. Se utilizan varias ALUs simultáneamente para evitar lo más posible iteraciones. Se aplica en sistemas orientados a procesar información o hacer cálculos con velocidad, como aplicaciones militares y satélites.

Se quizo que el pracesador principal para el junction box fuera Intel, para mantener regularidad con el CAM y poder utilizar las herramientas de desarrollo existentes. El módulo de Token Bus exigió un procesador que manejara totalmente el medio de acceso a la red y el ligado lógico de los datos. La selección se orientó a Motorola, pues ha desarrollado microprocesadores para redes Token Ring y Token Bus. El módulo de comunicaciones para adquisición de datos solicitaría un chip para manejar el protocolo RS-485.

El *junction box* posee paralelismo por procesador pues coexisten dos de ellos compartiendo memoria, y por instrucción por la naturaleza de los procesadores, pues Intel maneja paralelismo por instrucción en la familia 80x86.

3.6 CLASIFICACIÓN DEL SISTEMA

Existen cuatro clasificaciones de Flynn según la multiplicidad de flujo de datos (de entrada y resultados parciales o totales) o de flujo de instrucciones para los sistemas digitales:

1. SISD: un solo flujo de datos y uno solo de instrucciones.
2. SIMD: múltiple flujo de datos y un solo flujo de instrucciones.
3. MISD: un solo flujo de datos y multiplicidad de instrucciones. Esta clasificación es ideal; querría decir que con solo unos datos se podría procesar mucha información.
4. MIMD: múltiple flujo de datos e instrucciones. Se puede asociar con los sistemas de multiprocesamiento.

El *junction box* es, según la clasificación de Flynn, un sistema digital tipo SIMD cuyo esquema se muestra en la figura 3.2.

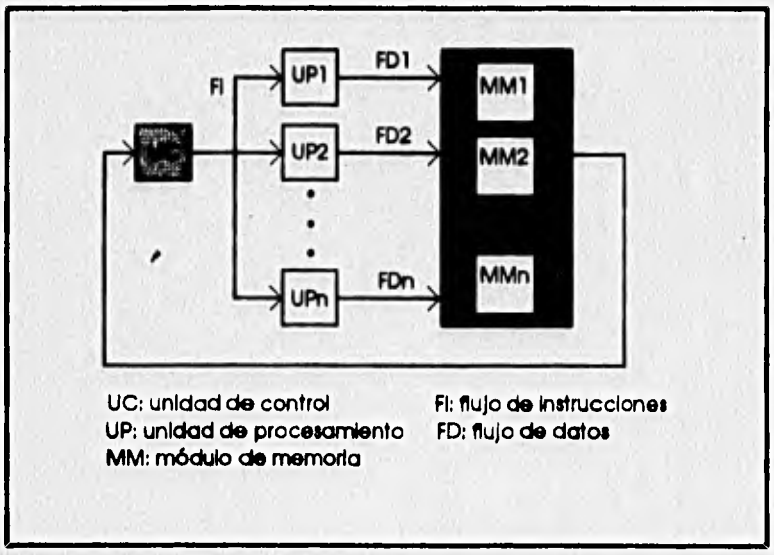


Fig. 3.2 Sistema digital tipo SIMD

4. DISEÑO DEL HARDWARE

El desarrollo del hardware y del software para el junction box fue desarrollado en muchas ocasiones en forma paralela, sin embargo su descripción se hace independientemente con objeto de no complicar la descripción del sistema y desarrollarlo en forma metódica.

4.1 SELECCIÓN DE COMPONENTES

Para seleccionar los componentes de un sistema es necesario tener en mente un diseño preliminar, basado en el diagrama de bloques obtenido a partir de las especificaciones (fig. 3.1). La enumeración siguiente se hace en base a dicho diagrama.

□ Coprocesadores

El primer componente seleccionado fue el MC68824 Token Bus Controller de Motorola –TBC–, un chip que maneja el control del medio de acceso o MAC (Media Access Control) para el estándar IEEE 802.4 de estaciones para LANs, y la porción del control del ligado lógico o LLC (Logical Link Control) para el estándar IEEE 802.2 en receptores, en forma completa para el tipo 3 (LLC type 3) y como soporte para los tipos 1 y 2. ¿Qué significa esto?

El estándar IEEE 802 define los protocolos estándar para redes de área local o LANs (Local Area Networks) en lo que respecta a las capas física y de ligado de datos del modelo OSI (physical and data link layers). Existen dos porciones de este estándar:

1. MAC. Se refiere a los medios físicos de obtención de datos entre los que destacan los documentos
   - IEEE 802.3 CSMA/CD bus, que es el protocolo de Ethernet
   - IEEE 802.4 Token Bus
   - IEEE 802.5 Token Ring

La principal función de esta subcapa es realizar la detección de errores y el encuadramiento de mensajes (framing) que corresponden a la capa de ligado de datos (data link layer); y a manejar en su totalidad la capa física (physical layer) del modelo OSI o especificación MAP[1].

[1] Ver apéndice A

2. LLC. Se refiere al ligado de datos que otorga servicios al usuario. Su documento correspondiente es el IEEE 802.2 y realiza las demás funciones descritas en la capa de ligado de datos (data link layer) del modelo OSI. Existen tres subcapas de LLC que son

- LLC type 3, protocolo de sin-conexión (connectionless)
- LLC type 2, protocolo orientado a la conexión (connection-oriented)
- LLC type 1, protocolo orientado a la conexión (connection-oriented)

En la práctica se utiliza el tipo 3 de LLC que define un protocolo resumido como SDN (Send-Data-with-No-acknowledge)[2].

El MC68824 está disponible en el mercado en tres versiones según su frecuencia de operación: 10, 12.5 y 16.67 MHz. Ya que se requería una velocidad de transmisión de 3 Mbits/s –velocidad de transmisión en el bus de campo del CAM– se escogió a una frecuencia de 10 MHz, es decir el MC68824-10.

El TBC opera de dos formas: como esclavo y como maestro. Cuando es accesado como esclavo sus 8 registros internos representan un espacio de puertos para el procesador principal (host processor). Cuando es maestro puede solicitar el bus y tomarlo para utilizar la memoria RAM, trabajando como otro procesador en el sistema.

En la figura 4.1 se muestra al TBC con sus señales agrupadas según su funcionalidad.

[2]Ver apéndice A

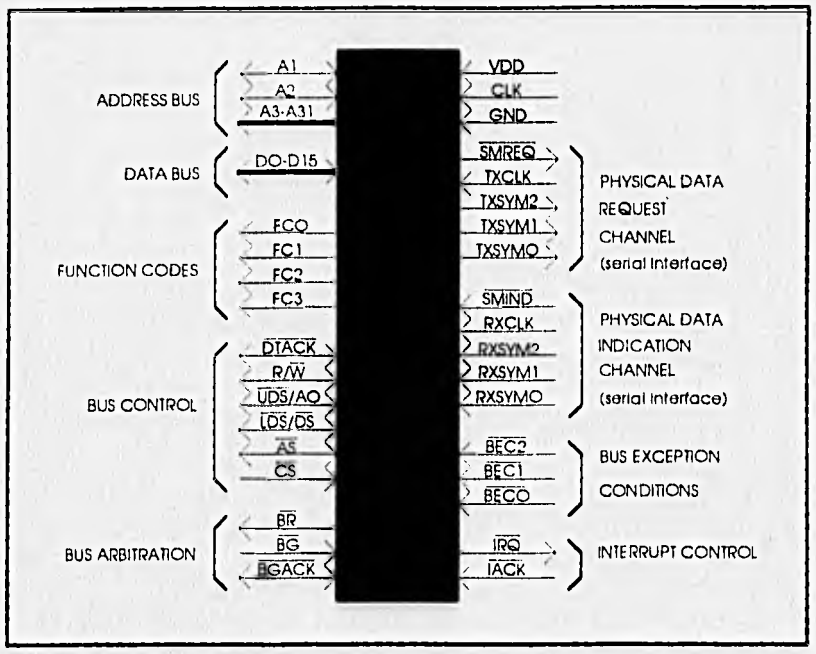


Fig. 4.1 Señales del TBC según su funcionalidad

□ Microprocesador principal o "anfitrión" (host processor)

Los principales aspectos para elegir el microprocesador se presentan en seguida, sin orden jerárquico.

- Frecuencia de operación. La frecuencia de operación debía ser mayor que la máxima velocidad de transmisión de las señales de comunicación. Con respecto a las señales de RS-485 –red de adquisición de datos–, las máximas velocidades son de 19.2 y 38.4 kbauds que pueden lograrse fácilmente con frecuencias mayores a 1 MHz. Con respecto a Token Bus la velocidad de transmisión requerida fue de 3Mbits/s. Por lo tanto, el microprocesador debía trabajar a 10MHz o más para aprovechar toda la capacidad del TBC.

- Memoria. El TBC (Token Bus Controller) requiere un espacio de memoria RAM para su funcionamiento de 1280 bytes mínimo, pues necesita una tabla de inicialización y un área privada, ambas escritas en RAM. Además debe reservarse memoria

suficiente para la transmisión y recepción de mensajes (memoria de usuario). Los estándares del IEEE especifican que un *frame* de datos no debe exceder de 8 kbytes, aunque el TBC puede transmitir *frames* de hasta 64 kbytes.

Muy probablemente una memoria RAM de 64 kbytes hubiera sido suficiente, pero recordando que se trata de un primer diseño se decidió adquirir una memoria RAM de 128 kbytes. La memoria elegida fue la HM628128 de Hitachi.

La cantidad de memoria ROM pudo estimarse gracias a que existe un programa de aplicación para TBC, donde debían reservarse mínimo 64 kbytes. Además se necesitaba más memoria para el programa monitor –del que se hablará posteriormente– que debe estar grabado en ROM. Recordando las especificaciones de diseño, se deseaba una memoria que pudiera ser borrada y reprogramada sin necesidad de ir al campo[3], por lo que se eligió la Am29FO1O, una memoria Flash de 128 kbytes con las siguientes características:

- arquitectura de borrado por sectores: posee 8 sectores de 16kbytes cada uno, que pueden ser borrados en cualquier combinación concurrentemente. También soporta el borrado completo del chip
- borrado y preprogramación del chip o sus sectores mediante un algoritmo llamado Embedded Erase™
- programación y verificación de los datos mediante un algoritmo llamado Embedded Program™
- protección de sectores: para cualquier combinación de ellos

Se determinó que se necesitaba un microprocesador que pudiera direccionar por lo menos 256 kbytes de memoria. El espacio para puertos es un parámetro no crítico pues el TBC –cuando es esclavo– ocupa sólo 8 localidades y no se requieren periféricos.

- Buses. El TBC puede trabajar en ambientes con buses de datos de 8 y 16 bits. Lo ideal sería elegir un microprocesador de Motorola de la familia 68000 para poder realizar la conexión de las señales de control de la manera más fácil, trabajar con buses de 16 bits y aprovechar toda la velocidad del chip. Por razones de espacio, se eligió un procesador de 8 bits en el bus de datos, pues al usar 16 bits era necesario duplicar el número de memorias –se necesita una memoria para las direcciones pares y otra para las direcciones impares–. Al trabajar con el TBC en un ambiente de 8 bits se pierde velocidad en el sistema únicamente en la escritura y lectura hacia la

[3] Capítulo 3, sección 3. *Especificación del diseño*

RAM, sin perder eficiencia en el procesamiento y recepción de datos; así que el sacrificio no es muy grave. Las mejores opciones para microprocesadores de 8 bits son de Intel y Zilog, pero en el IIE el software de soporte y el ambiente de desarrollo ha sido todo desarrollado para Intel. Es evidente que conservar la regularidad de componentes era muy recomendable.

- Periféricos. Para el junction box era necesaria una unidad de comunicación serial –UART– para manejar la red de adquisición de datos. Las velocidades típicas de transmisión son de 9600, 19200 y 32400 bauds. Debido a que la función principal del junction box es la transmisión y recepción exhaustiva de datos, se decidió que convenía utilizar un controlador de DMA para liberar al software de la labor de recepción y aumentar la velocidad en las comunicaciones, pues no se pierde tiempo de procesamiento revisando si existen recepciones pendientes ("poleo" del puerto serie). El controlador de DMA y el TBC provocan interrupciones al terminar sus procesos, por lo cual era necesario añadir un controlador de interrupciones.

En base a los requerimientos citados, el microprocesador seleccionado fue el 80C188EC de Intel, un procesador revestido (embedded processor) que tiene integrado lo siguiente

- un CPU 80188
- cuatro canales de DMA
- dos puertos serie (UARTs) soportados por DMA, con generador de velocidad de transmisión *(baud rate)*
- 24 pines de puertos con entrada/salida multiplexada
- dos controladores de interrupción compatibles con el 8259A
- tres contadores o timers de 16 bits programables
- watchdog timer de 32 bits
- 10 chip selects programables con generador de estados de espera
- unidad de refresco de memoria para DRAM
- unidad de control de potencia disipada
- oscilador controlado por cristal

El procesador es un 80188 que tiene un bus externo de 8 bits y uno interno de 16 bits, así que la velocidad de procesamiento es igual a la de un 80186 excepto al accesar memoria o puertos. Este procesador puede procesar valores numéricos hasta 65535 en una sola instrucción, puede direccionar 1 Mbyte de espacio de memoria y 64

kbytes de espacio para puertos; y manejar hasta 256 interrupciones.

Los periféricos se incluyeron al elegir un procesador revestido. El 80C188EC fue una buena selección pues contiene todos los que se especificaron y permite ahorrar espacio en la tarjeta al reducir la cantidad de componentes externos. De este modo se pudo ahorrar el diseño de la lógica de decodificación del mapa de memoria, el alambrado de puertos serie y paralelo, y de los controladores de interrupción y de DMA. Además, tiene incluida la unidad de watchdog que asegura la operación ininterrumpida del sistema.

Todos los periféricos se manejan mediante registros programables por software. Dichos registros están contenidos en un bloque llamado PCB (Peripheral Control Block) que puede estar ubicado en espacio de memoria o puertos. Por defecto el PCB está ubicado en espacio de puertos en las direcciones FF00H-FFFFH.

En la figura 4.2 se muestra un diagrama de bloques del microprocesador 80C188EC.

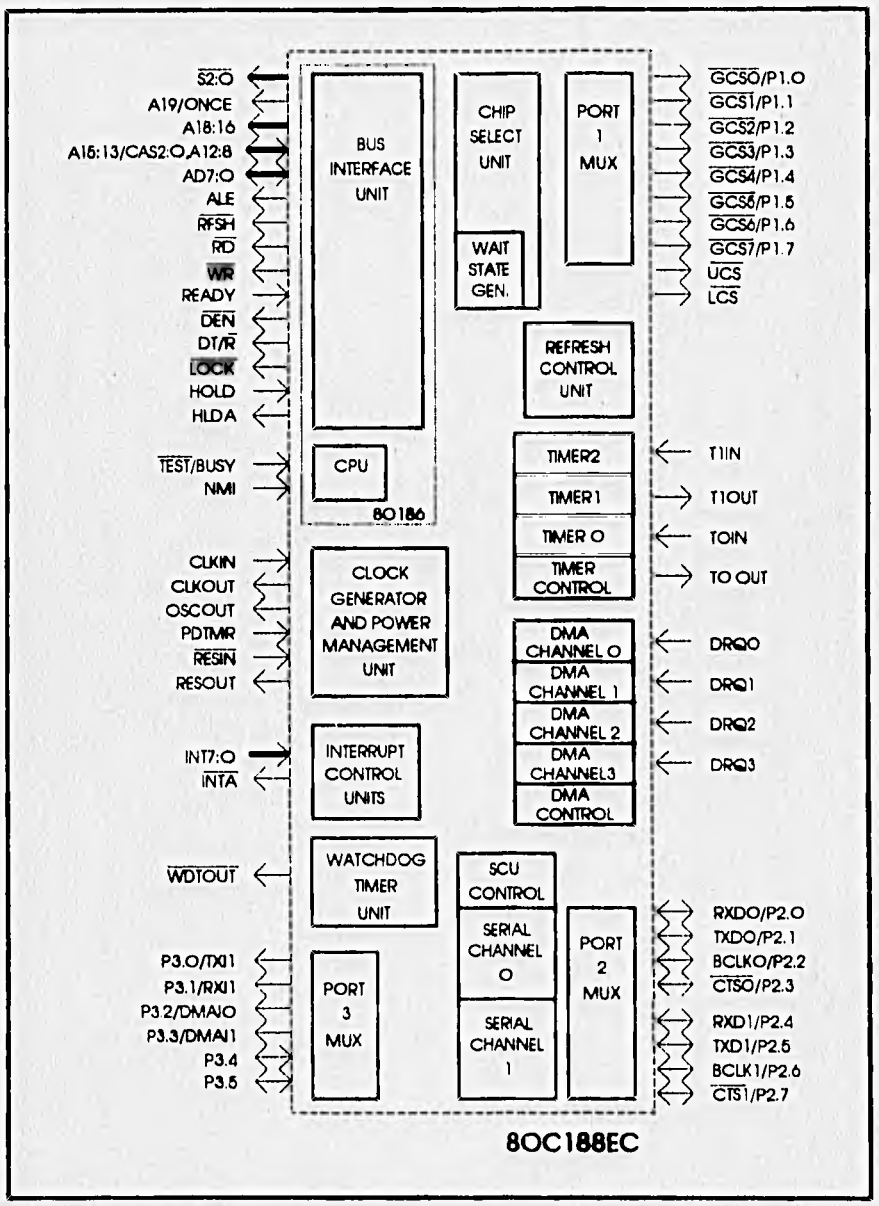


Fig. 4.2 Diagrama de bloques del 80C188EC

En el junction box, el 80C188EC funciona como procesador principal (host processor).

□ Requerimientos de entrada/salida

El último componente a definir fue un manejador o driver para las líneas de transmisión y recepción de la red de adquisición de datos, que utilizan el estándar RS-485 (según se mencionó en el capítulo 2). Se eligió el DS3696 de National, un pequeño chip que maneja totalmente las líneas de Tx y Rx, y posee además una salida para indicar fallas por sobrecalentamiento.

4.2 DISEÑO DE LOS MÓDULOS

El junction box fue dividido para su diseño en dos sistemas: el sistema 80C188EC (procesador principal) y el sistema TBC (coprocesador). Cada uno de ellos se dividió en los siguientes módulos

Para el sistema 80C188EC los módulos son:

- oscilador
- reset
- buses
- señales de control para las memorias
- señales de control para el acceso al TBC esclavo

Y para el sistema TBC son:

- oscilador
- reset
- buses
- señales de control para la memoria RAM

En seguida se describirán los módulos para el 80C188EC y posteriormente los del TBC.

4.2.1 Oscilador del 80C188EC

El 80C188EC puede operar a dos frecuencias: 13 y 16 MHz. Se eligió la frecuencia más baja (13 MHz) pues es suficiente velocidad para la aplicación y así se evitan problemas por ruido que suelen presentarse al trabajar a frecuencias mayores.

No hay que olvidar, de ningún modo, que ésta es la primera versión del junction box.

El 80C188EC soporta una conexión de cristal para operar, o bien la entrada de un oscilador al doble de la frecuencia de operación. Se utilizó un oscilador de 26 MHz CMOS.

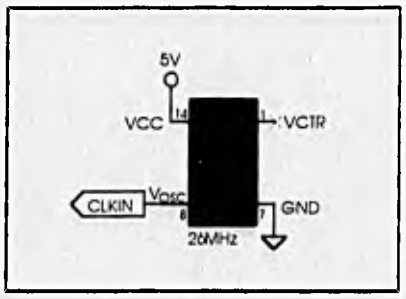


Fig. 4.3 Oscilador

4.2.2 Reset del 80C188EC

Existen tres casos de reset:

1) al encender la fuente (cold reset)
2) el reset manual por medio de un interruptor cuando el procesador está activo (warm reset)
3) el reset por watchdog

La señal de reset para el µP es activa baja, y se denotará como /RESIN.

1) En un cold reset se debe tener la precaución de que el pulso de reset dure lo suficiente en activo bajo para que el µP lo reconozca, después de que la fuente ha llegado a tomar un voltaje válido. Por ello, es necesario esperar que la fuente tenga una señal de 5V y que el voltaje de entrada a /RESIN sea menor o igual a 1V, es decir un cero lógico (las señales del µP son compatibles con TTL) durante 32 ciclos de reloj, de acuerdo con la hoja de especificaciones. La señal /RESIN soporta una entrada RC, por ello se utiliza un arreglo como el de la figura 4.4.

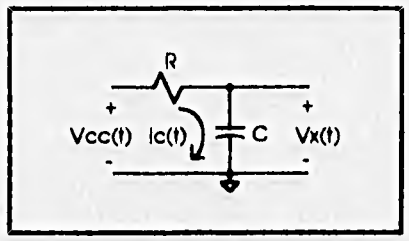


Fig. 4.4 Circuito RC

Se resolverá el circuito por medio de transformada de Laplace para obtener los valores de R y C.

La señal de entrada Vcc(t) es exponencial, con una constante de tiempo tan grande que se considera a Vcc(t) como un escalón. Entonces

$Vcc(t)=Vcc\,u(t)$

donde u(t) es la función escalón y Vcc es una constante, normalmente 5 V. Su transformada de Laplace es

$$\mathcal{L}\{Vcc(t)\}=Vcc(s)=\frac{Vcc}{s}$$

El voltaje de un capacitor es

$$Vx(t)=\int\frac{1}{C}Ic(t)dt$$

y su transformada de Laplace es

$$\mathcal{L}\{Vx(t)\}=Vx(s)=\frac{1}{sC}Ic(s) \quad ...(A)$$

Se resuelve la malla del circuito

$Vcc(s)=RIc(s)+Vx(s)$

La corriente Ic(s) se puede obtener de la ecuación (A). Sustituyendo en la ecuación anterior

$Vcc(s)=sRCVx(s)+Vx(s)=(sRC+1)Vx(s)$

de donde

$$Vx(s)=\frac{1}{sRC+1}Vcc(s)=\frac{Vcc}{s(sRC+1)}$$

la ecuación se ordena para calcular su transformada inversa

$$Vx(s)=\frac{\frac{Vcc}{RC}}{s(s+\frac{1}{RC})}$$

y se obtiene

$Vx(t)=Vcc(1-e^{\frac{-1}{RC}t})$

se despeja RC

$$RC = \frac{-t}{\ln(1-\frac{Vx(t)}{Vcc})} \quad ...(8)$$

Los 32 pulsos de reloj representan un tiempo tx=2.47µs. Por lo tanto, se desea que Vx(2.47µs)≤1V para asegurar que se cumple el temporizado de /RESIN. Sustituyendo a t=tx=2.47µs, a Vcc=5V y a Vx(t)=Vx(tx)=1V se obtiene

$RC=1.1x10^{-6}$

Si se utiliza un capacitor de 1µF, la resistencia debe ser de 1Ω. Estos son los valores justos para asegurar el retraso pero se debe considerar que los componentes tienen tolerancias y variaciones, así que se deben exagerar los valores de los componentes. En la práctica se trató de utilizar un capacitor de 1nF con una resistencia de 4.7kΩ, pero no se consideró que dicho capacitor no es electrolítico. Después se cambió el capacitor y los componentes utilizados fueron R=4.7kΩ y C=1µF. Con estos valores la constante de tiempo es RC=4.7ms, suficientemente grande para asegurar el reset.

Si la alimentación del circuito se suspende instantáneamente –al apagar la fuente o por faltas en el suministro de energía– es posible que la corriente almacenada en el capacitor dañe al microprocesador. Para prevenir esto se conecta un diodo entre el capacitor y la fuente en polaridad inversa.

2) En un warm reset la única restricción es que el pulso dure al menos cuatro ciclos de reloj. Esto se cumplirá siempre que una persona pulse el *push button* como pulsa normalmente las teclas de una calculadora o una computadora.

3) El watchdog es una señal proveniente de un contador interno de tipo "un-disparo" *(one-shot)*, que es reseteado cada vez que se ejecuta un ciclo de fetch o se recibe una señal de reconocimiento de datos desde memoria o puertos (p.e. /DTACK en Motorola). Si por alguna razón se pierde el control, sea por hardware o por software, el contador alcanza su cuenta máxima provocando un pulso de salida. Este pulso puede utilizarse para solicitar una interrupción o resetear el sistema.

El reset por watchdog se logra con una compuerta AND de la señal RC con la del pin /WDTOUT, y la salida se conecta a /RESIN. Ya que la señal RC toma valores de voltaje inválidos para la entrada de la compuerta –no es una señal digital–, es necesario utilizar una compuerta Schmitt Trigger, que son compuertas con histéresis para trabajar con señales analógicas.

La inclusión de un chip de cuatro ANDs Schmitt Trigger para utilizar sólo una

compuerta presenta la desventaja de ocupar espacio y aumentar los costos. El alambrado del reset por watchdog se pospuso hasta estudiar la posibilidad de usar las otras tres compuertas y obtener un beneficio adicional.

El circuito final de reset se muestra en la figura 4.5.

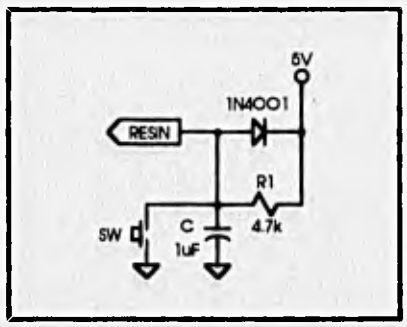


Fig. 4.5 Circuito de Reset

4.2.3 Buses de direcciones y datos del 80C188EC

Los buses de direcciones y datos del 80C188EC están multiplexados en las señales AD7-AD0, las señales CAS2-CAS0 con A15-A13 y las señales S6-S3 están multiplexadas con A19-A16. Para separarlas, se utilizan latchs de tres estados que retendrán el valor de las direcciones durante todo el ciclo de bus. Los latchs son transparentes (74HC373) para evitar el retraso de las señales por un latch con flip-flop (74HC374). Según la hoja de especificaciones, las direcciones A15-A8 no necesitan ser demultiplexadas para los ciclos de lectura y escritura, lo que permite el ahorro de un chip. La señal de Address Latch Enable (ALE) del 80C188EC se conecta directamente a la entrada Latch Enable (LE) de los 373.

En el momento que el TBC quiera adueñarse de los buses, los 74HC373 deben adquirir un estado de alta impedancia. El microprocesador tiene una salida llamada HoLD Acknowledge (HLDA) que se activa cuando necesita poner los buses en tercer estado o en alta impedancia. Esta señal se conecta a la entrada Output Control (/OC) de los 373 para que pongan sus salidas en tercer estado cuando el microprocesador lo indica.

El bus de datos puede conectarse directamente pues el microprocesador hace el demultiplexaje cuando va a tomar datos del bus o sacarlos al mismo.

En la figura 4.6 se muestra la conexión preliminar de los buses. En la sección de

análisis de temporizado se demostrará que existe un error de diseño.

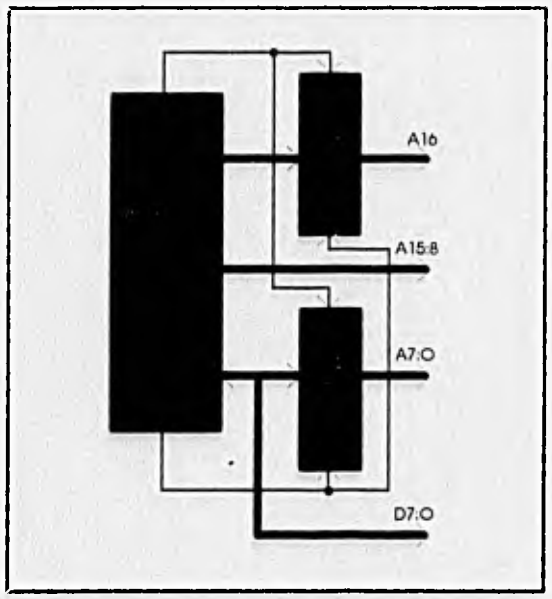


Fig. 4.6 Conexión de los buses del 80C188EC

4.2.4 Señales de control para las memorias del 80C188EC

Las señales del µP para manejar memoria son los chip selects y las señales de lectura y escritura /RD y /WR. Estas señales se programan gracias a la unidad generadora de chip selects y estados de espera (ver fig. 4.2) mediante sus registros correspondientes en el PCB, donde se indican la dirección inicial y la final para el chip select; y el número de estados de espera (0-15). Se conecta /RD a /OC (/Output Control) y /WR a /WE (/Write Enable) de cada memoria. Para seleccionar la RAM se conecta su señal de Chip Select (/CS) a la señal Lower Chip Select (/LCS) del 80C188EC, mientras que para la Flash se utiliza la señal Upper Chip Select (/UCS). ¿Por qué? Porque la tabla de vectores se escribe en RAM y debe estar en las primeras direcciones (00H a la 80H), mientras que el *boot-strap* (puesta de botas o inicio) debe estar en la dirección FFFF0H, es decir la última dirección del mapa de memoria. La señal /UCS es la única señal activa después del reset y direcciona, por defecto, en el rango FFC00H-FFFFFH.

La conexión se muestra en la figura 4.7.

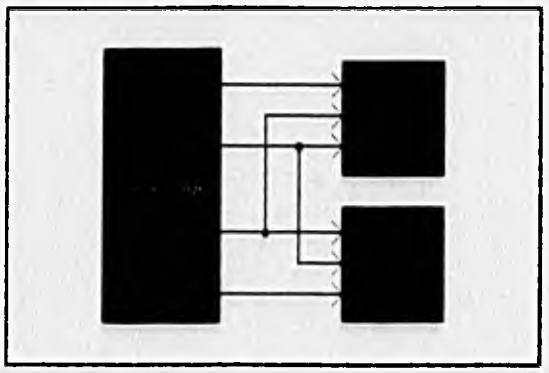


Fig. 4.7 Señales de control para las memorias

4.2.5 Señales de control para el acceso del 80C188EC al TBC esclavo

El TBC representa para el 80C188EC un espacio de puertos. Ya que será configurado para trabajar con 8 bits de datos, la señal Upper Data Strobe (/UDS) funcionará como la dirección A0 del bus y la señal Lower Data Strobe (/LDS) debe permanecer en uno. Para ser accesado, el microprocesador utiliza una señal de chip select generada internamente (/GCS0) y las señales ReaD y WRite de lectura y escritura (/RD y /WR); mientras que el TBC usa las señales Chip Select (/CS) y Read/Write (R/W).

Las señales del microprocesador no son compatibles con las del TBC. Todas estas señales deben seguir una secuencia, que se diseñará mediante una máquina de estados algorítmica (ASM). Esta lógica se representará como una caja negra (PLD) para diseñarla posteriormente. La conexión esquemática está en la figura 4.8.

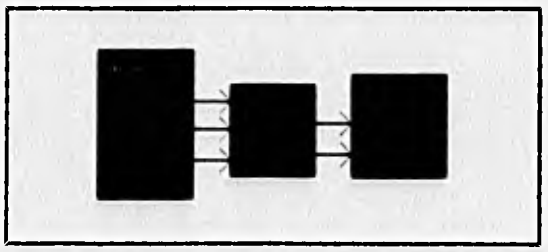


Fig. 4.8 Conexión de las señales del 80C188EC al TBC

La dirección de los registros del TBC es arbitraria y se han elegido las direcciones 2000H-2007H. La localización del PCB es por defecto en el último bloque del espacio de puertos. En la figura 4.9 se muestra el mapa de memoria para el sistema.

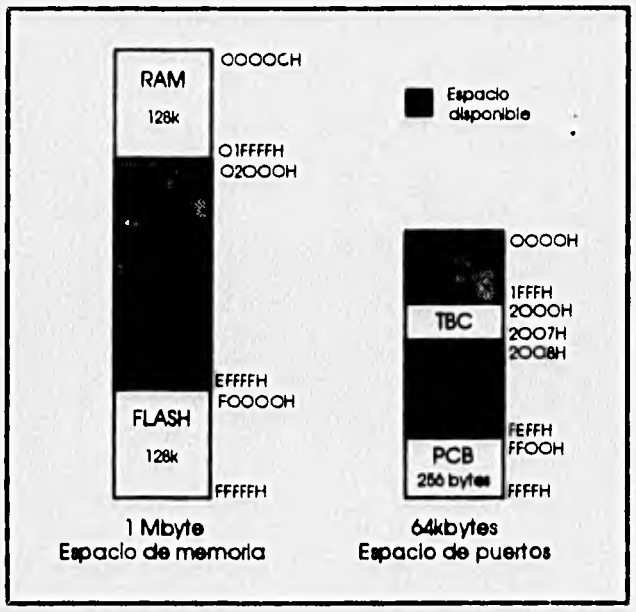


Fig. 4.9 Mapa de memoria del sistema

A continuación se presentan los módulos para el subsistema TBC.

4.2.6 Oscilador del TBC

Se utiliza un oscilador semejante al del microprocesador, también CMOS. El TBC trabajará a una frecuencia de 10MHz, según se mencionó con anterioridad. El diagrama se encuentra en la figura 4.10.

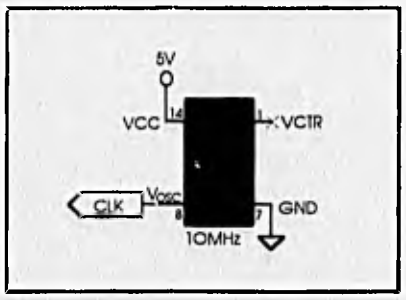


Fig. 4.10 Oscilador para el TBC

4.2.7 Reset del TBC

El reset del TBC se logra codificando las señales de excepción en el bus o Bus Exception Code (BEC2-BEC0). La siguiente tabla muestra las diferentes excepciones de bus para Motorola

/BEC2	/BEC1	/BEC0	Condition
H	H	H	No exception
H	H	L	Halt
H	L	H	Bus error
H	L	L	Retry
L	H	H	Relinquish and retry
L	H	L	Undefined (reserved)
L	L	H	Undefined (reserved)
L	L	L	Reset

Como las demás excepciones no se utilizarán, es posible evitar la lógica de decodificación al unir las tres entradas para lograr el reset. La señal de entrada /BEC se obtiene de RESOUT, previamente negada por ser activa alta.

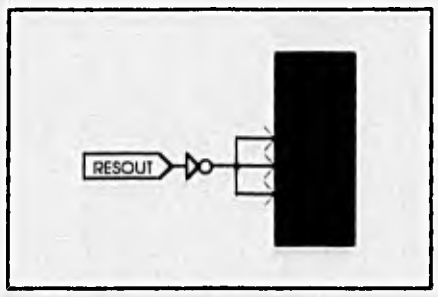


Fig. 4.11 Reset del TBC

4.2.8 Buses de direcciones y datos del TBC

Los buses en el TBC están demultiplexados o excepción de la dirección AO, que está multiplexada con otra señal (/UDS). Al seleccionar el modo de operación de 8 bits, la señal /UDS es ignorada y el bus de direcciones se comporta como el de cualquier procesador de 8 bits. Entonces, de las 32 líneas de direcciones se toman A16:O para la memoria RAM y de las 16 del bus de datos se toman sólo D7:O.

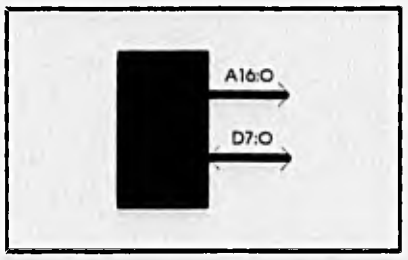


Fig. 4.12 Buses para RAM del TBC

4.2.9 Señales de control para la memoria RAM del TBC

La señal Address Strobe (/AS) permanece baja durante todo el ciclo de bus y el tiempo restante permanentemente alta mediante una resistencia de pull-up, por lo que es posible utilizarla para seleccionar la memoria RAM y con esto ahorrar la lógica de decodificación. Las señal de lectura /OC para la memoria RAM se obtiene negando la señal Read/Write (R/W) del TBC porque es activa alta, y lo de escritura /WE se puede obtener directamente. En un ciclo de lectura la señal R/W no cambia, por lo que la salida /OC permanecería activa aún cuando haya terminado el ciclo de lectura. Para

evitarlo se utiliza una compuerta OR con /AS, para que la señal /OC tenga la duración del ciclo. La señal Data Transfer ACKnowledge (/DTACK) se obtiene a partir de /AS. Hasta el momento no se sabe si es necesario retrasarla -que equivale a insertar estados de espera-, por lo que se incluye como caja negra (PLD).

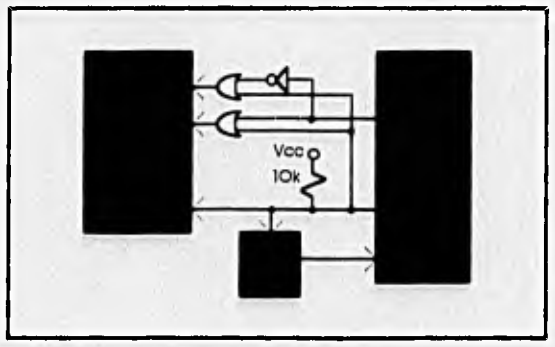


Fig. 4.13 TBC a RAM

4.3 SISTEMA FINAL

4.3.1 PLDs

La lógica que permite unir los sistemas 80C188EC y TBC está contenida en un PLD (Programmable Logic Device). Un PLD es un circuito integrado que permite crear funciones lógicas mediante arreglos lógicos generales. Los PLDs se clasifican en tres categorías: PALs y GALs, FPGAs y CPLDs, dependiendo de su complejidad interna.

Los PLDs están formados por arreglos de compuertas y flip-flops llamados celdas lógicas. Una celda lógica es la unidad básica que conforma un PLD. Las PALs y GALs tienen un número pequeño de celdas y poca capacidad de interconectarlas. Los FPGAs (Field Programmable Gate Arrays o arreglos de compuertas programables en el campo) contienen mayor capacidad de interconectar sus celdas al incluir un arreglo especial de interconexión. Los CPLDs (Complex PLDs o PLDs complejos) manejan varias celdas como Bloques de Arreglos Lógicos -BALs- y éstos pueden interconectarse con otros BALs mediante Arreglos de Interconexión Programables -AIPs-. En los CPLDs las celdas lógicas son llamadas *macroceldas*.

La macrocelda contiene un arreglo de compuertas AND de muchas entradas y una OR cuyas entradas son las salidas de las AND para obtener cualquier ecuación

booleana en forma de suma de productos; flip-flops programables que emulan la funcionalidad de flip-flops tipo D, T, JK y SR; buffers de tres estados, negadores y arreglos de interconexión que permiten realimentar las salidas y saltar compuertas o flip-flops que no se desea utilizar.

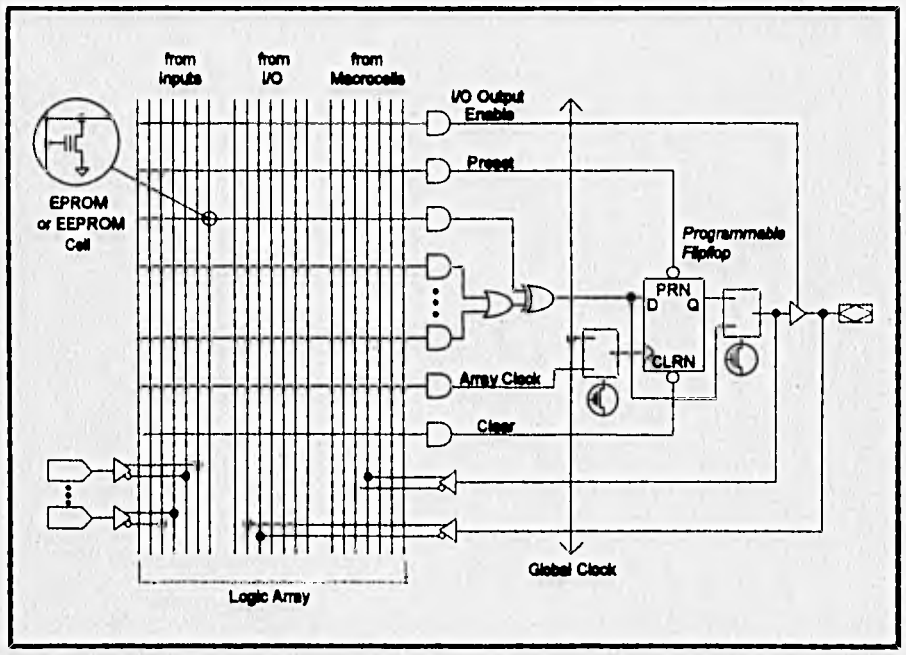


Fig. 4.14 Macrocelda

Las entradas para cada macrocelda son suplidas por los pines de entrada/salida del chip y los AIPs, de tal forma que las entradas de cada macrocelda pueden ser salidas de otras macroceldas o entradas en los pines del chip. Los AIPs permiten también dirigir las salidas de las macroceldas a los pines de salida o a otras macroceldas. Este tipo de PLDs es ideal para construir máquinas de estados gracias a su versatilidad de interconexión.

El PLD utilizado fue el EPM5032 de la familia MAX5000 de Altera, que pertenece a la categoría de CPLDs del tipo EPLDs (Erasable CPLDs o CPLDs borrables). Se eligió este PLD porque se necesitaba diseñar dos máquinas de estado algorítmicas -ASMs- para unir los sistemas del 80C188EC y el TBC. Esto permite también incluir todas las demás compuertas en el PLD y ahorrar chips.

Una de las enormes ventajas al utilizar PLDs es poder hacer la construcción del hardware sin tener su diseño completamente especificado; sólo se necesita conocer con certeza cuáles señales intervienen para los distintos módulos y tratar a la lógica como caja negra –según se hizo en el diseño de los módulos–, para programar la lógica e inclusive hacer cambios posteriormente. Es posible mandar a hacer el circuito impreso mientras se termina de diseñar el hardware, lo que representa un gran ahorro de tiempo y una mejor organización del trabajo.

4.3.2 Selección de memoria RAM

La memoria RAM se selecciona con una compuerta AND –observar la salida /CE en la tabla–, pues cuando los sistemas 80C188EC o TBC no hacen uso del bus para accesar memoria, las señales de chip select (/LCS y /AS con pull-up) se mantienen en estado lógico alto; y cuando la desean accesar (sólo un sistema a la vez) activan su señal correspondiente en estado bajo.

/LCS	/AS	Estado	/CE (RAM)
0	0	Nunca se presenta	X
0	1	80C188EC a RAM	0
1	0	TBC a RAM	0
1	1	No selección de memoria	1

Las señales de escritura y lectura del 80C188EC toman alta impedancia cuando el bus es cedido. En el caso del TBC la señal R/W está conectada directamente a /WE de la memoria y la señal /OC proviene de un negador. A pesar de que el TBC también pone en alta impedancia a R/W, esta señal está conectada a una salida del PLD y puede provocar corto circuito con las señales /RD y /WR del microprocesador. Para poder alambrarlas directamente es necesario darles salidas de tres estados controladas por HLDA. La señal /DTACK es bidireccional, por lo que es necesario agregar un buffer de tres estados también controlado con HLDA.

4.3.3 Señales de control restantes

Con respecto a las interrupciones, sólo es necesario adaptar las señales Interrupt ReQuest (/IRQ) del TBC con la INTerruption 0 (INT0) del 80C188EC –mediante un negador– para que la lógica sea compatible.

El arbitraje del bus y el acceso al TBC se hacen mediante máquinas de estados,

ambas incluidas en el PLD. La programación del PLD y el diseño de las máquinas se detallará en la sección de análisis de temporizado. Sin embargo, es posible dibujar el sistema final pues ya se tienen determinadas las entradas y salidas para el PLD. El alambrado se muestra en la figura 4.15.

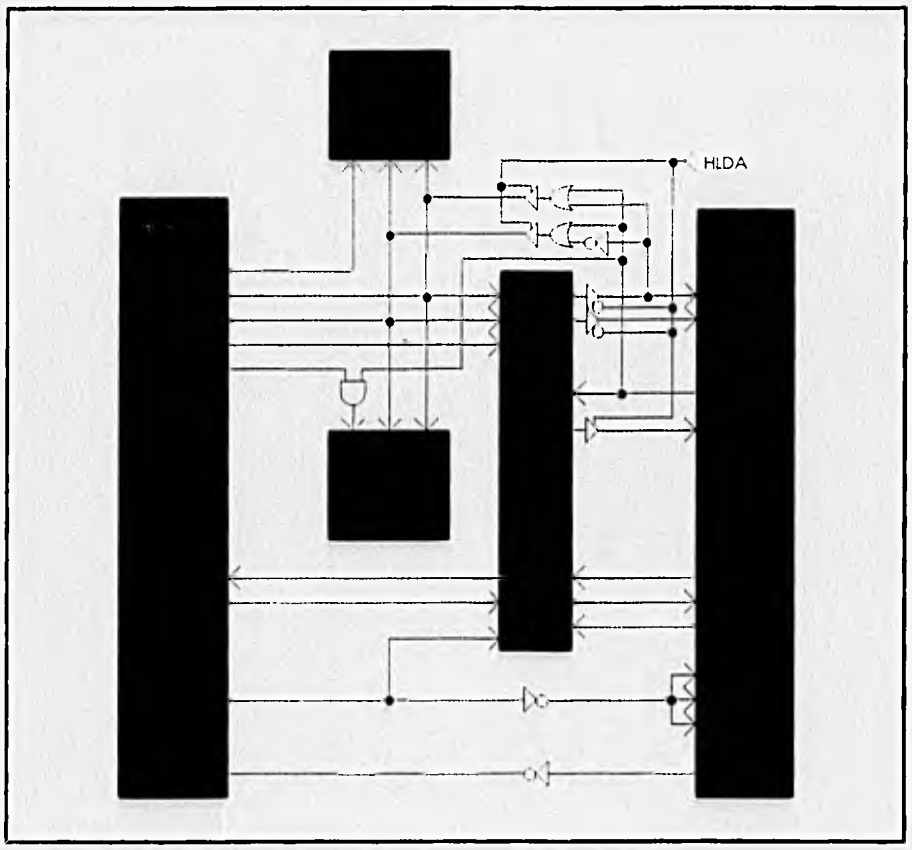


Fig. 4.15 Sistema final (señales de control)

4.4 ANÁLISIS DE TEMPORIZADO

El análisis de los diagramas de tiempos es básico para asegurar el correcto funcionamiento del hardware. Se trata de considerar los retrasos máximos de todas las señales del sistema para asegurar que se cumplen, en cada uno de los componentes del sistema, las cotas estipuladas por los fabricantes.

4.4.1 Retrasos en el PLD

Los retrasos en las señales de control provocados por las compuertas incluidas en el PLD se pueden obtener de un archivo de análisis de temporizado (JUNCTBOX.TAO) generado al programar el PLD. Este archivo proporciona los retrasos máximos en base a los retrasos estipulados por el fabricante.

En seguida se muestra un fragmento del archivo en el que se muestran sólo los retrasos de las señales de control para memorias. La programación del PLD se explicará posteriormente.

Archivo JUNCTBOX.TAO (fragmento)

DELAY MATRIX

Destination

Source	/bec	/cenam	int0	/ds	/rd	r/w	/uds	/wr
/as		15.0ns						
/rq			15.0ns					
/cs		15.0ns						
r/w					15.0ns			15.0ns

4.4.2 Ciclos de lectura/escritura a memorias

En la figura 4.18 se presenta el diagrama de tiempos para el 80C188EC en los ciclos de lectura y escritura.

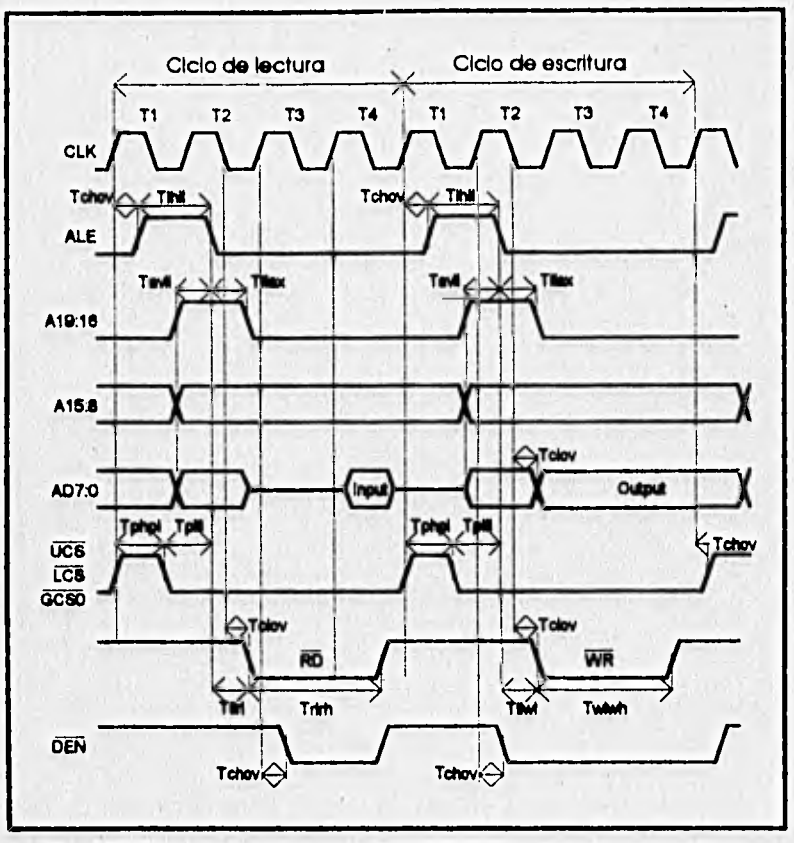


Fig. 4.18 Diagrama de tiempos del 80C188EC para lectura/escritura

Los valores máximos de los retrasos son

Tchov=Tclov=30ns

Tihll=62ns

Tavll=Tllax=Tphpl=Tpll=29ns

Trlrh=Twhwl=149ns

Tlrl=Tlwl=24ns

- Retraso de los 74HC373

Primeramente es necesario comprobar que los 74HC373 son capaces de soportar la operación a la frecuencia requerida. Los parámetros críticos en este caso

son el ancho del pulso en la entrada de habilitación LE (Tw) y los tiempos de anteposición y sostenimiento de los datos (setup time Tsu and hold time Th), mostrados en la figura 4.19.

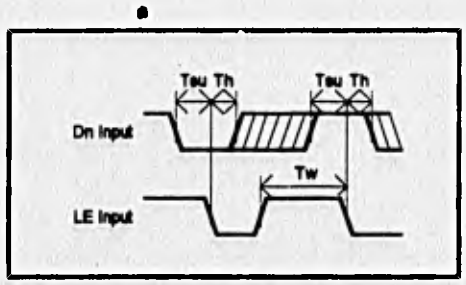


Fig. 4.19 Temporizado de un 74HC373

Debe cumplirse que Tihl >Tw, Tavll >Tsu y Tllax >Th. En el peor de los casos, los valores son

Tw=24 ns <62ns ☑

Tsu=15 ns <29ns ☑

Th=5 ns <29ns ☑

Los 74HC373 provocan un retraso máximo de Tp=45ns en la aparición de las direcciones en el bus.

En seguida se presenta el diagrama de tiempos típico de lectura/escritura para una memoria, acotado según el estándar de símbolos para parámetros (no es notación JEDEC).

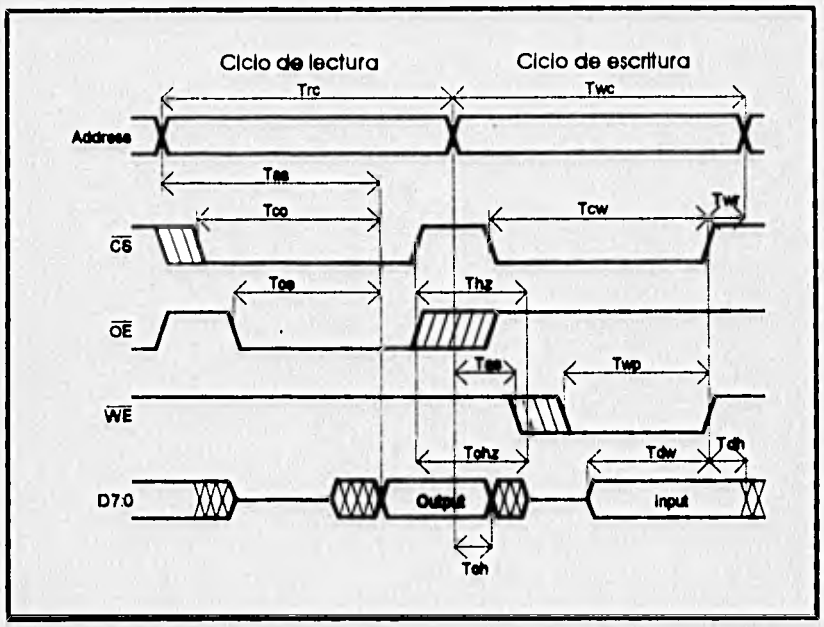


Fig. 4.20 Temporizado típico de una memoria para lectura/escritura

• 80C188EC a RAM

La memoria RAM elegida fue la HM628128P-10 de Hitachi, cuyo tiempo de acceso es de 100ns. Esta memoria tiene las siguientes cotas:

Trc=100ns; tiempo del ciclo de lectura (mínimo)

Taa=100ns; tiempo de acceso para direcciones (máximo)

Tco=100ns; tiempo entre activación de chip select (/CS o /CE) y salida válida (máximo)

Toe=50ns; tiempo entre activación de output enable (/OE o /OC) y salida válida (máximo)

Tohz=35ns; tiempo entre desactivación de /OE y salida en tercer estado (máximo)

Twc=100ns; tiempo del ciclo de escritura (mínimo)

Tcw=90ns; tiempo entre activación de chip select y final de escritura (mínimo)

Twr=5ns; tiempo de recuperación después de escritura (mínimo)

Twp=75ns; ancho de pulso de write enable (/WE) (mínimo)

Tdw=40ns; tiempo de superposición de los datos hasta el fin del ciclo (mínimo)

Si no se desea insertar estados de espera debe cumplirse que Trc<4T para lectura y Twc<4T para escritura, donde T=77ns y es la duración de un ciclo de reloj del 80C188EC. Los valores de Trc y Twc son 100ns para el peor de los casos. Entonces

Trc=Twc=100ns <308ns ☑

En todas las mediciones se tomó como referencia el comienzo del ciclo T1.

En un ciclo de lectura, el 80C188EC necesita que los datos estén presentes en el bus al iniciar **T4** es decir, que hayan transcurrido máximo **T1+T2+T3=231ns**. Tomando como referencia a **T1**, se tiene que las direcciones aparecen en el bus en

Tchov(retraso con respecto al reloj)+**Tihll**(duración de ALE)-**Tavll**(anticipación de las direcciones)+**Tp**(retraso del 74HC373)=108ns ...(A)

A partir de ese momento pueden suceder tres cosas

1. Que las señales /CS1 y /OE ya hayan cumplido sus cotas (Tco y Toe) y el tiempo máximo en aparecer los datos es **Taa**=100ns. Con esto, los datos estarán listos en

   t=108(ecuación A)+100=208ns <231ns ☑

2. Que la señal /CS1 se retrase más que las direcciones. El máximo retraso de /LCS es

   Tchov(retraso con respecto al reloj)+**Tihll**(duración de ALE)-**Tpll**(anticipación de /LCS con respecto a ALE)+**Tpld**(retraso por la AND)=78ns ...(B)

Esto quiere decir que /LCS se anticipa a las direcciones, por lo que sólo resta revisar que cumpla su cota Tco=100ns. /LCS dura activa al menos T2 y T3, es decir 154ns. Entonces se cumple que

Tco=100ns<154ns ☑

3. Que la señal /OE se retrase más que las direcciones. El máximo retraso de /RD es

$$T1+\frac{T2}{2}+Tclov(\text{retraso con respecto al reloj})=136ns \quad ...(C)$$

Se puede ver que el pulso de /RD se retrasa más que las direcciones. Si Toe=50ns, los datos estarán listos en

t=136ns(ecuación C)+Toe=186ns <231ns ☑ ...(D)

Los datos se mantienen en el bus durante **Tohz**=35ns como máximo después de desactivar /LCS –que es la última señal en desactivarse–, esto es, durante 65ns en **T1** (**Tchov+Tohz**). Sin embargo las direcciones siguientes pueden aparecer en las terminales AD7:0, en el mejor de los casos (Tchov=0), en

Tihll(duración de ALE)-**Tavll**(anticipación de las direcciones)=33ns<65ns ☑

Se ha descubierto un corto circuito de las direcciones del microprocesador con los datos de la RAM (que puede durar hasta 32ns), por lo que debe incluirse un transceptor o *transceiver* para aislar las terminales AD7:0 del bus de datos. Se incluyó el 74HC245, un transceiver octal con salidas de tres estados.

Este transceiver posee dos señales de control: una para habilitar las salidas (/G) y otra para indicar la dirección de los datos (DIR). Estas señales se conectan a las señales Data Transmit/Receive (DT/R) que indica la dirección de los datos, y Data ENable(/DEN) que indica que los datos son válidos para el bus o para el microprocesador. El retraso máximo que provoca es de Ttrans=27ns, que sumado a los retrasos máximos es aún menor a los 231ns de que se dispone. Es decir

186ns(ecuación D)+Ttrans(retraso del 74HC245)=213ns<231ns ☑.

El 74HC245 necesita que la señal de control /G esté lista con 45ns de anticipación para salir del tercer estado antes de manejar datos. Esto significa que el microprocesador debe generar /DEN en

T1+T2+T3-45ns(anticipación de /G)=186ns

El máximo retraso de /DEN es

T1+T2+Tchov(30ns)=184ns<186ns ☑

El diseño de los buses para el 80C188EC ha cambiado (Fig. 4.7), siendo ahora el mostrado en la figura 4.21.

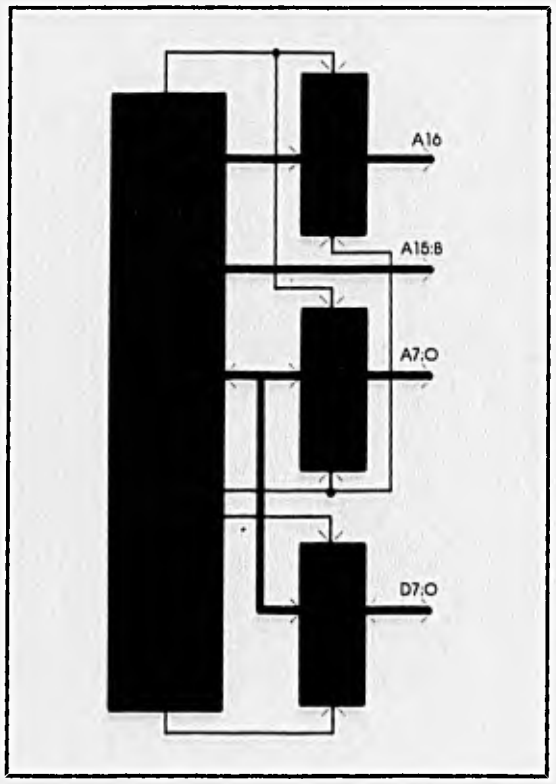


Fig. 4.21 Nueva conexión de los buses del 80C188EC

Para el ciclo de escritura lo importante es que los datos aparezcan en el bus durante Tdw=40ns antes de que termine el ciclo y que el ancho del pulso /WE sea mayor a Twp. Se sabe que para el 80C188EC el ancho del pulso /WR es Twlwh=149ns, entonces se cumple que

Twp=75ns<149ns ☑

Los datos aparecen en AD7:0 en

Tchov+Tlhll+Tllax+Ttrans(retraso del 74HC245)=148ns

La señal /WR se comporta igual que /RD, entonces se retrasa 136ns (ecuación C) y dura Twlwh=149ns. La diferencia de tiempo entre la aparición de los datos y la deshabilitación de /WR es de 136+149-148=137ns. Con esto, se cumple que

Tdw=40ns<164-20ns ☑

Se vio que /LCS se retrasa menos que /RD o /WR (ecuación B), obviamente se cumple la cota Tcw ☑.

Sólo resta verificar que se cumpla Twr. Debe cumplirse que Twr<Tihll-Tavll.

Twr=5ns<33ns ☑

El tiempo Tdh no causa conflicto con las terminales AD7:0 al haber incluido el 74HC245.

• 80C188EC a FLASH

Los ciclos de lectura/escritura para la memoria Flash son exactamente iguales a los de la memoria RAM (los diagramas de tiempos son idénticos) y la señal /UCS se comporta de la misma manera que /LCS. Por ello, si las cotas de la memoria Flash son menores que las de la RAM, quiere decir que también aquella cumple con el temporizado. En la tabla se muestra la comparación de las cotas. La memoria Flash utilizada fue la Am29F010-90PC de Advanced Micro Devices.

Cota	RAM	FLASH	$T_{RAM}>T_{Flash}$?
Trc	100	90	☑
Taa	100	90	☑
Tco	100	90	☑
Toe	50	35	☑
Tohz	35	20	☑
Twc	100	90	☑
Tcw	90	45	☑
Twr	5	0	☑
Twp	75	45	☑
Tdw	40	45	■

La cota Tdw es mayor para la Flash porque indica que es posible escribirle datos con mayor anticipación; es decir, no importa –y es mejor– que sea mayor. La memoria Flash es más rápida que la RAM y cumple los requisitos de temporizado ☑.

• TBC a RAM

El ciclo de lectura/escritura de los procesadores de Motorola es un poco distinto. En la figura 4.22 se muestra el diagrama de tiempos del TBC[a].

[a] Ver en el apéndice D los valores de las cotas para todos los diagramas de temporizado

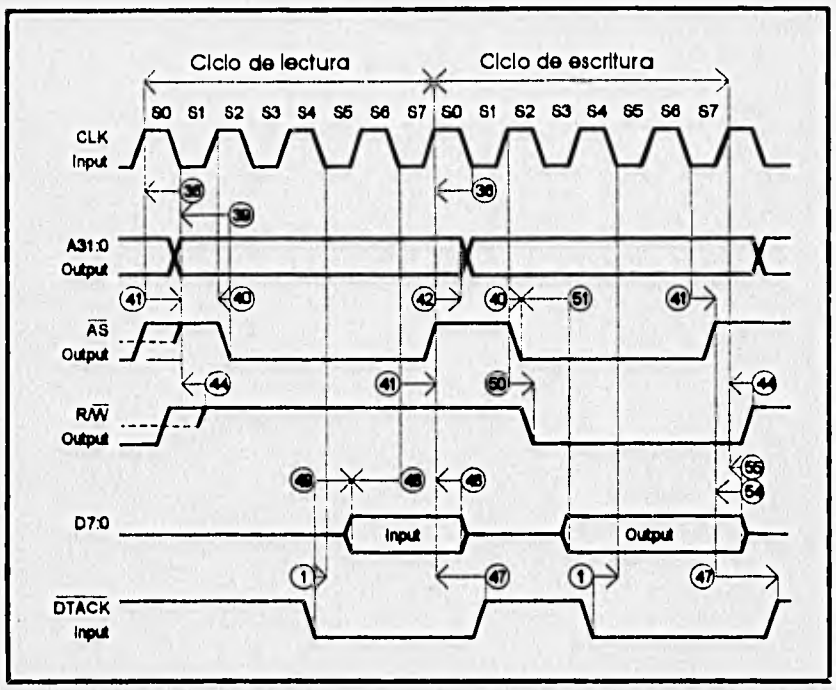


Fig. 4.22 Diagrama de tiempos del TBC para lectura/escritura

En el ciclo de lectura debe cumplirse que SO+S1+...+S7>Trc. Si cada medio ciclo dura 5Ons, se cumple que

Trc=1OOns<4OOns ☑

Para el TBC es necesario que los datos aparezcan en el bus en

t=S0+S1+S2+S3+S4+S5+S6-(46)=34Ons como máximo si no se desea insertar estados de espera.

La señal de /CS1 se obtiene de /AS en una AND. El máximo retraso es

S0+S1+(40)+Tpld(AND)=165ns ...(E)

Mientras tanto, la señal de /OE se obtiene al negar R/W. Esta señal se retrasa

S0+(44)+Tpld(negador)=12Ons

Esto quiere decir que la señal más retrasada es /CS1, por lo que hay que esperar Tco para que aparezcan los datos en el bus. El tiempo total es

165ns+Tco=265ns<34Ons ☑

Por lo tanto, /DTACK puede obtenerse de /AS en forma directa para que no existan estados de espera. De esta forma se cumplen (1) (2Ons) y (47) (O ns mínimo).

El tiempo que la memoria sostiene los datos Tohz es indiferente por la cota (46) (O ns).

Para el ciclo de escritura lo importante es que los datos aparezcan en el bus durante Tdw=4Ons antes de que termine el ciclo y que se cumplan Tcw y Twp.

La señal /WE se retrasa

S0+S1+(50)+Tpld=175ns

y la señal /CS1 se retrasa 165ns (ecuación E).

Estas señales permanecen activas por lo menos hasta S6, por lo que tienen intervalos de duración mínimo de

S0+S1+...+S6-175ns=175ns para /WE y

S0+S1+...+S6-165ns=185ns para /CS1

Comparando con Twp y Tcw se tiene que

Twp=75ns<175ns ☑ y que

Tcw=9Ons<185ns ☑

Los datos aparecen en

S0+S1+(40)+(51)=24Ons

La mínima diferencia de tiempo entre la aparición de los datos y la deshabilitación de /WE o /CS1 ocurre cuando (41) es cero. Esta diferencia es

S0+S1+...+S6-24O=11Ons

Entonces se cumple que

Tdw=4Ons<11Ons ☑

4.4.3 Ciclos de lectura/escritura del 80C188EC al TBC esclavo

En la figura 4.23 se muestra el diagrama de temporizado para el TBC cuando es esclavo.

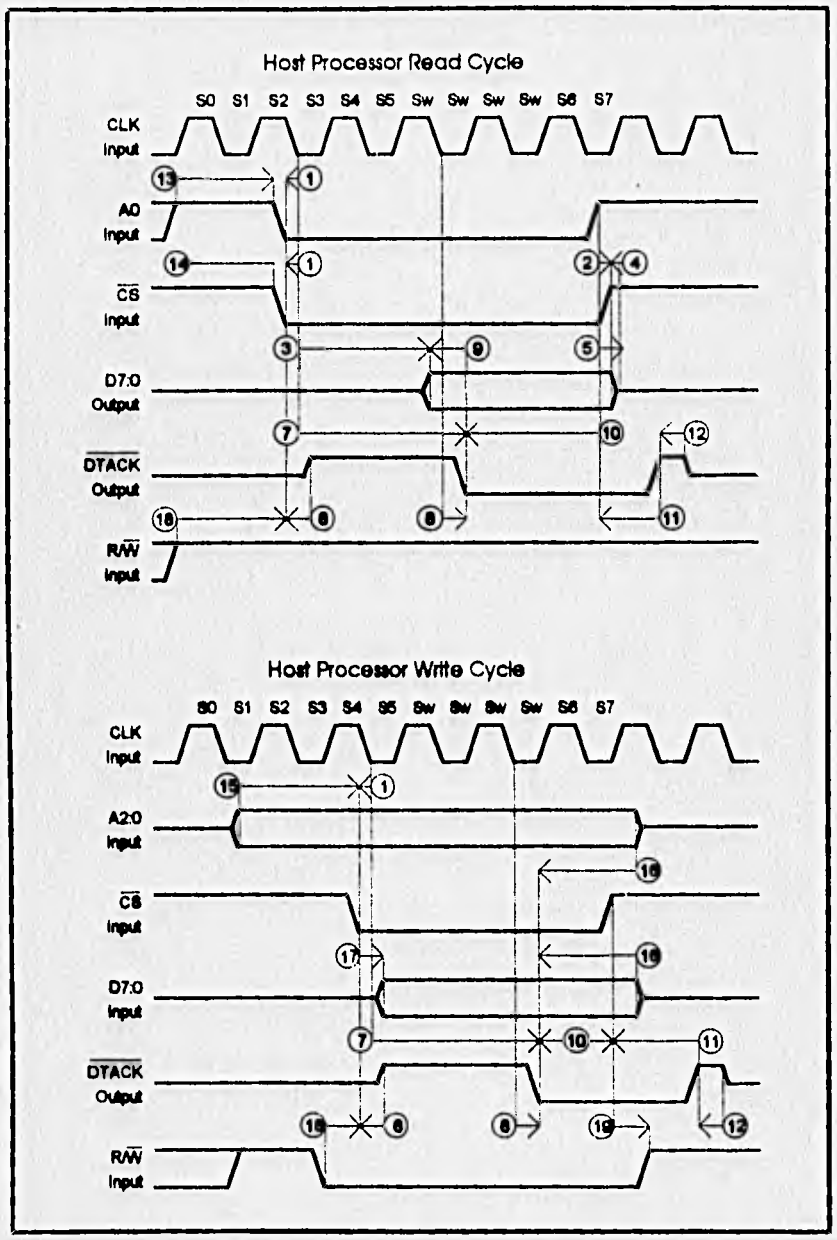


Fig. 4.23 Ciclos de lectura/escritura al TBC como esclavo

A partir del análisis de temporizado se descubrió que debía diseñarse una máquina de estados para que el 80C188EC pudiera accesar al TBC como un puerto o periférico –TBC como esclavo–. El 80C188EC genera primero el chip select /GCS0 con 29ns de retraso (Tphpl) y después indica si el ciclo es de lectura o de escritura (ecuación C, 136ns de retraso); en cambio el TBC necesita saber si el ciclo es de lectura o escritura (cota ⑱=20ns) antes de que se presente la señal de chip select.

Para los ciclos de lectura o escritura se sabe que las direcciones aparecen en el bus en 108ns (ecuación A), por lo tanto están listas antes de que aparezca R/W a los 136ns.

Una vez que se ha cumplido ⑱ debe generarse /CS. En el peor de los casos /CS puede no haber cumplido con ① (setup para entrada asíncrona) y debe transcurrir otro ciclo de reloj para ser reconocida, por lo que el tiempo transcurrido desde que el 80C188EC comenzó el ciclo es

136ns+⑱+100ns(otro ciclo)+①=276ns hasta llegar al ciclo S3.

A partir de ese momento deben cumplirse las cotas ⑦ (290ns) y ⑩ (100ns) para desaparecer las direcciones o desactivar la señal /CS (lo que ocurra primero), lo que representa un retraso total de 276+290+100=666ns. Para el 80C188EC cualquiera de los dos ciclos dura T1+T2+T3+T4=308ns<666ns ■, de donde se deduce que deben insertarse estados de espera.

Con objeto de sincronizar las señales y asegurar que se cumpla el temporizado se decidió utilizar una máquina de estados síncrona. El reloj de la máquina es el del 80C188EC pues es el reloj que indica el inicio del ciclo y el más rápido.

El primer estado de la máquina (T1) es esperar a que se genere /GCS0 para saber que se desea accesar al TBC. Las salidas de este estado son R/W=/CS=1. Una vez que se genera /GCS0 se pasa al segundo estado (T2) de la máquina esperando que se genere /RD o /WR para poder generar R/W. El tiempo mínimo transcurrido entre /GCS0 y /RD o /WR es 136ns(ecuación C)-29ns(Tphpl, ver Fig. 4.18)=107ns. Como un estado de la máquina dura un ciclo de reloj (77ns), es necesario un tercer estado (T3) para esperar la aparición de /RD o /WR. Las salidas del segundo y tercer estado son R/W=/WR (si el ciclo es de lectura /WR=1, y si es de escritura en el tercer estado ya se cumple que /WR=0) y /CS=0.

La máquina pasa al cuarto estado y permanece en él hasta que el procesador haya leído o escrito datos. A partir de la generación de /CS debe esperarse

100ns(otro ciclo)+①+⑦+⑩ =510ns para que el ciclo termine. Por tanto, es necesario insertar $\frac{510}{77}$ =6.6 ó 6 estados de espera (TW) que el 80C188EC insertará

entre T3 y T4. Después de los 6 estados de espera o 462ns, el 80C188EC hará la lectura o escritura de datos en T4 sumando 539ns que cumplen ser más de 510ns ☑.

La máquina debe pasar al primer estado al terminar T4. En dicho ciclo las señales /RD o /WR son desactivadas, por lo que la AND de las dos señales sirve para saber cuándo salir del estado.

En la transición del cuarto al primer estado de la máquina, es decir al desactivar /CS, ocurre lo siguiente: si el ciclo es de lectura los datos permanecen en el bus durante ④ (6Ons). Este retraso no importa pues el 74HC245 ya ha sido deshabilitado por /DEN. Si el ciclo es de escritura se tiene la certeza de haber cumplido las cotas ⑯ (es igual a ⑩=100ns) y ⑲ (O ns).

La ASM síncrona se muestra en la figura 4.24.

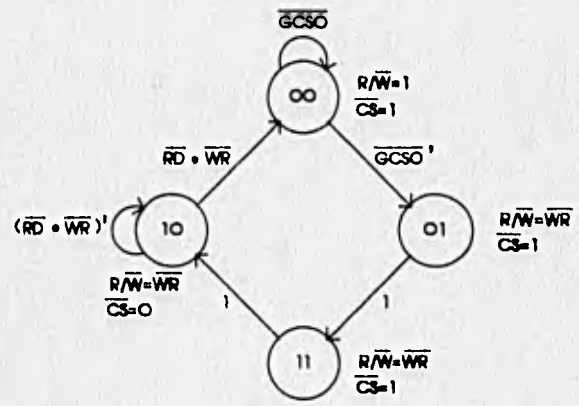


Fig. 4.24 ASM síncrona para accesar al TBC como esclavo

4.4.4 Arbitraje del bus

El 80C188EC es el procesador 'anfitrión' –host processor– en el sistema, es decir que posee los buses normalmente. Cuando se le solicita el bus mediante la señal de entrada HOLD el microprocesador termina el ciclo de instrucción, inserta periodos de tiempo 'ociosos' –idle state– y se prepara para ceder el bus. Al poner en tercer estado o estado inactivo sus señales de control (chip selects, /RD, /WR etc.) contesta con HoLD Acknowledge (HLDA) avisando que el bus ha sido cedido. La señal HOLD debe permanecer activa durante todo el tiempo que se desea tener el bus. Al desactivarse HOLD, el microprocesador desactiva HLDA y retoma la posesión de los buses.

El TBC solicita el bus mediante la señal Bus Request (/BR) y espera a que se le diga que el bus ha sido cedido con Bus Grant (/BG). Entonces el TBC activa la señal Bus Grant ACKnowledge (/BGACK) y desactiva /BR mientras ejecuta sus ciclos de bus. Al terminar, el TBC libera el bus desactivando /BGACK una vez que ha puesto sus señales en tercer estado o desactivadas –según sea el caso–.

En la figura 4.25 se muestra el diagrama de temporizado para el arbitraje del bus. Las señales se suceden de arriba para abajo, es decir primero /BR, después HOLD, HLDA, /BG y por último /BGACK. /AS se activa cuando el TBC ya posee el bus.

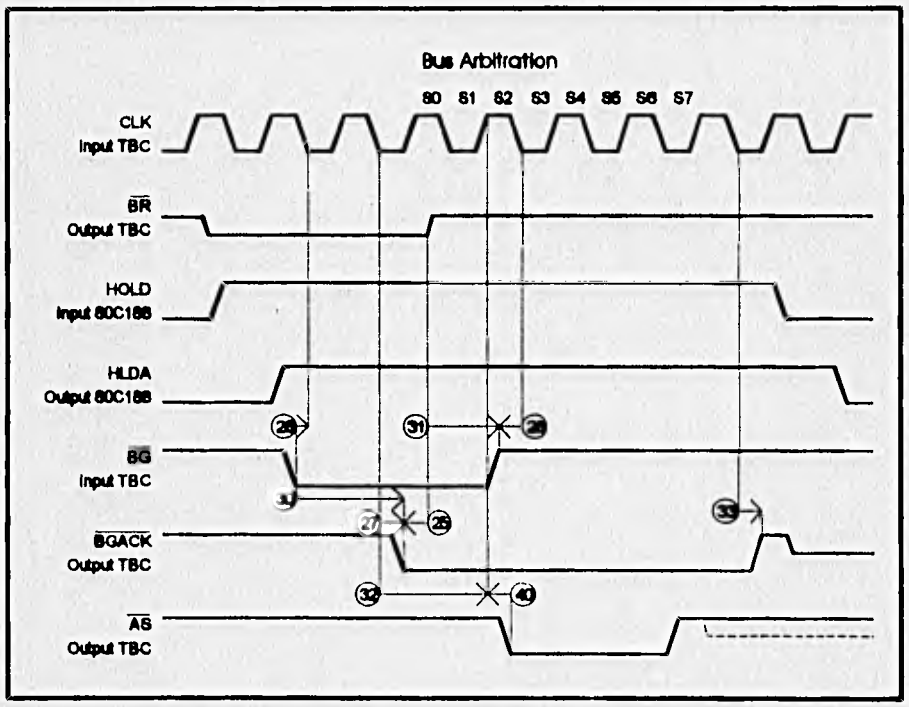


Fig. 4.25 Diagrama de tiempos para el arbitraje del bus

Es imposible realizar una conexión directa, por lo que se usa una máquina de estados para adaptar las señales entre los procesadores. Ya que no es necesario sincronizar las señales se utiliza una máquina de estados asíncrona, que tiene la ventaja de ser más veloz y eliminar la latencia en el arbitraje. La primera tentativa se presenta en la figura 4.26.

El TBC solicita el bus mediante la señal Bus Request (/BR) y espera a que se le diga que el bus ha sido cedido con Bus Grant (/BG). Entonces el TBC activa la señal Bus Grant ACKnowledge (/BGACK) y desactiva /BR mientras ejecuta sus ciclos de bus. Al terminar, el TBC libera el bus desactivando /BGACK una vez que ha puesto sus señales en tercer estado o desactivadas –según sea el caso–.

En la figura 4.25 se muestra el diagrama de temporizado para el arbitraje del bus. Las señales se suceden de arriba para abajo, es decir primero /BR, después HOLD, HLDA, /BG y por último /BGACK. /AS se activa cuando el TBC ya posee el bus.

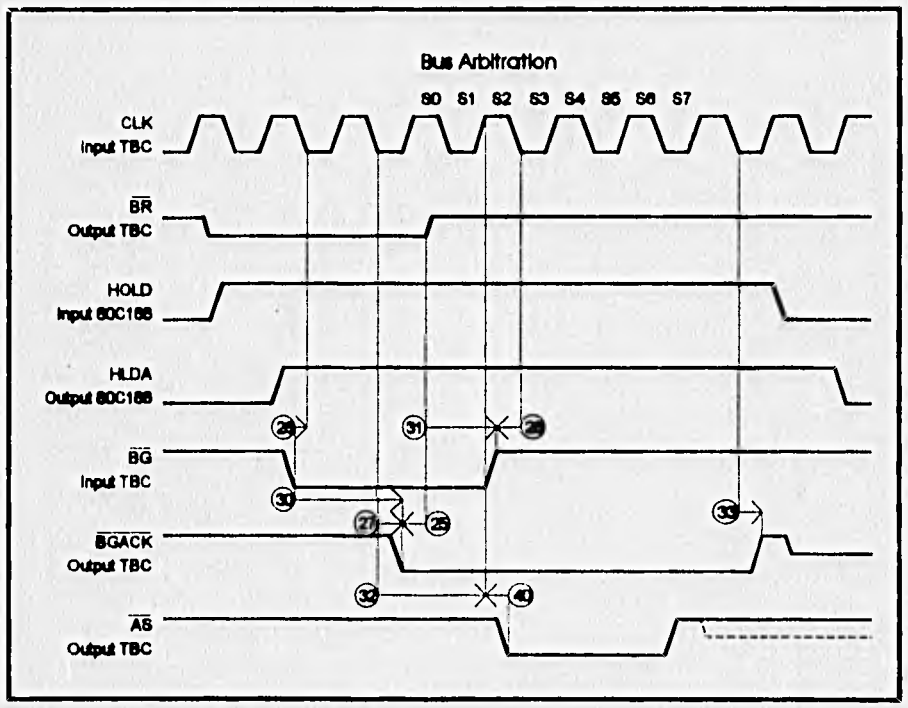


Fig. 4.25 Diagrama de tiempos para el arbitraje del bus

Es imposible realizar una conexión directa, por lo que se usa una máquina de estados para adaptar las señales entre los procesadores. Ya que no es necesario sincronizar las señales se utiliza una máquina de estados asíncrona, que tiene la ventaja de ser más veloz y eliminar la latencia en el arbitraje. La primera tentativa se presenta en la figura 4.26.

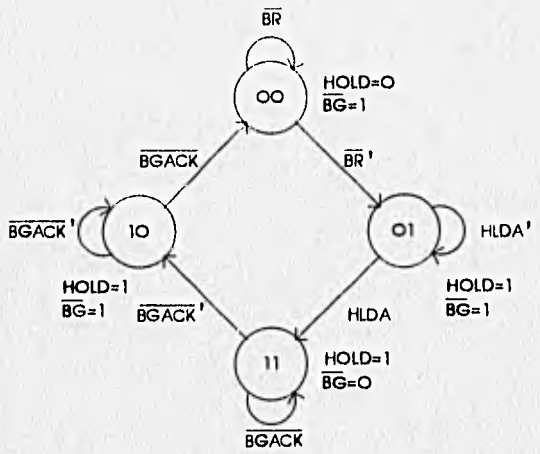


Fig. 4.26 ASM1 para el arbitraje del bus

En el primer estado se espera a que el TBC solicite el bus, por lo que las salidas son HOLD=O (petición desactivada) y /BG=1 (no se ha solicitado el bus). Una vez que aparece /BR (activa baja) se pasa al segundo estado donde se avisa al 80C188EC que se solicita el bus con HOLD=1 y se espera a que flote sus señales. El bus no ha sido cedido por lo que aún /BG=1. Cuando el procesador está listo para ceder el bus activa HLDA, entonces se avisa al TBC que el bus está disponible (/BG=O) pasando al tercer estado de la máquina. Se debe permanecer en este estado hasta que el TBC confirma que ha tomado el bus activando /BGACK. En ese momento se pasa al cuarto estado y de desactiva /BG para que el TBC pueda solicitar nuevamente el bus; HOLD=1 todavía. El ciclo termina cuando el TBC libera el bus desactivando /BGACK y se regresa al primer estado.

La máquina de estados no tiene condiciones de carrera porque en cada transición de estado sólo cambia una variable o un bit (secuencia OO,O1,11,1O,OO etc.). Para eliminar los *hazards* o "riesgos" estáticos y dinámicos se deben minimizar las ecuaciones sin dejar aislado ningún mintérmino. Las ecuaciones se obtienen por un método de inspección visual[4]. Sean Z2 y Z1 las variables de excitación (estado siguiente), y z2 y z1 las variables secundarias (estado presente). Entonces

[4] *Método para la especificación y síntesis de circuitos secuenciales síncronos* por Pedro A. Molina Foncerrada. Documento paro lo revista Boletín IIE de enero/febrero de 1991, vol. 15, num. 1

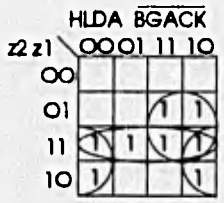
Z2=z2'z1HLDA+z2z1+z2z1'/BGACK'

Z1=z2'z1'/BR'+z2'z1+z2z1/BGACK

HOLD=z2'z1+z2z1+z2z1'

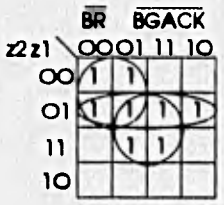
/BG=z2'z1'+z2'z1+z2z1'

Para Z2



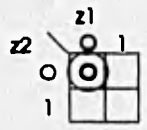
Z2=z2z1+z1HLDA+z2/BGACK'

Para Z1



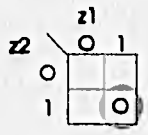
Z1=z2'/BR'+z2'z1+z1/BGACK

Para HOLD



HOLD=(z2'z1')'=z2+z1

Y para /BG



/BG=(z2z1)'=z2'+z1'

En la práctica existió un problema con esta máquina porque había contención en el bus; esto es que los dos procesadores utilizan los buses al mismo tiempo. Con ayuda del osciloscopio se detectó que la máquina cedía el bus en el estado 11, no se detenía jamás en el estado 10 y pasaba al estado 00 liberando el bus al 80C188EC (la salida del estado es HOLD=0) un instante después de haberlo concedido al TBC. ¿El problema? Un *hazard* esencial.

Un *hazard* es una condición en la que un solo cambio de variable produce un cambio momentáneo en la salida cuando no debe ocurrir, debido a los retrasos de las compuertas que forman el circuito combinacional. Existen tres tipos de *hazards*

1. Estático: cuando se desea que la salida permanezca en uno y cambia momentáneamente a cero (estático 0) o viceversa (estático 1)
2. Dinámico: cuando se desea que la salida cambie de cero a uno o de uno a cero y lo hace cambiando más de tres veces de valor (por ejemplo 0-1-0-1)
3. Esencial: cuando se producen salidas no deseadas debido a que la diferencia de retardo entre dos rutas de salidas provocadas por la misma entrada es muy grande.

Los *hazards* estáticos y dinámicos se evitan minimizando las ecuaciones según se dijo anteriormente. Los *hazards* esenciales son de característica particular pues dependen de la máquina específica, y se eliminan mediante el análisis del circuito lógico y del temporizado.

En la figura 4.27 se muestra el circuito lógico de la ASM1 (figura 4.26).

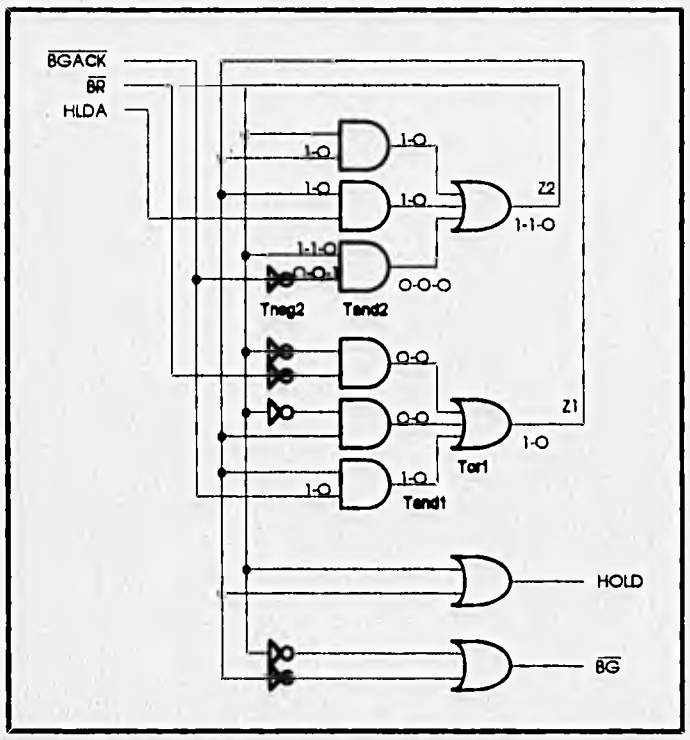


Fig. 4.27 Diagrama lógico de la ASM1

El error se presenta al pasar del estado 11 al estado 1O. En el estado 11 el 80C188EC ha cedido el bus (HLDA se activa) y se avisa al TBC activando /BG. Cuando el TBC responda con /BGACK=O se pasará al estado 1O. Las entradas y salidas en el estado 11 son las indicadas por los números de la izquierda.

En el momento que /BGACK=O, las señales cambian a los números de la derecha. La transición empieza cuando la salida Z1 cambia rápidamente a cero (depende directamente de /BGACK) con un retraso **Tand1+Tor1** y se retroalimenta para obtener Z2. Si el retraso **Tneg2+Tand2** es mayor que **Tand1+Tor1** se presentarán salidas intermedias para la variable Z2 provocando que cambie de 1 a O aunque debería permanecer en 1 –para lograr el estado 1O–. Al retroalimentar Z2 como O se llegará al estado OO en forma estable muy rápidamente, liberando la señal HOLD y provocando contención en el bus. La suposición de la diferencia de retrasos entre las compuertas se pudo comprobar en la práctica y se descubrió un *hazard* esencial que provocó el mal funcionamiento de la máquina, ¡qué hallazgo!

La primera tentativa de solución fue buscar que el cambio de estado desde 11 hasta 10 no comenzara al activarse /BGACK (/BGACK=0) sino hasta que /BR se desactive (quitando a /BGACK de Tand1), de este modo se aprovecharía el retraso que marca la cota (25)=20ns para que la salida del negador Tneg2 esté estable (0-1-1 en vez de 0-0-1) y lograr que Z2 no cambie a cero en ningún instante de la transición. Sin embargo, en la práctica esta solución tampoco funcionó porque los 20ns no fueron suficientes para permitir que las señales estuvieran estables.

La solución definitiva fue hacer que el cambio de estado se hiciera cuando /AS se active, pues en ese momento se tiene la seguridad de que el TBC ya posee el bus. El tiempo mínimo que transcurre desde que /BGACK se activa hasta que /AS se activa son 1.5 periodos de reloj o 150ns (cota (32)).

La ASM1 tiene además un *race* o condición de carrera. Si el TBC solicita nuevamente el bus en el estado 10 la máquina pasará rápidamente por el estado 00, hará HOLD=0 y no esperará a que se desactive HLDA para terminar el ciclo. Por lo tanto, debe incluirse a HLDA para salir del estado 00.

Para las ecuaciones de salida sólo se utilizan las variables de excitación y se evita la minimización.

Para corregir el hazard esencial y el race se hizo una segunda máquina, que se presenta en la figura 4.28.

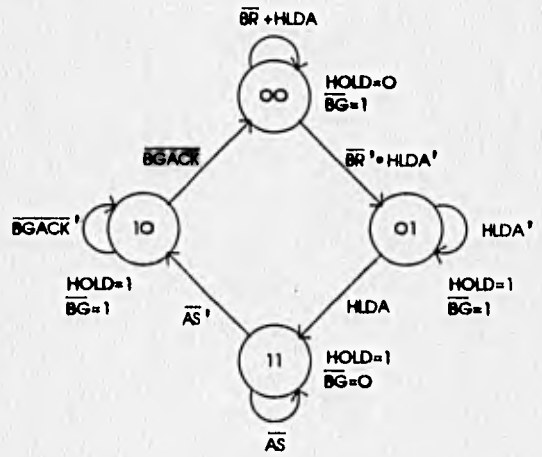


Fig. 4.28 ASM2 para el arbitraje del bus

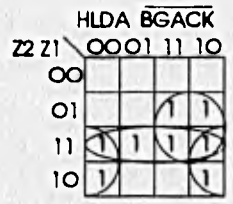
Por inspección visual

Z2=z2'z1HLDA+z2z1+z2z1'/BGACK'

Z1=z2'z1'(/BR'HLDA')+z2'z1+z2z1/AS

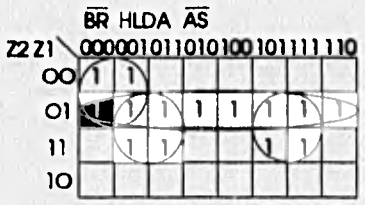
Para verificar y eliminar hazards se minimizan las ecuaciones

Para Z2



Z2=z2z1+z2/BGACK'+z1HLDA

Para Z1



Z1=z2'z1+z1/AS+z2'/BR'HLDA'

Las ecuaciones de las salidas no cambian

Para HOLD

HOLD=z2+z1

Y para /BG

/BG=z2'+z1'

4.5 PROGRAMACIÓN DEL PLD Y DIAGRAMA ESQUEMÁTICO COMPLETO DEL JUNCTION BOX

La programación del PLD se hace mediante un paquete de software especializado de Altera. En ese paquete se utiliza un lenguaje de descripción de hardware para definir funciones booleanas y máquinas de estados síncronas sin necesidad de minimizarlas.

Las ecuaciones y máquinas se escriben en un archivo de texto (JUNCTBOX.TDF); el software compila el archivo, las minimiza y genera varios archivos con distintas extensiones, entre los cuales destacan tres: uno de información de la optimación de las funciones (JUNCTBOX.PRN), otro de análisis de temporizado (JUNCTBOX.TAO) y otro para grabar el código al PLD (JUNCTBOX.POF). Con la definición del PLD culmina el diseño del hardware.

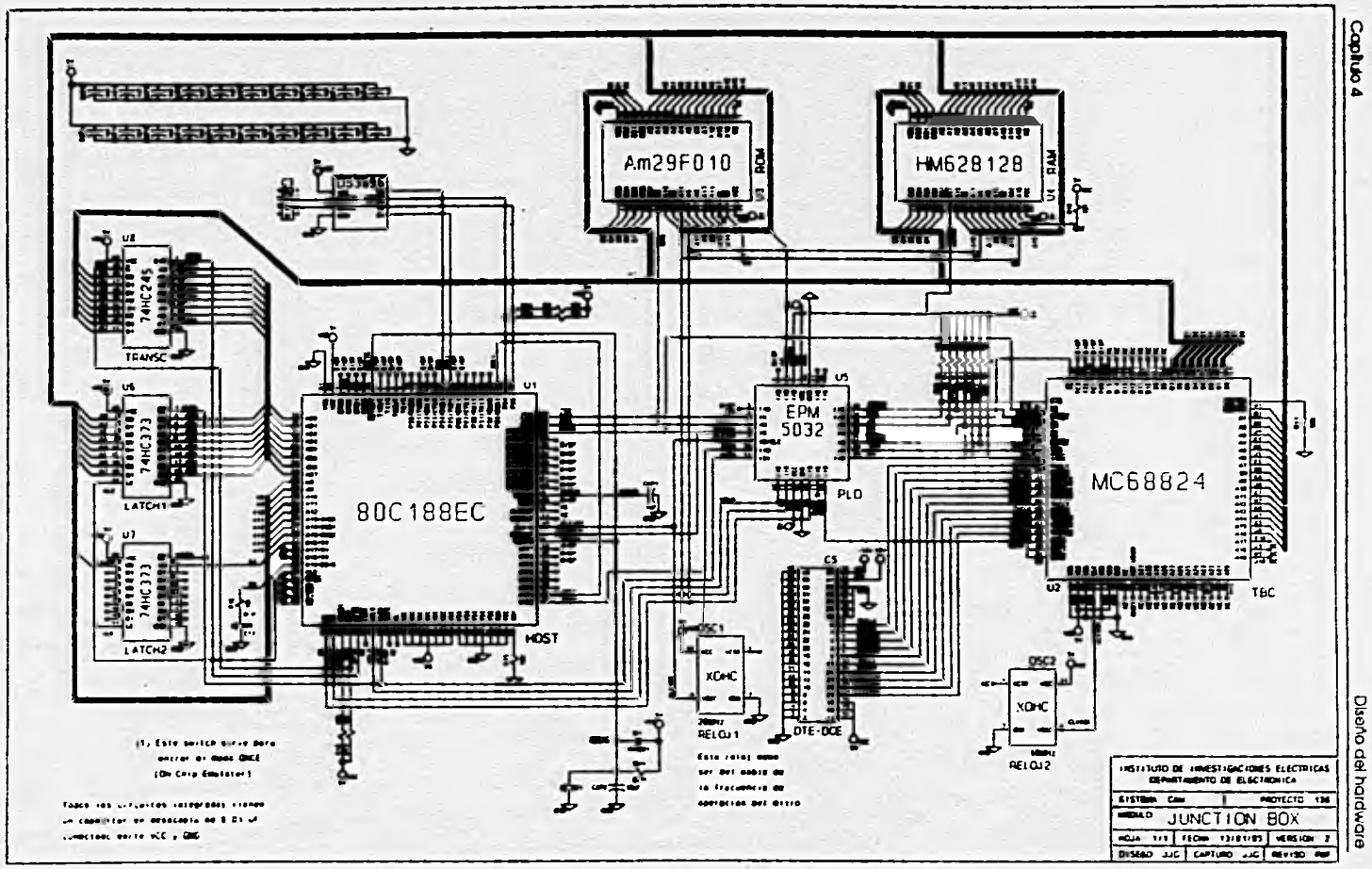
En seguida se presenta el archivo JUNCTBOX.TDF y al final, el diagrama esquemático completo del junction box.

Archivo JUNCTBOX.TDF

```
TITLE          "JUNCTION BOX";
DESIGN IS      "JUNCTBOX"
DEVICE IS EPM5032J-1

%=============================================================%
%                          EPM5032-w                          %
%              El sufijo depende de la velocidad              %
%              y temperatura deseadas, con las                %
%              opciones -15,-17,-20 y -25.                    %
% Notas:                                                      %
% Diagrama del sistema: JUNCTBOX.SCH en PCAD                  %
%=============================================================%

        BEGIN

        % Señales del 80C188 %
        Ard     @       18 : BIDIR;
        Awr     @       13 : BIDIR;
        hold    @       12 : OUTPUT;
        hlda    @       11 : INPUT;
        resout  @       6 : INPUT;
        /gcs0   @       7 : INPUT;
        clk188  @       9 : INPUT;
        Ale     @       8 : INPUT;
        Int0    @       10 : OUTPUT;

        % Señales del MC68824 TBC %
        As      @       23 : INPUT;
        Ablack  @       25 : BIDIR;
        r/w     @       24 : BIDIR;
        Ar      @       22 : INPUT;
        Arq     @       20 : INPUT;
        Agack   @       21 : INPUT;
        Ag      @       19 : OUTPUT;
        Abec    @       18 : OUTPUT;
        Ale     @       26 : OUTPUT;

        % Señales restantes %
        Acerem  @       2 : OUTPUT;

        END;

%=============================================================%
% El PLD contiene varios sistemas independientes que se denominarán %
% subsistemas. Los subsistemas son:                           %
%       1) cerem_and    Es la AND entre ALCS y /AS.           %
%       2) r/w_tbc      Es la separación de R/W en dos señales, %
%                       /RD y /WR, para controlar las memorias. %
%       3) interrupt    Es la negación de la señal /ARQ para  %
%                       conectarla a INT0.                    %
%       4) bus_esc      Es la negación de RESOUT para adaptarla a %
%                       la lógica negativa de /BEC2:0.        %
%       5) H188_tbc     Es una ASIM para adaptar las señales entre %
%                       los dos procesadores.                 %
%       6) bus_arbit    Es una máquina algorítmica asincrona para %
%                       el arbitraje del bus.                 %
%       7) as_dlack     Es la generación de Ablack a partir de As %
%=============================================================%

        SUBDESIGN jundbox
        (
        % cerem_and %
                Ale,As          : INPUT;
                Acerem          : OUTPUT;
        % r/w_tbc %
                hlda            : INPUT; % También se usa /as %
                r/w             : BIDIR; % En este caso es INPUT %
```

```
        /rd,/wr         : BIDIR;
% interrupt %
        /irq            : INPUT;
        int0            : OUTPUT;
% bus_err %
        resout          : INPUT;
        /bec            : OUTPUT;
% i186_tbc %
        clk186,/gcs0    : INPUT;  % También utiliza resout %
        /ca             : OUTPUT; % También se utilizan r/w,/rd y /wr %
% bus_arbt %
        /br,/bgack      : INPUT; % También se utiliza hida %
        /bg,hold        : OUTPUT;
% as_dtack %
        /dtack          : OUTPUT; % También se utiliza /as %
)

VARIABLE

%================%
% máquina i186_tbc %
%================%
es1: MACHINE OF BITS (q[2..1])
                WITH STATES(s0 = B"00",
                            s1 = B"01",
                            s2 = B"11",
                            s3 = B"10");
read_n, cs_n: NODE;

%========================%
% máquina bus_arbt asíncrona %
%========================%
z[2..1]: MCELL;

%=============================================================%
%       Empieza la definición de ecuaciones del PLD           %
%=============================================================%

BEGIN

DEFAULTS

hold = GND;
/bg = VCC;
read_n = VCC;
cs_n = VCC;

END DEFAULTS;

%==========%
% carem_and %
%==========%

        /carem = /ca & /as;

%========%
% r/w_tbc %
%========%

        /rd = TRI(!r/w # /as, hida);
        /wr = TRI( r/w # /as, hida);

%=========%
% interrupt %
%=========%

        int0 = !/irq;
```

```
%========%
% bus_esc %
%========%

        /bec = !resout;

%========%
% as_dtack %
%========%

        /dtack = TRI(/as,hide);

%========%
% i186_tbc %
%========%

        as1.clk = clk186;
        as1.reset = resout;
        r/w = TRI(DFF(read_n, clk186, VCC, VCC), !hide);
        /cs = DFF(cs_n, clk186, VCC, VCC);

        CASE as1 IS
                WHEN s0 =>
                        read_n = VCC;
                        cs_n = VC;
                        IF /gcs0 then
                                as1 = s0;
                        END IF;
                        IF !/gcs0 then
                                as1 = s1;
                        END IF;

                WHEN s1 =>
                        read_n = /wr;
                        cs_n = VCC;
                        as1=s2;

                WHEN s2 =>
                        read_n = /wr;
                        cs_n = VCC;
                        as1=s3;

                WHEN s3 =>
                        read_n = /wr;
                        cs_n = GND;
                        IF !(/rd & /wr) then
                                as1 = s3;
                        END IF;
                        IF  (/rd & /wr) then
                                as1=s0;
                        END IF;

        END CASE;

%========%
% bus_arbt %
%========%

        z2 = ((z2 & z1) # (z1 & hide) # (z2 & !/bgack)) & !resout;
        z1 = (((z2 & !/br & !hide) # (!z2 & z1) # (z1 & /as)) & !resout;
        hold = z1 # z2;
        /bg = !z1 # !z2;

END;
```


Am29F010
ROM
HM628128
U4 RAM
74HC245
TRANSC
74HC373
LATCH1
74HC373
LATCH2
80C188EC
HOST
EPM
5032
PLD
MC68824
TBC
XOHC
RELOJ1
DTE-DCE
XOHC
RELOJ2
INSTITUTO DE INVESTIGACIONES ELECTRICAS
DEPARTAMENTO DE ELECTRONICA
JUNCTION BOX

BIBLIOGRAFÍA DE REFERENCIA

Michael Slater
Microprocessor-based Design
Mayfield Publishing Company
California, USA 1987

Morris Mano
Diseño Digital
Prentice Hall Hispanoamericana
Edo. de México, México 1987

Earl W. Swokowsky
Cálculo
Wadsworth International Iberoamérica
Massachusetts, USA 1982

MC68824 Token Bus Controller User's Manual
Motorola Inc. 1987

80C188EC User's Manual
Intel
Order #272047

5. DISEÑO DEL SOFTWARE

5.1 ¿POR QUÉ LENGUAJE C?

En el IIE se desarrolló un software en lenguaje C para controlar el MC68824 Token Bus Controller[1] . Los requerimientos que se consideraron para la selección del lenguaje fueron

- capacidad de hacer operaciones a nivel bit
- manejo de estructuras de información
- manejo de rutinas de interrupción

Para el junction box los requerimientos son idénticos, así que se continuó utilizando lenguaje C para desarrollar el software. Pero ¿por qué C?

Lenguaje C es un excelente lenguaje para desarrollar sistemas por dos razones. En primer lugar es un lenguaje relativamente de bajo nivel que permite especificar con detalle la lógica de un programa, de manera que se logre el máximo rendimiento de un sistema. En segundo lugar es un lenguaje estructurado relativamente de alto nivel, que oculta al usuario la programación en lenguaje ensamblador pero logra el máximo rendimiento de un microprocesador.

En C pueden realizarse operaciones a nivel bit y especificarse direcciones físicas, se pueden manejar estructuras para el fácil manejo de información relacionada y se pueden manejar rutinas de interrupción en forma eficiente.

El compilador utilizado fue el IC-86 de Intel, que permite ubicar los programas en direcciones específicas de memoria y desarrollar programas en sistemas donde no existe sistema básico de entrada/salida (bios) o sistema operativo.

5.2 PRUEBAS INICIALES AL 80C188EC

El primer paso fue probar el subsistema del 80C188EC mediante un programa sencillo que permitiera observar señales para verificar el funcionamiento del microprocesador. La idea fundamental fue enviar por un puerto paralelo los bytes Ox1O y OxOO alternativamente, para observar en la terminal de salida P3.4 una señal cuadrada. El programa fue escrito en lenguaje ensamblador y se empleó el ambiente

---

[1] Fco. Javier Figueroa Velasco
Diseño y desarrollo del software para el controlador de token bus MC68824
Tesis profesional de ingeniero en comunicaciones y electrónica para el IPN
México, D.F. 1993

de programación de Intel para obtener el código que se graba en una EPROM. En seguida se encuentra el código fuente de dicho programa.

```
        MOV     DX,FF46H                ; selección del registro Port 3 Direction
        MOV     AX,0000H
        OUT     DX,AX
        MOV     DX,FF4CH                ; selección del registro Port 3 Mux Control
        OUT     DX,AX                   ; selección del Puerto 3 como puerto paralelo
        MOV     DX,FF4EH                ; selección del registro Port 3 Latch
LOOP:   MOV     AX,0010H                ; loop en el que se genera una señal cuadrada
        OUT     DX,AX                   ; en el pin P3.4 del microprocesador
        MOV     AX,0000H
        OUT     DX,AX
        JMP     LOOP
```

Una vez comprobado que el procesador realiza correctamente los ciclos de lectura a ROM (el *fetch* de instrucciones y direccionamiento directo) y la salida a un puerto paralelo, se hizo un programa para probar los puertos serie. Este programa fue muy importante porque, una vez comprobado el funcionamiento de los puertos serie, se podría utilizar un programa de ayuda para depuración del hardware –del cual se hablará posteriormente–. Para esta prueba se alambró a la salida de los puertos un driver para RS-232. El programa de prueba fue el siguiente:

```
        MOV     DX,FF5CH                ; selección del registro Port 2 Mux Control
        MOV     AX,00001111B            ; selección de los pines P2.4 a P2.7 como puerto serie (SCU0)
        OUT     DX,AX
        MOV     DX,FF58H                ; selección del registro Port 2 Direction
        MOV     AX,00H
        OUT     DX,AX                   ; selección del puerto como salidas
        MOV     DX,FF60H                ; selección del registro SCU0 Baud
        MOV     AX,8032H                ; usar reloj interno y dividirlo entre 0x32 para obtener 19.2 kbauds
        OUT     DX,AX
        MOV     DX,FF64H                ; selección del registro SCU0 Control
        MOV     AX,0001000001B          ; 8 bits de datos, no paridad, receptor deshabilitado, /CTS deshabilitado
        OUT     DX,AX

LOOP    MOV     DX,FF6AH                ; selección del registro SCU0 TxBuffer
        MOV     AX,'j'
        OUT     DX,AX                   ; transmisión de una 'j' al CRT
NOT_RDY         MOV     DX,FF66H        ; selección del registro SCU0 Status
                IN      AX,DX           ; se los al registro
                TEST    AX,08H          ; se comprueba si ha terminado la transmisión con TxEmpty
                JEZ     NOT_RDY         ; se espera hasta que la transmisión se complete

        MOV     DX,FF6AH                ; se hace lo mismo que arriba pero para transmitir una 'e'
        MOV     AX,'e'
        OUT     DX,AX                   ; transmisión de una 'e'
NOT_RDY2        MOV     DX,FF66H
```

```
        IN      AX,DX
        TEST    AX,08H
        JEZ     NOT_RDY2
JMP     LOOP                    ; se transmiten una 'j' y una 'e' indefinidamente
```

El resultado de este programa fue observar en un monitor o CRT (Cathode-Ray-Tube) 'jejejejeje...' indefinidamente. Una vez probado el puerto serie 0, se modificó el programa para probar el puerto serie 1. Para todo el desarrollo posterior se siguió utilizando el driver de RS-232.

Los dos programas de prueba anteriores sirvieron para localizar y corregir errores en el diseño del hardware referentes a señales que estaban desconectadas. Por ejemplo, la entrada NMI (Non-Maskable Interrupt) necesita una resistencia de pull-up si no se desea utilizar, para evitar que se detecte una interrupción por ruido. Otras señales que necesitaron pull-up fueron P3.4, P3.5 y /TEST; y la señal PDTMR necesitó un capacitor de 1nF.

5.3 MONITOR DE PRUEBAS DEL JUNCTION BOX

El Monitor de Pruebas es un programa desarrollado en el IIE escrito en lenguaje PLM86, que tiene rutinas para auxiliar la depuración de hardware y software en un sistema con microprocesador de la familia 8Ox86 de Intel.

Para poder utilizar dicho programa en el junction box fue necesario modificar las rutinas que manejan los puertos serie y el controlador de interrupciones. Las rutinas para el puerto serie hechas en PLM86 son muy similares a las de saca_carácter y obten_dato hechas en lenguaje C para el programa principal[2]. El controlador de interrupciones solamente fue deshabilitado.

Una vez hechas las modificaciones, se grabó en una EPROM el Monitor de Pruebas del junction box que permite, entre otras cosas, recibir programas por el puerto serie, grabarlos en RAM y ejecutarlos. Esta función fue la llave para el desarrollo del software del junction box.

La pantalla que se presenta al usuario es la siguiente

[2]Ver apéndice C, archivo LIB_JBOX.C

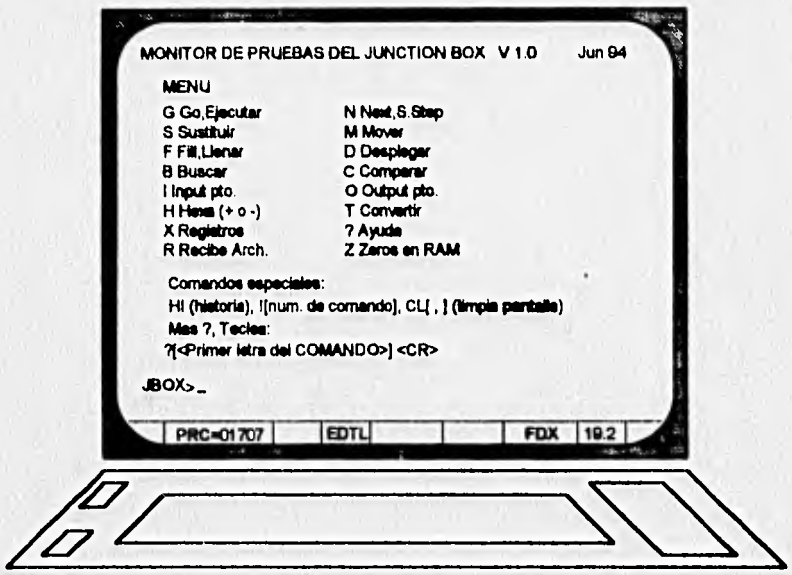


Fig. 5.1 Funciones del monitor de pruebas

Las funciones más importantes son

- recepción de archivos (R): recibe programas por el puerto serie desde una PC
- ejecutar (G): permite correr programas en el sistema
- sustituir (S): permite escribir datos directamente a la memoria
- desplegar (D): muestra el contenido de memoria
- registros (X): muestra los valores de los registros en un instante del programa
- input pto. (I): permite la lectura de puertos
- output pto. (O): permite la escritura a puertos
- next (N): permite correr paso a paso un programa

Utilizando el Monitor de Pruebas se volvieron a ejecutar los dos programas de prueba –señal cuadrada en el pin P3.4 y 'jejeje...' en el CRT–, mediante las opciones de recibir programas y ejecutarlos. Además, para probar el buen funcionamiento de la RAM en su totalidad, se hizo otro programa que escribiera datos en las 128k direcciones de memoria y después los leyera, comparando lo escrito y lo leído. Con ese último programa se dio por validado el funcionamiento del subsistema 80C188EC.

El Monitor de Pruebas del Junction Box se siguió usando durante todo el desarrollo del software.

5.4 PRUEBAS AL TBC

Para probar el subsistema TBC se hizo una rutina que utiliza un comando llamado Host Interface Test (prueba de la interfaz con el procesador principal), que accesa al TBC como esclavo y provoca que éste solicite el bus, lo tome, lea un grupo de datos, los copie en otra dirección de memoria y finalmente libere el bus. Este programa se ejecutó grabando el código y los datos en la memoria mediante el Monitor de Pruebas. La rutina de prueba se presenta en seguida.

Se define como área privada del TBC de la dirección 1000:0000 a la 1000:007F (128 bytes) en RAM. La tabla de inicialización se ubica de la dirección 1000:0080 a la 1000:017F (256 bytes).
El área privada del TBC se limpia con ceros. En la tabla de inicialización se escribe lo siguiente:

Dirección:	Dato:	Comentario:
1000:0080	00H	Cero
1000:0081	00H	(2 bytes)
1000:0082	00H	Apuntador
1000:0083	01H	al área
1000:0084	00H	privada
1000:0085	00H	(4 bytes)
1000:0086	00H	Cero
1000:0087	00H	(2 bytes)
1000:0088	1FH	HI-priority
1000:0089	FFH	(2 bytes)

A partir de la dirección 1000:0180 se escriben 34 datos (desde el 0 hasta el 33, es decir 00H hasta 21H), llegando hasta la dirección 1000:01A1.

Se realiza la siguiente escritura a puertos:

```
Output pto.     2000H,FFH     ; Comando "Reset"
Output pto.     2000H,D0H     ; Comando "Load Initialization Table Function Code" (8 bits de datos)
Output pto.     2004H,00H     ; Dirección inicial
Output pto.     2005H,01H     ;  de la tabla de
Output pto.     2006H,00H     ;  inicialización
Output pto.     2007H,80H     ;  (4 bytes)
```

Se inicializa el TBC:

```
Output pto.     2000H,C/H     ; Comando "Initialize"
```

Se revisa que el TBC inicialice el área privada. Se verifica el CPA result en la tabla de inicialización (offset 9CH, dirección 1000:0080 + 9C = 1000:011C); si no hubo errores cambió a 01H. Se limpia el CPA result escribiendo 00H en la memoria.

Se hace la prueba:

```
Output pto.     2000H,BCH     ; Comando "Host Interface Test"
```

Verificar el CPA result. Si no hay errores, se revisa que el TBC haya copiado los 34 datos en la dirección

1000.01A2. Si los datos están correctos, ¡el TBC funciona!

Con esta rutina se localizó una contención en el bus al observar que el grupo de datos era direccionado por el TBC pero escrito por el 80C188EC. Esto ocurrió porque no se consideró que la señal /DEN del microprocesador toma el estado de alta impedancia al ceder el bus y necesita una resistencia de pull-up, ya que controla el driver de tercer estado de los datos –74HC245–. También se localizó el *hazard* esencial mencionado en el capítulo 4.

Una vez que funcionó esta rutina se dio por validado el funcionamiento del subsistema TBC y en consecuencia, del junction box.

5.5 LIBRERÍAS E INTERRUPCIONES PARA EL JUNCTION BOX

El software se dividió en varios módulos, y éstos a su vez en subrutinas. Los principales módulos fueron

- Librería de rutinas básicas para el Junction Box
- Librerías para el TBC
- Librerías para cálculo numérico (manejo de variables y apuntadores)
- Librerías para entrada/salida
- Librerías para la memoria Flash

Primeramente se hizo una rutina para inicializar al junction box. Esta rutina contiene la programación inicial de la unidad de chip select para la ROM, la RAM y el TBC; la programación de los puertos serie a 19.2 kbauds, la programación de los controladores de interrupción o PICs, las rutinas básicas de entrada y salida de datos para los puertos serie y las rutinas para manejar interrupciones[3].

Con base al programa controlador del TBC para el CAM se obtuvieron unas rutinas de interrupción por software para poder utilizarlas en el junction box. Para ello, se clasificaron las rutinas existentes, se separaron en diferentes archivos, se crearon archivos de encabezamiento *(headers)* y de variables globales, y se recompilaron todos los archivos nuevamente. De este modo cualquier sistema con un TBC puede utilizar estas rutinas, siempre y cuando defina un archivo de inicialización[4].

Se contaba con librerías de cálculo numérico y de entrada/salida, desarrolladas en el IIE. Se incluyeron más rutinas a estos archivos.

---

[3] Ver apéndice C, archivos ETAS_JBOX.LIT y LIB_JBOX.C
[4] Ver apéndice C, archivo JBOX.INI

Con respecto a la memoria Flash, se desarrollaron rutinas para permitir la programación, el borrado por sectores y el borrado completo de la memoria. Estas rutinas no se incluyeron como librerías, sino como un programa aparte.

La descripción general de las librerías y rutinas de interrupción se detalla en las siguientes tablas.

LIB_JBOX.C

Rutina	Descripción	Notas
void ini_ jbox(void)	Programación de los registros del PCB para inicializar chip selects, puertos serie etc.	Flash: e0000h-fffffh (128 kbytes) RAM: 00000h-02000h (128 kbytes) TBC: 2000h-2064h (puertos) PICs: edge trigger, cascade mode, ICW4 nedded, normal EOI Interrupciones: externas 20h-27h, internas 28h-2fh (ICW2) SCU: 19.2 kbauds, 8 bits de datos, no paridad, receptor habilitado, pin CTS deshabilitado
void saca_caracter(unsigned char c)	Rutina básica para sacar datos por el puerto serie 0.	El registro de status se borra por completo al leerlo. Al revisar la bandera de transmisión debe guardarse la de recepción.
void saca_caracter1(unsigned char c)	Idem, puerto 1.	
unsigned char obten_dato()	Rutina básica para leer datos por el puerto serie 0.	
unsigned char obten_dato1()	Idem, puerto 1.	
void habilita(void)	Desenmascara las interrupciones.	Interrupción del TBC: 20h (master PIC, externa). Interrupción del slave PIC: 27h. Ver #define MASCARA y #define NO_MASC.
void deshabilita(void)	Enmascara las interrupciones.	
void fin_int(void)	Manda el comando de fin de interrupción al PIC.	Non specific EOI, IR level 0. Ver # define EOI.

TBC[5]

Rutina	Descripción	Notas
#pragma interrupt("inicializacion") void far inicializacion(void);	Comandos reset, load_ti, initialize, comandos al modem (setmode, physical, idle). Escritura de la Tabla de Inicialización y Área Privada, inicialización de apuntadores para Tx y Rx, inicialización de estructura para direcciones de los registros del TBC.	Ver manual del usuario MC68824 Token Bus Controller.
#pragma interrupt("transmision") void far transmision(void);	Transmite a la red el mensaje escrito en la dirección del apuntador para Tx.	mem_tx guarda el mensaje, mem_size el tamaño y mem_tx_prior la prioridad. Ver archivo de inicialización (JBOX.INI).
#pragma interrupt("recepcion") void far recepcion(void);	Escribe los mensajes recibidos por la red en la dirección del apuntador para Rx, ordenados por prioridad.	mem_rx guarda los mensajes, mem_prior_rx las prioridades de cada uno. Ver archivo de inicialización (JBOX.INI).
#pragma interrupt("host_test") void far host_test(void);	Realiza el comando Host Interface Test para el TBC.	Ver manual del usuario MC68824 Token Bus Controller.
#pragma interrupt("full_duplex") void far full_duplex(void);	Realiza el comando Full-Duplex Loopback Test para el TBC.	Ver manual del usuario MC68824 Token Bus Controller.
#pragma interrupt("cambia_dest") void far cambia_dest(void);	Elige la dirección destino de los mensajes.	
#pragma interrupt("tbc_int") void far tbc_int(void);	Atiende las interrupciones del TBC.	Escribe 0x88 en los offsets 90 y 92 del CPA (tabla de inicialización). Ver manual del usuario MC68824 Token Bus Controller.

UBCALC.C

Rutina	Descripción	Notas
unsigned long int int_mot_4(unsigned long int);	Cambia formato intel a motorola para datos tipo entero largo sin signo.	Rutina utilizada exclusivamente por el controlador de TBC.
unsigned int int_mot_2(unsigned int);	Cambia formato intel a motorola para datos tipo entero sin signo.	Rutina utilizada exclusivamente por el controlador de TBC.
unsigned long int valor_ptr(char far *);	Calcula el valor real de un apuntador.	
unsigned long int ptr_int(char far *);	Convierte un apuntador en un número entero largo.	
char far *int_ptr(unsigned long int);	Convierte un número entero largo en apuntador.	
void ptr_os(char far *, unsigned *, unsigned *);	Obtiene offset:segment de un apuntador.	
char far *os_ptr(unsigned *, unsigned *);	Obtiene un apuntador a partir de offset:segment.	

[5]Archivos PQ*.C en la tesis Diseño y desarrollo del software para el controlador de token bus MC68824 (ver bibliografía al final del capítulo)

LIBIO.C (rutinas para el CRT –tubo de rayos catódicos–)

Rutina	Descripción	Notas
int far imprime(char *,int);	Transmite cadenas de caracteres.	
int far imprime1(char *,int);	Idem, puerto 1.	
unsigned char obten_num(void);	Obtiene un número decimal de su equivalente ASCII.	Si se teclea '0' (ASCII 48) se obtiene 0x00.
unsigned char obten_ascii(void);	Obtiene un byte de su carácter ASCII equivalente.	Si se teclea 'a' (ASCII 97) se obtiene 0x0a.
unsigned int obten_num4(void);	Obtiene de uno a cuatro caracteres ASCII con posibilidad de corrección.	No importa si se teclean mayúsculas o minúsculas. Para corregir se utiliza Back Space (BS, ASCII 08).
void seca_byte(unsigned long int);	Imprime los caracteres ASCII de un byte.	Si el byte es 0x05 se obtiene '05' (ASCII 48 y ASCII 53).
void seca_int(int);	Idem para enteros.	
void seca_lint(long int);	Idem para enteros largos.	
void seca_ptr(char far *);	Imprime un apuntador.	Si la dirección es 0xfe000 se obtiene 'fe000' (ASCII 102,101,48,48,48)
void mostrar_mem(char far *,unsigned int);	Despliega el contenido de memoria en bloques de 128 bits.	Esta rutina tiene una opción para incrementar por un múltipio el siguiente bloque a desplegar, definido por el argumento unsigned int.
void pausa(void);	Provoca una pausa en el programa.	Imprime "Pulse una tecla para continuar..."

FLASH.C

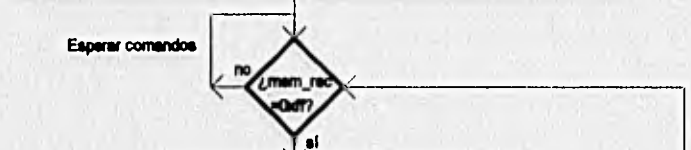
Rutina	Descripción	Notas
void sector_protection()	Informa qué sectores están protegidos.	
void program()	Permite la programación de la Flash.	Variables principales: número de bytes, dirección fuente y dirección destino.
void chip_erase()	Borra toda la memoria.	
void sector_erase()	Borra la memoria por sectores.	La indicación de los sectores (*ptr=0x30) es un tiempo crítico

5.6 PROGRAMA PRINCIPAL

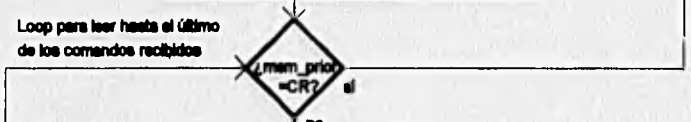
El programa principal está contenido en el archivo JBOX.C[6]. Este programa es general para cualquier tipo de dispositivos de adquisición de datos. Su estructura es la siguiente

[6]Ver apéndice C

Proceso	Notas
Inicializar Junction Box	void ini_jbox()
Escribir vectores de interrupción	La asociación de vectores del slave PIC está establecida por Intel, ver #defines TMR0,TMR1,DMA2,DMA3,TMR2,SCU_RX0 y SCU_TX0 para la correspondencia
Programar fuentes internas de interrupción y desenmascarar interrupciones	Tmr2 (0000h-ffffh), Tmr0(0000h-00ffh) DMA3 (source: s1rbuf destination: mem_rx_red, UART1 request)
Inicializar TBC	pragma interrupt ("inicializacion")
Inicializar red de adquisición de datos	void inicia_red() con defaults
Limpiar banderas	mem_rec=0x00, mem_trans=0xff



Proceso	Notas
Leer mensajes	pragma interrupt ("recepcion") mem_rec=0x00



Proceso	Notas
Traducir comando	unsigned int lle_red(char far *), obtiene el tamaño esperado de la respuesta. Se le indica dónde está la adquisición.
Escribir prioridad del comando en proceso	mem_prior_bx
Preparar recepción por DMA si es necesaria	alza, programación de los registros D3CON y D3TC
Transmitir el comando a la red de adquisición de datos	mem_tx_red
Indicar adquisición de datos en proceso	mem_trans=0x00, se evita contención en la red de adquisición
Arranque de los timers (prevención de errores por falta de respuesta)	Siempre y cuando se espere recepción
Esperar que las interrupciones DMA3 o Tmr0 indiquen fin de adquisición de datos	Ambas interrupciones hacen mem_trans=0xff (ver rutinas de interrupción)
Detener timers	

Fig. 5.2 Estructura del programa principal JBOX.C

Las rutinas de interrupción realizan los siguientes procesos

Pragma interrupt ("DMA3')

Proceso	Notas
Reubicar el destino de DMA3.	Registros D3DSTL y D3DSTH.
Traducir la adquisición de datos.	unsigned int red_lie(), proporciona el tamaño del mensaje a transmitir por Token Bus, guardado en mem_size.
Transmitir por Token Bus en el protocolo apropiado.	pragma interrupt ("transmision").
Indicar fin de adquisición de datos.	mem_trans=0xff.

Pragma interrupt ("TmrO")

Proceso	Notas
Reubicar el destino de DMA3, reprogramar timers	Registros D3DSTL y D3DSTH; T0CON, T2CON, T0CMPA, T0CMPB, T2CMPA y T2CMPB
Escribir mensaje de error	
Transmitir por Token Bus	pragma interrupt ("transmision")
Indicar fin de adquisición de datos	mem_trans=0xff

5.7 EJEMPLO PARA DISPOSITIVOS ADAM

Las rutinas lie_red y red_lie se ilustran con un ejemplo para dispositivos ADAM. Debido a que no existe aún un protocolo para el IIE los comandos recibidos no sufren cambio, se envían a la red, se hace la adquisición de datos y se transmite la adquisición sin traducir. Para ello fue necesario simular los dispositivos ADAM en una PC[7], aprovechando el driver para RS-232 usado en el Monitor de Pruebas.

IIE_ADAM.C

Rutina	Descripción	Notas
unsigned char ascii_byte(char ,char);	Convierte dos caracteres ASCII en un byte.	Si se recibe '05' (ASCII 48 y ASCII 53) se obtiene el byte 0x05.
void inicia_red(void);	Define la base de datos de configuración de la red con defaults.	La base de datos se guarda en mem_red.
unsigned int iie_red(char far *);	Se analiza el comando para determinar el tamaño de la respuesta (se debe traducir el comando también)	El comando se transcribe pero si se analiza.
unsigned int red_iie(void);	Se determina el tamaño del mensaje a transmitir.	Los datos se envían sin modificaciones.

Con este programa se finalizó el desarrollo del junction box como traductor de

[7]Ver apéndice C, archivo ADAM.C

comandos y se comprobó su funcionamiento. Solamente faltó hacer la adquisición de datos en el protocolo RS-485, prueba que se hará hasta adquirir dispositivos inteligentes de adquisición de datos en el IIE.

BIBLIOGRAFÍA DE REFERENCIA

Les Hancock, Morris Krieger
Introducción al lenguaje C
McGraw-Hill
Edo. de México, México 1989

Fco. Javier Figueroa Velasco
Diseño y desarrollo del software para el controlador de token bus MC68824
Tesis profesional de Ingeniero en comunicaciones y electrónica para el IPN
D.F., México 1993

6. RESULTADOS Y CONCLUSIONES

6.1 RESULTADOS

Con respecto a la metodología empleada y a las herramientas de desarrollo se tienen los siguientes resultados:

- se obtuvo un sistema digital con aplicación en el sector industrial
- la utilización de PLDs en el sistema facilitó el desarrollo del hardware y permitió utilizar tecnología de vanguardia
- la inclusión de la sección análisis de temporizado es importante pues intenta ejemplificar la metodología que debe seguirse al desarrollar sistemas digitales
- el desarrollo del software forzó el aprendizaje de la programación en lenguaje C
- se propone la inclusión en los temarios de las materias "Diseño de sistemas con microprocesadores" y "Diseño de sistemas digitales" los siguientes subtemas:
  1. Análisis de temporizado en sistemas digitales
  2. Desarrollo de sistemas en base a PLDs
  3. Diseño y análisis de máquinas de estados asíncronas

Sería muy deseable adquirir en la UNAM software especializado para PLDs –presumiblemente de Altera–, lo que representaría mayor aprendizaje a los alumnos a un precio muy bajo, pues el PLD que adquirieran serviría para todos sus proyectos incluyendo el de tesis profesional.

6.2 CONCLUSIONES

Conviene recordar que el objetivo inicial de esta tesis fue desarrollar un subsistema de comunicaciones para una arquitectura de control distribuido en sistemas abiertos, que satisfaciera los requerimientos del modelo de interconexión de sistemas abiertos –OSI– para el nivel de ligado de datos (data link layer). Los requerimientos del modelo OSI se repiten nuevamente aquí

2. Ligado de datos (data link layer)	IEEE 802.2 LLC IEEE 802.4 Token Bus ISO 8802.4 Token Bus Realización en hardware (MAC), software (LLC) y firmware	Funciones: - inicio y fin de la liga de transmisión - sincronización y cuadratura de mensajes - control de flujo y secuencia - reconocimiento y corrección de errores - intercambio de parámetros e identificaciones - monitoreo de la conexión física - manejo del nivel de ligado de datos Servicios: - sección de transmisión - ligado de las unidades de datos del servicio - conexión del identificador de punto terminal en el ligado de datos - control de flujo y secuencia - reporte de errores - parámetros de la calidad del servicio

Mediante el subsistema TBC el junction box realiza la liga de transmisión con el nivel de red, controlando totalmente el acceso a ella. El subsistema TBC realiza la cuadratura y sincronización de mensajes, y controla el flujo y la secuencia de la información. Para ello se utilizó, según se propone en el modelo, el estándar IEEE 802.4 en la realización de los niveles MAC y LLC, cubriendo las necesidades de la subcapa de ligado de datos[1]. Trabajando en conjunto, el subsistema 80C188EC controla el flujo y la secuencia de los datos, reporta errores y realiza el ligado de las unidades de datos mediante la conversión de protocolos. El junction box es un sistema que tiene capacidad de recuperación ante errores, que lo hace ser un sistema muy recomendable para aplicaciones de control y automatización de procesos.

Por tanto, se puede concluir que el junction box satisface el objetivo inicial en forma cabal y es un sistema que realiza físicamente las funciones definidas por el modelo OSI para el nivel de ligado de datos. El junction box tiene aún capacidad para realizar tareas básicas de control utilizando sus puertos paralelos y las interrupciones restantes.

El junction box es un sistema que puede utilizarse en aplicaciones diferentes a las del sector eléctrico, siempre y cuando se esté utilizando un sistema abierto de control distribuido –que es la tendencia actual–. En los edificios inteligentes se utilizan sistemas de control de ese tipo.

Mediante esta tesis se ha resuelto una necesidad específica y actual en el campo del desarrollo de tecnología para el sector eléctrico, con aplicaciones directas a la industria. Se espera que muy pronto el sistema se instale en todas las plantas de

[1] Ver apéndice A

generación eléctrica del país y otras industrias que adquieran un controlador de acceso múltiple como el CAM.

6.3 EVALUACIÓN FINAL DEL SISTEMA

Al tratar de integrar todas las capacidades del junction box, se descubrió que es necesario añadir un módulo de EPROM al sistema para alojar el programa principal en ella. La memoria Flash no puede funcionar como EPROM porque cuando se borra algún sector de la memoria es necesario no accesarla por un tiempo determinado, sin embargo en el esquema actual el fetch de instrucciones se hace desde la misma Flash. En otras palabras, la memoria Flash no puede borrarse a sí misma.

La idea de tener una memoria programable en el campo es poder actualizar bases de datos a distancia. El proceso sería así: desde la unidad de control o de supervisión se mandaría a algún junction box un comando para actualizar o cambiar su base de datos de protocolos vía Token Bus –corresponde a las rutinas de lle_red y red_lle– ubicada en cierto sector de la Flash. Se borraría sólo el sector –o los sectores– que contenga la base de datos sin tener que reprogramar al sistema. Esta capacidad de programación vía red es importantísima pues se puede cambiar el programa en cualquier junction box sin necesidad de ir al campo a cambiar su ROM.

La Flash servirá para almacenar las bases de datos que especifican la configuración de la red de adquisición y los programas que permiten la traducción de protocolos, y todo podrá hacerse por medio de la red. Sin embargo, aún hace falta desarrollar el protocolo único del bus de campo (protocolo IIE en este caso) para incluir en el software las rutinas de borrado y programación de la Flash –las rutinas básicas forman parte de esta tesis[2]–.

Es necesario hacer aún varias decisiones referentes al hardware como son la inclusión del driver de RS-232 en el sistema final, el tamaño definitivo de las memorias y la utilización del otro puerto serie, que puede servir para soportar otra red de adquisición o para la supervisión posterior del junction box mediante un CRT (monitor).

---

[2] Ver apéndice C, archivo FLASH.C

APÉNDICE A. ESTÁNDAR IEEE 802.4

A.1 RELACIÓN ENTRE EL ESTÁNDAR IEEE 802.4 Y EL MODELO OSI

El protocolo IEEE 802.4 define tres capas de control en donde el modelo OSI define sólo dos. Esto se ilustra en la figura A.1.

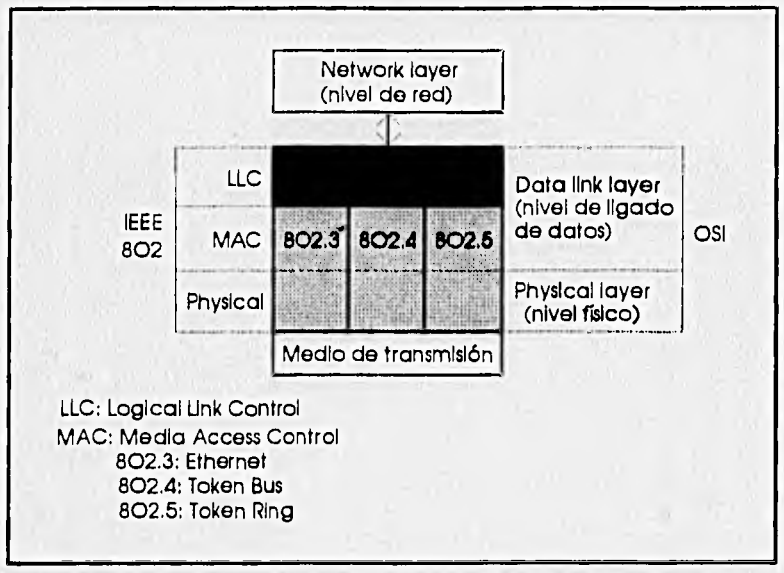


Fig. A.1 Estándar IEEE 802 y su relación con el modelo OSI

A.2 SUBCAPAS DEFINIDAS POR EL ESTÁNDAR IEEE 802.4

- Subcapa MAC (Media Access Control)

No importando el estándar utilizado en esta subcapa –Ethernet, Token Bus o Token Ring– existen una serie de servicios definidos para el usuario que contienen ciertos parámetros asociados. Estos son:

1. MA.DATA.request. Este servicio contiene la dirección destino –individual, de grupo o de red–, una unidad de servicio de datos que contiene la información a transmitir –llamada LLC-PDU– y la clase de servicio asociado con el PDU –prioridades–.

2. MA.DATA.indication. Este servicio no contiene parámetros asociados, y sirve únicamente para el intercambio de señales o *handshake*.
3. MA.DATA.confirmation. Este servicio incluye un parámetro para evaluar la solicitud hecha por una sentencia MA.DATA.request como satisfactorio o fallido. Este parámetro no se genera en la capa LLC de otra estación, sino como un servicio de MAC local, e indica si la transmisión se realizó correcta o incorrectamente hacia la red.

• Subcapa LLC (Logical Link Control)

Esta capa está basada en un protocolo de ligado de datos de alto nivel llamado HDLC. Contiene dos servicios para el usuario:

1. L_DATA.Indication (N_PDU). Es sólo la indicación de que se desea el servicio.
2. L_DATA.request (N_PDU). Este servicio tiene asociados los parámetros de las direcciones fuente y destino, y la unidad de servicio de datos que contiene la información a transmitir. Este último es conocido como unidad de protocolo de datos del nivel de red o N_PDU (Network layer Protocol Data Unit). Las direcciones fuente y destino son una concatenación de las direcciones recibidas por la capa MAC más un parámetro llamado punto de acceso al servicio o LLC-SAP (Service Access Point) usado por las terminales o DTEs (Data Terminal Equipment) para funciones de establecer rutas entre capas.

Existen dos tipos de protocolo para el usuario: sin-conexión y orientado a la conexión. En la práctica, en la mayoría de las LANs se utiliza un protocolo de sin-conexión del tipo "manda datos sin reconocimiento" o SDN (Send-Data-with-No-acknowledge), que significa que se prescinde de la directiva L_DATA.Indication y la transmisión se realiza solamente mediante directivas L_DATA.request.

A.3 TRANSMISIÓN DE MENSAJES

• Interacción entre las capas MAC y LLC

La forma de mandar mensajes desde una DTE hacia otra, mediante los servicios de las distintas subcapas definidas por el estándar IEEE 802, se describe en la figura A.2.

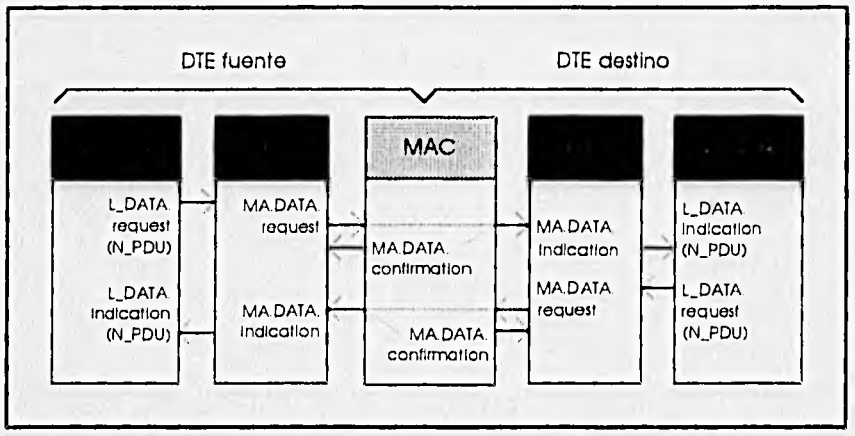


Fig. A.2 Transmisión de mensajes mediante el estándar IEEE 802

En la figura A.3 se puede observar a detalle cada uno de los procesos que sufre el mensaje en cada subcapa, desde su concepción en el nivel de red hasta lograr la transmisión a la red física.

Cuando se desea enviar un mensaje, se transmiten los datos del usuario y la directiva de solicitud del servicio (L_DATA.request) hacia el nivel de ligado de datos (Data link layer). En la subcapa LLC se leen, a partir de la directiva, las direcciones destino y fuente de acceso al servicio (DSAP y SSAP) que se concatenan a las direcciones destino y fuente (DA y SA) en la red. Después se añade la clase del servicio (Service class), el tamaño del mensaje (Length indicator) y el campo de bits del usuario (N-PDU) que proviene del nivel de red (Network layer).

Hecho lo anterior, se hace una solicitud de servicio de la subcapa MAC (MA.DATA.request) cuyo campo de datos del usuario es el que proviene de la subcapa LLC, es decir el campo de datos original más los campos añadidos por dicha subcapa (LLC-PDU). A partir de la directiva de solicitud, se leen las direcciones destino y fuente (DA y SA) y la clase del servicio (Service class); y se añaden el indicador de longitud del mensaje (Length indicator) y el nuevo campo de datos del usuario (LLC-PDU). Una vez obtenido este 'cuadro de bits' (frame), la subcapa MAC añade campos de bits al mensaje según lo necesite el tipo de bus de la red (Ethernet o Token) y finalmente se transmite la información hacia la red.

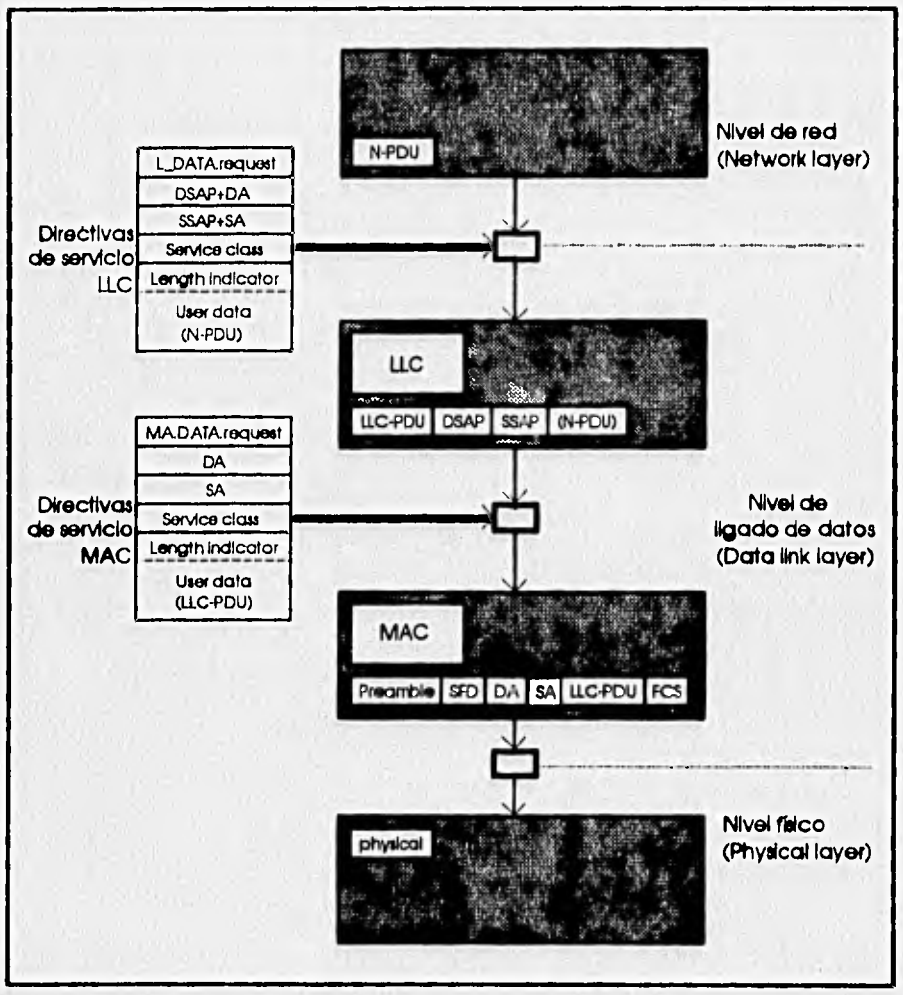


Fig. A.3 Transmisión de un mensaje desde el nivel de red hasta el físico

APÉNDICE B. FUNCIONAMIENTO DE UNA RED TOKEN BUS

B.1 CODIFICACIÓN

El tipo de codificación que se utiliza en este tipo de redes es *carrierband* o portadora. Los estados lógicos se representan con voltajes senoidales: el uno lógico a cierta frecuencia (normalmente 5 Mbits/s) y el cero lógico al doble de la frecuencia del uno lógico. Al transmitir unos y ceros lógicos las senoidales no sufren cambio de fase ni de amplitud, sólo de frecuencia. La gran ventaja de este tipo de decodificación es que se tiene gran inmunidad al ruido, pues es posible añadir dos filtros en el receptor y quitar las demás frecuencias añadidas por ruido –cosa que no se puede hacer ante decodificaciones que manejan señales cuadradas, pues al filtrar el ruido se filtra también la señal–. Las señales se transmiten por medio de cable coaxial. La modulación, demodulación y generación del reloj está a cargo de un módulo de interfaz física *(modem)* que antecede al control del medio de acceso (MAC layer). En la figura B.1 se muestra la configuración física de la red y la codificación por carrierband.

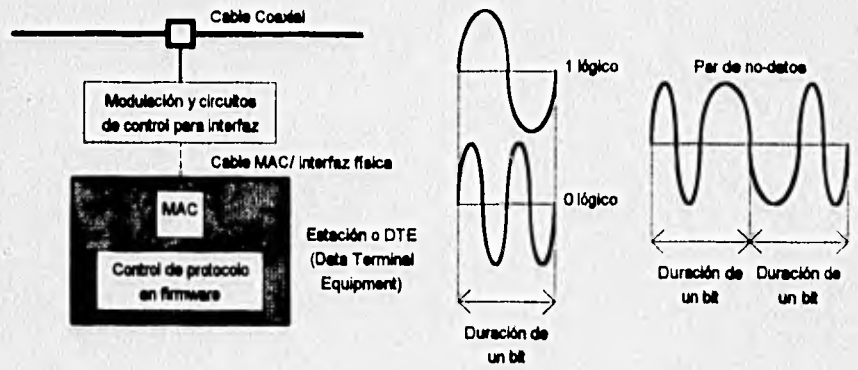


Fig. B.1 Interfaz física y codificación carrierband

B.2 FORMATO DE LOS CAMPOS DE BITS

El formato de los campos de bits o *frames* es el mostrado en la figura B.2.

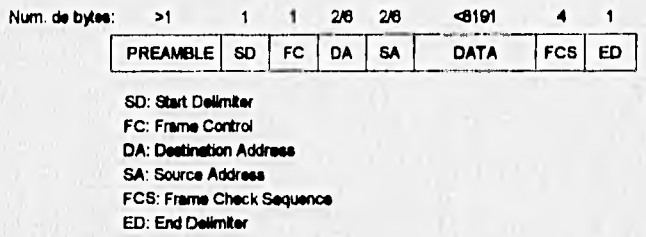


Fig. B.2 Formato de un *frame* para Token Bus

Los campos son los siguientes:

SD: delimita el comienzo del frame. Su forma es

J	K	0	J	K	0	0	0

donde J y K son un par de no-datos como los mostrados en la figura B.1.

FC: control del frame. Su forma es

F	F	Z	Z	Z	Z	Z	Z

donde F indica si el frame es de tipo MAC (para control de la red) y Z son los bits de control.

DA y SA: son las direcciones destino y fuente. Pueden constar de 16 o 48 bits (uno de los dos valores para toda la red). Su forma es

I/G	15/47 bits

donde I/G indica si la dirección es individual (I/G=0) o de grupo (I/G=1).

DATA: es el campo de datos. Puede constar de hasta 8191 bytes.

hasta 8191 bytes

FCS: son 4 bytes para checar errores utilizando código CRC (Ciclic Redundance Check).

4 bytes CRC

ED: delimita el fin del frame. Su forma es

J	K	1	J	K	1	1	E

donde J y K son un par de no-datos como en el caso de SD, y E es un bit para indicar si hubo errores en la transmisión.

B.3 PRINCIPIO DE OPERACIÓN

El principio de operación es como sigue: sólo existe un *token* de control y la estación que lo posea es la única que puede transmitir campos de bits o *frames*. Todas las estaciones o DTEs (Data Terminal Equipment) que pueden hacer uso de la red están ligadas en la forma de un anillo lógico y el token se pasa físicamente por medio de un bus. Al recibir una estación el token proveniente de su estación antecesora, puede transmitir sus *frames* –hasta un máximo– y después debe pasar el token a su estación predecesora.

Es necesario recalcar dos propiedades de la red Token Bus. Primero, que como todas las estaciones están conectadas al mismo bus, al transmitir un mensaje éste es recibido por todas las estaciones activas de la red. Segundo, que existe un tiempo máximo de espera de respuesta llamado slot time, que considera el retardo que puede sufrir la señal al propagarse en la red (transmisión y recepción) y el tiempo máximo que una estación tarda en generar una respuesta. Este tiempo le da la característica de red determinística, y está definido por

Ts(slot time)=2xTd(propagación máxima)+Tp(retardo de la respuesta)

Bajo operación normal, el token es pasado de una estación a otra mediante un pequeño *frame*. La estación necesita solamente saber la dirección de la estación que le sigue para continuar el anillo lógico. Si una estación falla al aceptar el token la estación emisora realiza una serie de procedimientos para restablecer el anillo lógico. Existen otros procedimientos para cuando una estación quiere formar parte del anillo o salir de él, para establecer prioridades en los mensajes y para inicializar y reinicializar la red. Estos procedimientos se describirán en seguida.

B.3.1 Paso del token

Cuando una estación recibe el token puede transmitir los mensajes que tenga en cola de espera, y al terminar pasa el token a la siguiente estación. Después de enviar el *frame* del token, la estación emisora permanece escuchando esperando actividad en el bus para asegurarse que el token fue pasado correctamente. Si el siguiente *frame* es correcto, asume que el paso del token fue correcto. Si escucha ruido o un *frame* incorrecto espera hasta cuatro slot times. Si después de ese tiempo escucha un *frame* correcto, asume que la estación sucesora tiene el token. Si no se

escucha nada en el bus, la estación asume que el token fue corrompido y lo retransmite. Si nuevamente falla el paso del token después de monitorear el bus, la estación emisora supone que la siguiente estación ha fallado y trata de restablecer la red pidiendo un nuevo sucesor (new successor). Para ello, la estación emite un *frame* llamado who-follows-me (¿quién me sigue?) que contiene la dirección de su estación sucesora. La estación cuya antecesora fuera la estación que falló responde a la directiva mandando su dirección, para que la estación que está tratando de restablecer el anillo la grabe como su nueva estación sucesora y saltar a la estación que falló.

Si la estación no recibe respuesta a la directiva **who-follows-me**, emite otra directiva llamada solicit-successor (¡solicito sucesor!) mediante un *frame* que contiene como dirección destino su propia dirección. Varias estaciones pueden contestar a esta directiva, si esto sucede el anillo se restablece mediante un proceso llamado **response-window** (ventana de respuesta). Si no existe respuesta, la estación considera que ha habido una catástrofe: que las otras estaciones han fallado, que se ha roto el cable o que ha fallado su propia sección receptora. Entonces, la estación deja de transmitir y se dedica a escuchar esperando actividad en el bus.

B.3.2 Ventana de respuesta (**Response window**)

Este procedimiento se realiza en intervalos de tiempo arbitrario para permitir la entrada de nuevas estaciones a la red. La ventana de respuesta es el tiempo que tarda una estación en responder a un *frame*, es decir dura un slot time. Cada vez que se transmite un *frame* solicit-successor se especifican la dirección fuente y la destino (SA y DA). Las estaciones cuya dirección está en el rango entre SA y DA que desean entrar al anillo lógico responden con una directiva llamada set-successor (establece sucesor). Si sólo responde una estación, la estación emisora establece el anillo adoptando a la nueva estación como su sucesora. Es evidente que puede haber más de una estación en el rango, por lo que habrá varios *frames* en el bus al mismo tiempo. Para resolver el problema de contención se tiene otro procedimiento.

La estación emisora del solicit-successor comienza a enviar *frames* llamados resolve-contention hasta que la contención se resuelve, es decir, recibe *frames* correctos. Todas las estaciones que hayan contestado a solicit-successor –y que no recibieron el token, indicio de que hubo contención en el bus– comienzan a esperar de 0 a 3 slot times en forma arbitraria para mandar nuevamente un set-successor. Si una estación escucha actividad en el bus después de expirar su tiempo, renuncia a formar parte de

la red en ese momento y espera a una siguiente ventana de respuesta. Si la estación no escucha actividad en el bus, espera otro *frame* de resolve-contention para entrar a la red. El tiempo para resolver contenciones es limitado.

B.3.3 Inicialización

La inicialización del anillo es un procedimiento basado en el de ventana de respuesta. Cada vez que un DTE escucha actividad en el bus resetea un contador de inactividad (**inactivity timer**). Si este contador expira, la estación sabe que no existe token o se ha perdido y manda un comando llamado claim-token (demando token). Varias estaciones puede emitir el comando, pero sólo una debe generar el nuevo token. Para obtener una estación ganadora del token, la longitud del frame claim-token es de 0,2,4 o 6 slot times proporcionalmente a los primeros dos bits de dirección de la estación. Si una estación manda su comando y al terminar escucha actividad en el bus, sabe que otra estación con una dirección mayor intenta ganar el token y declina. Si la estación no escucha actividad en el bus, vuelve a mandar el **claim-token** con una longitud proporcional a los dos siguientes bits de su dirección. El proceso de escuchar en el bus y declinar o continuar se repite hasta que la estación con la mayor dirección gana el token. La estación que posee el token inicializa la red con un procedimiento de response-window.

B.3.4 Operación por prioridades

Las prioridades en Token Bus son llamadas *clases de acceso*. Sólo existen cuatro clases de acceso, que son

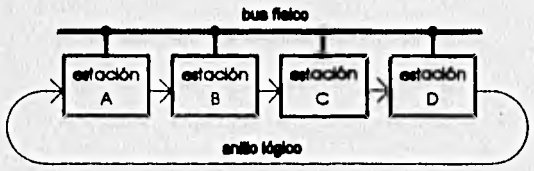
- Clase 6. Mensajes urgentes, como son alarmas y funciones de control críticas
- Clase 4. Mensajes referentes a acciones normales de control y mantenimiento del anillo
- Clase 2. Mensajes relativos a la recolección y transporte de datos
- Clase 0. Bajada de programas y transferencia general de archivos

Cada estación o DTE tiene dos contadores para controlar la transmisión de frames: el THT (Token Hold Timer) que le indica el tiempo que ha mantenido el token; y el HPTHT (High Priority Token Hold Timer) que le indica el tiempo que puede retener el token para transmitir mensajes de alta prioridad. El tiempo estipulado por el HPTHT es el mismo para todas las estaciones de la red; de esta forma se asegura que todas las estaciones compartan equitativamente la capacidad de la red para transmitir

mensajes de alta prioridad. Cada estación posee un contador para conocer el tiempo que ha transcurrido desde la última vez que recibió el token hasta la llegada del siguiente. Este tiempo se guarda en una variable llamada TRT (Token Rotation Time).

Cuando una estación recibe el token transfiere la variable TRT al contador THT, que comienza su cuenta regresiva. Al mismo tiempo, la estación transmite los mensajes de más alta prioridad (si existen) mientras el contador HP-THT alcanza su valor máximo. Después se compara el tiempo restante con una variable fija llamada TTRT (Target Token Rotation Time), que indica el tiempo máximo que una estación puede retener el token para transmitir mensajes de baja prioridad. Si la diferencia TTRT-TRT es positiva la estación transmite los mensajes de baja prioridad que esperaban en cola mientras el contador THT alcanza el valor de TTRT. Si la diferencia es cero o negativa, la estación no transmite ningún mensaje pendiente y pasa el token a la siguiente estación. Este mecanismo se ilustrará con un ejemplo.

Supóngase que todos los frames tienen la misma longitud para la red. Para cada estación sean HPTHT=3 –cada vez que una estación reciba el token podrá enviar tres frames de mensajes de alta prioridad– y TTRT=8 –cada estación podrá enviar un máximo de 8 frames de mensajes de baja prioridad–. Supóngase que las estaciones A y D transmiten sólo frames de alta prioridad (pueden hacerlo siempre) y que las estaciones B y C sólo de baja prioridad. Las estaciones B y C transmitirán siempre el número máximo de frames que puedan.



Rotación	TRT	XMIT	TRT	XMIT	TRT	XMIT	TRT	XMIT
1	0	0	0	0	0	0	0	0
2	0	3	3	5	8	0	8	3
3	11	3	11	0	6	2	8	1
4	6	0	3	5	8	0	6	0
5	5	3	8	0	3	5	8	etc.

TRT: Token Rotation Time XMIT: frames transmitidos

Se usará la notación n(Xm) al calcular TRT para indicar que se transmitieron *n* frames por la estación *X* en la rotación *m* del token.

En la primera rotación del token ninguna estación transmite mensajes, por lo que TRT=O. Para la segunda rotación del token, la estación A transmite tres frames de alta prioridad, por lo que TRT=3 para la estación B. Entonces, B puede transmitir 8-3=5 frames de mensajes de baja prioridad (TTRT-TRT). Para la estación C, TRT=3(A2)+5(B2)=8 por lo cual no puede transmitir mensajes de baja prioridad (8-8=O) y pasa el token a la estación D. Para ésta, TRT=3(A2)+5(B2)+O(C2)=8; a pesar de que no puede transmitir baja prioridad la estación C transmite 3 frames de alta prioridad.

En la tercera rotación, TRT=3(A2)+5(B2)+O(C2)+3(D2)=11 para la estación A, y transmite 3 frames de alta prioridad. Para B, TRT=5(B2)+O(C2)+3(D2)+3(A3)=11 y está imposibilitada para transmitir (8-11=-3). Para C, TRT=O(C2)+3(D2)+3(A3)+O(B3)=6 y transmite 8-6=2 frames de baja prioridad. Para D, TRT=3(D2)+3(A3)+O(B3)+2(C3)=8 pero transmite 1 frame de alta prioridad.

En la cuarta rotación las estaciones A y D no tiene frames por transmitir. Entonces para B, TRT=O(B3)+2(C3)+1(D3)+O(A4)=3 y puede transmitir 8-3=5 frames, mientras que para C, TRT=2(C3)+1(D3)+O(A4)+5(B4)=8 y no puede transmitir.

Para la quinta rotación A transmite 3 frames. Entonces para la estación B, TRT=5(B4)+O(C4)+O(D4)+3(A5)=8 y no puede transmitir; mientras que para la estación C, TRT=O(C4)+O(D4)+3(A5)+O(B5)=3 y transmite 5 frames.

Este proceso continúa para todas las rotaciones del token.

APÉNDICE C. LISTADOS DE LOS PROGRAMAS

C.1 ARCHIVO ETASJBOX.LIT

```
/*****************************************************************
  Etiquetas para el segmento Peripheral Control Block del i80C186EC
*****************************************************************/

/*  Direccion base del segmento PCB (valor por default: 0xff00) */

#define  dir_base_PCB   0xFF00

   /*Puertos serie del i80c186EC  */

#define  b0cmp      0x060
#define  s0cnt      0x062
#define  s0con      0x064
#define  s0sts      0x066
#define  s0rbuf     0x068
#define  s0tbuf     0x06A

#define  b1cmp      0x070
#define  s1cnt      0x072
#define  s1con      0x074
#define  s1sts      0x076
#define  s1rbuf     0x078
#define  s1tbuf     0x07A

   /* Mascaras de los puertos serie */

#define  rx_listo_scu          0x040
#define  tx_listo_scu          0x008


   /* Timers del i80C186EC */

#define  t0cnt      0x030
#define  t0cmpa     0x032
#define  t0cmpb     0x034
#define  t0con      0x036

#define  t1cnt      0x038
#define  t1cmpa     0x03A
#define  t1cmpb     0x03C
#define  t1con      0x03E

#define  t2cnt      0x040
#define  t2cmpa     0x042
#define  t2con      0x046

   /* Watchdog del i80C186EC */

#define  wdtrtdh               0x020
#define  wdtrtdl               0x022
#define  wdtcnth               0x024
#define  wdtcntl               0x026
#define  wdtclear              0x028
#define  wdtdisable            0x02A

   /* Puertos paralelos del i80C186EC */

#define  p3dir      0x048
#define  p3pin      0x04A
#define  p3con      0x04C
#define  p3ltch     0x04E
```

```
#define  p1dir      0x050
#define  p1pin      0x052
#define  p1con      0x054
#define  p1ltch     0x056

#define  p2dir      0x058
#define  p2pin      0x05A
#define  p2con      0x05C
#define  p2ltch     0x05E

  /* Chip selects del i80C188EC */

#define  gcs0st     0x080
#define  gcs0sp     0x082
#define  gcs1st     0x084
#define  gcs1sp     0x086
#define  gcs2st     0x088
#define  gcs2sp     0x08A
#define  gcs3st     0x08C
#define  gcs3sp     0x08E
#define  gcs4st     0x090
#define  gcs4sp     0x092
#define  gcs5st     0x094
#define  gcs5sp     0x096
#define  gcs6st     0x098
#define  gcs6sp     0x09A
#define  gcs7st     0x09C
#define  gcs7sp     0x09E
#define  lcsst      0x0A0
#define  lcssp      0x0A2
#define  ucsst      0x0A4
#define  ucssp      0x0A6

  /* Relocation register, Step identifier y Power control registers */

#define  relreg     0x0A8
#define  stepid     0x0BC
#define  pwrcon     0x0B8
#define  pwrsav     0x0BE

  /* Refresh */

#define  rfbase     0x0B0
#define  rftime     0x0B2
#define  rfcon      0x0B4
#define  rfaddr     0x0B6

  /* DMA Controllers */

#define  d0srcl     0x0C0
#define  d0srch     0x0C2
#define  d0dstl     0x0C4
#define  d0dsth     0x0C6
#define  d0tc       0x0C8
#define  d0con      0x0CA

#define  dmapri     0x0CC
#define  dmahalt    0x0CE

#define  d1srcl     0x0D0
#define  d1srch     0x0D2
#define  d1dstl     0x0D4
#define  d1dsth     0x0D6
#define  d1tc       0x0D8
#define  d1con      0x0DA

#define  d2srcl     0x0E0
#define  d2srch     0x0E2
#define  d2dstl     0x0E4
```

```
#define  d2deth     0x0E6
#define  d2tc       0x0E8
#define  d2con      0x0EA

#define  d3srcl     0x0F0
#define  d3srch     0x0F2
#define  d3dstl     0x0F4
#define  d3dsth     0x0F6
#define  d3tc       0x0F8
#define  d3con      0x0FA

  /* Registros de interrupcion interna */

#define  scuirl     0x00A
#define  dmairl     0x00C
#define  timirl     0x00E

  /* PICs */

#define  masterpic_0        0x000
#define  masterpic_1        0x002
#define  slavepic_0         0x004
#define  slavepic_1         0x006


/* Defines para la inicializacion del PIC */

#define ICW1_m    0x11 /* Edge trigger, cascade mode, ICW4 needed */
#define ICW2_m    0x20 /* Interrupt 32=20H */
#define ICW3_m    0x80 /* S7 has slave */
#define ICW4_m    0x01

#define ICW1_s    0x19
#define ICW2_s    0x28
#define ICW3_s    0x07
#define ICW4_s    0x01
```

C.2 ARCHIVO LIB_JBOX.C

```
/*****************************************************************************
  Programa LIB_JBOX.C  Rutinas básicas para el Junction Box
*****************************************************************************/
#include <i8086.h>
#include <c:\stbc\header\atse\jbox.lib>

#define CR    0x0D
#define LF    0x0A
#define BS    0x08
#define CLS   0x1a
#define ESC   0x1b
#define BELL  0x07

extern char rec_pdte0 = 0;
extern char rec_pdte1 = 0;

/*****************************************************************************
  Inicializacion del Junction Box
*****************************************************************************/
void ini_jbox(void)
{
/* ROM */
outword(dir_base_PCB + ucset,0xe000);
outword(dir_base_PCB + ucssp,0xfffe);

/* RAM */
outword(dir_base_PCB + icset,0x0000);
outword(dir_base_PCB + icssp,0x200a);

/* PIC ICWs */
outword(dir_base_PCB + masterpic_0,ICW1_m);
outword(dir_base_PCB + masterpic_1,ICW2_m);
outword(dir_base_PCB + masterpic_1,ICW3_m);
outword(dir_base_PCB + masterpic_1,ICW4_m);

outword(dir_base_PCB + slavepic_0,ICW1_s);
outword(dir_base_PCB + slavepic_1,ICW2_s);
outword(dir_base_PCB + slavepic_1,ICW3_s);
outword(dir_base_PCB + slavepic_1,ICW4_s);

outword(dir_base_PCB + masterpic_1,0x00ff);
outword(dir_base_PCB + slavepic_1,0x00ff);

/* SCUs */
outword(dir_base_PCB + p2con,0x00ff);  /* Seleccion de SCU0 y SCU1 */
outword(dir_base_PCB + b0cmp,0x804e);  /* SCU0 a 19.2 kbauds */
outword(dir_base_PCB + b1cmp,0x804e);  /* SCU1 a 19.2 kbauds */
outword(dir_base_PCB + s0con,0x0021);  /* 8 bits de datos, no paridad */
outword(dir_base_PCB + s1con,0x0021);  /* y receptor habilitado  */

/* TBC */
outword(dir_base_PCB + gcs0at,0x2008);
```

```
outword(dir_base_PCB + gcs0sp,0x2048);
}

/*************************************************************************
 Libreries para el Junction Box
*************************************************************************/
/* Puerto 0 */
void saca_caracter(unsigned char c)
{
 char status0;
 status0 = inbyte(dir_base_PCB + s0sts);
 if ((status0 & rx_listo_scu) == rx_listo_scu)
   rec_pdte0 = status0;
 while ((status0 & tx_listo_scu) == 0 )
   {
   status0 = inbyte(dir_base_PCB + s0sts);
   if ((status0 & rx_listo_scu) == rx_listo_scu)
   rec_pdte0 = status0;
   }
 outbyte(dir_base_PCB + s0tbuf, c );
}

unsigned char obten_dato()
{
 while (((inbyte(dir_base_PCB + s0sts) | rec_pdte0) & rx_listo_scu) == 0 );
 rec_pdte0 = 0;
 return(inbyte(dir_base_PCB + s0rbuf));
}

/* Puerto 1 */
void saca_caracter1(unsigned char c)
{
 char status1;
 status1 = inbyte(dir_base_PCB + s1sts);
 if ((status1 & rx_listo_scu) == rx_listo_scu)
   rec_pdte1 = status1;
 while ((status1 & tx_listo_scu) == 0 )
   {
   status1 = inbyte(dir_base_PCB + s1sts);
   if ((status1 & rx_listo_scu) == rx_listo_scu)
   rec_pdte1 = status1;
   }
 outbyte(dir_base_PCB + s1tbuf, c );
}

unsigned char obten_dato1()
{
 while (((inbyte(dir_base_PCB + s1sts) | rec_pdte1) & rx_listo_scu) == 0 );
 rec_pdte1 = 0;
 return(inbyte(dir_base_PCB + s1rbuf));
}

/*************************************************************************
 Defines utilizados para el manejo de interrupciones
```

```
*****************************************************************************/
#define EOI    0x20
#define MASCARA 0xff
#define NO_MASC 0x7e

/*****************************************************************************
 Funciones para implementar el manejo de las interrupciones
*****************************************************************************/
void habilita(void)
{
outbyte(dir_base_PCB + masterpic_1,NO_MASC);
}

void deshabilita(void)
{
outbyte(dir_base_PCB + masterpic_1,MASCARA);
}

void fin_int(void)
{
outbyte(dir_base_PCB + masterpic_0,EOI);
}
```

C.3 ARCHIVO JBOX.INI

```
/*************************************************************************
  Inicializacion de las variables externas
*************************************************************************/

/* Variables para el TBC  en forma general */
int long_dir_mac=2;                     /* long  en bytes de la diraccion MAC, puede ser 2 o 6 bytes */
unsigned int num_est=0x1234;            /* numero de nuestra estacion dentro de la red  + -*/
unsigned int num_est_1=0x9ab4;          /* byte menos significativo  + -*/
unsigned int num_est_2=0x5678;
unsigned int num_est_3=0x1234;          /* byte mas significativo */
unsigned int memtarjeta=0x4000;         /* tamaño de la memoria disponible en la tarjeta (16 kbytes) */
unsigned int ldb=110;                   /* longitud en bytes de un db       */
unsigned int prpmem=2;                  /* proporcion entre mem. de tx y rx */
unsigned int prp=2;                     /* por cada fd hay 'prp' bd's       */

unsigned int bus_ancho=8;               /* ancho del bus de datos en bits   */
unsigned short puerto=0x2000;           /* acceso por puerto al tbc         */
unsigned long int memestr=0x10001500;   /* localizacion de mem para estruc. */
unsigned int ne_dest=0x5678;            /* numero de estacion destino dentro de la red  + -*/
unsigned int ne_1_dest=0x9ab8;          /* byte menos significativo     + -*/
unsigned int ne_2_dest=0xef01;
unsigned int ne_3_dest=0xabcd;          /* byte mas significativo */

unsigned long int mem_rx=0x1e000000;    /* mem a donde se va a rx   */
unsigned long int mem_rec=0x1e000802;   /* mem para indicar recepción(es) pendiente(s) */
unsigned long int mem_prior_rx=0x1e000810;  /* mem para guardar las prioridades de los mensajes recibidos */

unsigned long int mem_tx=0x1f000000;    /* mem a donde se va a tx   */
unsigned long int mem_prior_tx=0x1e000804;  /* mem para guardar la prioridad del mensaje a transmitir */
unsigned long int mem_size=0x1e000800;  /* mem para guardar la longitud del mensaje a transmitir */

/* Variables para el junction box exclusivamente */
unsigned long int mem_rx_red=0x1e000400;    /* mem para adquisición de datos */
unsigned long int mem_tx_red=0x1f000400;    /* mem para comandos a dispositivos inteligentes */
unsigned long int mem_trans=0x1e000803;     /* mem para indicar que se ha terminado de transmitir el mensaje */
unsigned long int mem_red=0x1f000b00;       /* mem para guardar el estado de la red */

/*************************************************************************
  Vaariables globales para los comandos
*************************************************************************/

/* set mode 1 */
unsigned char cufc=0;
unsigned char cacf=0;
unsigned char latt=1;
unsigned char pdrf=0;
/* set mode 2 */
unsigned char tbrm=0;
unsigned char lmd=1;
unsigned char srtb=0;
unsigned char brg =0;
unsigned char rar =0;
/* set mode 3 */
unsigned char rcds=0;
unsigned char tcds=0;
unsigned char hien=0;
unsigned char swap=0;
unsigned char ps3 =1;

/*************************************************************************
  Defines de los vectores de interupcion
*************************************************************************/
#define TBC 0x20                        /* propio del hardware */
#define INI 0x60
#define TX 0x61
#define RX 0x62
#define HIT 0x63
#define FULL 0x64
#define DEST 0x65
```

C.4 ARCHIVO JBOX.C

```
/*************************************************************
JBOX.C  Programa principal para el Junction Box
*************************************************************/
#include <i8086.h>
#include <c:\stbc\header\etasjbox.lit>
#include <c:\stbc\header\libjbox.h>
#include <c:\stbc\header\libio.h>
#include <c:\stbc\header\libcalc.h>
#include <c:\stbc\header\header.h>
#include <c:\stbc\header\lie_red.h>

#include <c:\stbc\header\structs.h>
#include "jbox.ini"

#define TMR0 0x28
#define TMR1 0x29
#define DMA2 0x2a
#define DMA3 0x2b
#define TMR2 0x2c
#define SCU_RX0 0x2d
#define SCU_TX0 0x2e

/*************************************************************************
Interrupciones internas del i80C186EC
*************************************************************************/
/*-------------------------------------------------------------
 Interrupcion por Dma3 al recibir mensajes de la red
-------------------------------------------------------------*/
#pragma interrupt("dma3_int3")
void far dma3_int3(void)
{
char far *p1;
int far *p2;

  outword(dir_base_PCB + slavepic_1,0xfe);
/*  saca_caracter(BELL);
  imprime("Dma3",1);*/
  /* se detiene Dma3 */
  outword(dir_base_PCB + d3con,0xa354);
  outword(dir_base_PCB + d3tc,0x0);

  p1=(char far *)mem_trans;
  p2=(int far *)mem_size;
  *p2=red_lie();
  causeinterrupt(TX);
  /*indica que ha terminado la transmision */
  *p1=0xff;
  outword(dir_base_PCB + dmairl,0x0800);
  outword(dir_base_PCB + masterpic_0,EOI);
  outword(dir_base_PCB + slavepic_0,EOI);
  outword(dir_base_PCB + slavepic_1,0xf6);
}

/*-------------------------------------------------------------
 Interrupcion por Tmr0 al no recibir respuesta de la red
-------------------------------------------------------------*/
#pragma interrupt("timer0_int0")
void far timer0_int0(void)
{
char far *p1;
int far *p2;

 outword(dir_base_PCB + slavepic_1,0xff);
 imprime("Tmr0",1);
 /* se detienen Tmr0, Tmr2 y Dma3 */
 outword(dir_base_PCB + t0con,0x4000);
 outword(dir_base_PCB + t2con,0x4000);
```

```
 outword(dir_base_PCB + d3con,0xa354);
 outword(dir_base_PCB + d3tc,0x0);

 p1=(char far *)mem_tx;
 p2=(int far *)mem_size;
 *p2=imprime("Error: no hubo respuesta de la red",1)+1;
 strcpy(p1,"Error: no hubo respuesta de la red");
 p1+=(*p2-1);
 *p1=CR;
 causeinterrupt(TX);
 /* se indica que ha terminado la transmision */
 p1=(char far *)mem_trans;
 *p1=0xff;
 outword(dir_base_PCB + timirl,0x0100);
 outword(dir_base_PCB + masterpic_0,EOI);
 outword(dir_base_PCB + slavepic_0,EOI);
 outword(dir_base_PCB + slavepic_1,0xf6);
}

/*************************************************************************
 PROGRAMA PRINCIPAL
*************************************************************************/
void main(void)
{
char selec;
char far *p1,*p2,*p3,*p4,*p5,*p6,*p7;
int far *p8;
unsigned long int a;
unsigned int size,time;

disable();

Ini_Ibox();
setinterrupt(TBC,lee_Int);
setinterrupt(DMA3,dma3_int3);
setinterrupt(TMR0,timer0_int0);
setinterrupt(INI,inicializacion);
setinterrupt(TX,transmision);
setinterrupt(RX,recepcion);
setinterrupt(HIT,host_test);
setinterrupt(FULL,full_duplex);
setinterrupt(DEST,cambia_dest);

/* Limpiar interrupciones pendientes */
outword(dir_base_PCB + sciirl,0x0f00);
outword(dir_base_PCB + dmairl,0x0f00);
outword(dir_base_PCB + timirl,0x0f00);

/* Programar las fuentes internas de interrupcion*/
/* TMR 2 */
   outword(dir_base_PCB + t2cnt,0x0000);
   outword(dir_base_PCB + t2cmpa,0xffff);
   outword(dir_base_PCB + t2con,0x0000);

/* TMR 0 */
   outword(dir_base_PCB + t0cnt,0x0000);
   outword(dir_base_PCB + t0cmpa,0x00ff);
   outword(dir_base_PCB + t0con,0x0000);

/* DMA3 */
   p7=(char far *)mem_rx_red;
   a=valor_ptr(p7);
   outword(dir_base_PCB + d3srcl,dir_base_PCB + s1rbuf);
   outword(dir_base_PCB + d3srch,0x0000);
   outword(dir_base_PCB + d3dstl,(int)a);
   outword(dir_base_PCB + d3dsth,(int)(a>>16));

   outword(dir_base_PCB + dmaprl,0x0500);
```

```
/* Desenmascarar interrupciones */
outword(dir_base_PCB + masterpic_1,0x7e);
outword(dir_base_PCB + slavepic_1,0xf8);

enable();

saca_caracter(CLS);
causeinterrupt(INI);
inicia_red();

p3=(char far *)mem_prior_tx;
p4=(char far *)mem_trans;
p6=(char far *)mem_rec;
p8=(int far *)mem_size;
*p8=0;
*p4=0xff;

imprime("IiE_Adam       Junction Box",1);
do
 {
 /***** automatico *****/
 while(!*p6);
/*
 imprime("Rx lista",0);
 obten_dato();
 imprime("",1);
*/
 p1=(char far *)mem_rx;
 p2=(char far *)mem_prior_rx;
 causeinterrupt(RX);
 while(*p2!=CR)
  {
  size=iie_red(p1);
  /* se ve el mensaje recibido */
  while(1)
   {
   saca_caracter(*p1);
   if (*p1==CR)
    break;
   p1++;
   }
  saca_caracter(LF);
  /* se escribe la prioridad del mensaje recibido */
  *p3=*p2;
  /* transmision del mensaje a la red */
  p5=(char far *)mem_tx_red;
  if(size!=0)
   {
   /* se prepara la recepcion por DMA */
   outword(dir_base_PCB + d3tc,size);
   /* recordar que a es el valor del apuntador a mem_rx_red */
   outword(dir_base_PCB + d3dsti,(int)a);
   outword(dir_base_PCB + d3dsth,(int)(a>>16));
   outword(dir_base_PCB + d3con,0xa356);
   *p4=0;
   }

  /* tiempo para que el puerto de la PC se recupere */
/*   for(time=0;time<=7000;time++);*/
  /* este tiempo no aparecera en el programa definitivo */

  while(1)
   {
   saca_caracter1(*p5);
   if (*p5==CR)
    break;
   p5++;
   }
 /* hay que esperar respuesta de la red, se programan los timers */
```

```
    outword(dir_base_PCB + t2cnt,0x0000);
    outword(dir_base_PCB + t0cnt,0x0000);
    outword(dir_base_PCB + t2con,0xc001);
    outword(dir_base_PCB + t0con,0xa009);

/*
    if(size==0)
      {
      *p8=imprime("Sampling...",1)+1;
      p7=(char far *)mem_bc;
      strcpy(p7,"Sampling...");
      p7+=(*p8-1);
      *p7=CR;
      causeinterrupt(TX);
      *p4=0xff;
      /* tiempo para que el puerto se recupere */
/*      for(time=0;time<=30000;time++);*/
      /* este tiempo no aparecera en el programa definitivo */
      }
*/
    while(!*p4);
    /* si no hay errores se detienen los timers */
    outword(dir_base_PCB + t2con,0x4000);
    outword(dir_base_PCB + t0con,0x4000);

    p1++;
    p2++;
    }
  imprime("",1);
}while(1);
}
```

C.5 ARCHIVOS ADAM.H E IIE_ADAM.C

```
/*************************************************************************
  ADAM.H - Configuracion de los dispositivos
*************************************************************************/
typedef struct adam ADAM;
extern struct adam
{
  char type;
  char format;
  char baud;
  char range;
};

/*************************************************************************
  IIE_ADAM.C - Programa para convertir protocolos de comunicación
*************************************************************************/
#include <i8086.h>
#include <c:\stbc\header\adam.h>
#include <c:\stbc\header\xtern.h>

/*************************************************************************
  Defines de los vectores de interupcion
*************************************************************************/
#define TBC 0x20
#define INI 0x60
#define TX 0x61
#define RX 0x62
#define HIT 0x63
#define FULL 0x64
#define DEST 0x65

#define CR 0x0d

/*************************************************************************
  Convierte los caracteres ASCII en bytes
*************************************************************************/
unsigned char ascii_byte(char alfa,char beta)
{
char byte=0;

  if (alfa>='0' && alfa <='9')
    {
    byte=(alfa-48)&0x0f;
    byte<<=4;
    }
  if (alfa>='a' && alfa <='f')
    {
    byte=(alfa-87)&0x0f;
    byte<<=4;
    }
  if (alfa>='A' && alfa <='F')
    {
    byte=(alfa-55)&0x0f;
    byte<<=4;
    }
  if (beta>='0' && beta <='9')
    byte=byte | ((beta-48)&0x0f);
  if (beta>='a' && beta <='f')
    byte=byte | ((beta-87)&0x0f);
  if (beta>='A' && beta <='F')
    byte=byte | ((beta-55)&0x0f);
  return byte;
}

/*************************************************************************
  Inicializacion de la red de dispositivos Adam
*************************************************************************/
void inicia_red()
```

```
{
ADAM far *ptr;
int i;

ptr=(ADAM far *)mem_red;
for(i=0;i<=0xff;i++)
 {
 ptr->format=0;
 ptr->baud='j';
 ptr->type=0;
 ptr->range='e';
 ptr++;
 }
}

/*****************************************************************************
  IIE_Adam
*****************************************************************************/
unsigned int iie_red(char far *rec)
{
ADAM far *ptr;
char far *tra,*p1;
int far *p2;
unsigned int size=0;

/* Comandos exclusivos para el Token Bus Controller
  (estos comandos son una propuesta pues no existe el protocolo IIE) */
if(*rec=='&')
 {
 p1=(char far *)mem_bc;
 p2=(int far *)mem_size;
 switch(*(rec+1))
  {
  case '1':
    causeinterrupt(INI);
    *p2=imprime("Reinicializacion OK!",1)+1;
    strcpy(p1,"Reinicializacion OK!");
    p1+=(*p2-1);
    *p1=CR;
    break;
  case '2':
    causeinterrupt(HIT);
    causeinterrupt(INI);
    *p2=imprime("HIT!",1)+1;
    strcpy(p1,"HIT!");
    p1+=(*p2-1);
    *p1=CR;
    break;
  case '3':
    causeinterrupt(FULL);
    causeinterrupt(INI);
    *p2=imprime("FULL!",1)+1;
    strcpy(p1,"FULL!");
    p1+=(*p2-1);
    *p1=CR;
    break;
  case '4':
    causeinterrupt(DEST);
    *p2=imprime("DEST!",1)+1;
    strcpy(p1,"DEST!");
    p1+=(*p2-1);
    *p1=CR;
    break;
  case '5':
    size=0;
    rec+=2;
    while(1)
     {
     *p1=*rec;
```

```
      size++;
      if(*rec==CR)
        break;
      p1++;
      rec++;
      }
     *p2=size;
     break;
     }
   /* Indicacion de no transmision a la red (ver jbox.c) */
   p1=(char far *)mem_trans;
   *p1=0xaa;
   /* se regresa cero pues no se desea programar la recepcion de la red */
   return 0;
 }
 /* si el comando no es para el TBC, se analiza para ADAM */

/*-----------------------------------------------------------------
 Interpretacion de comandos IIE para escribir comandos ADAM
-----------------------------------------------------------------*/
ptr=(ADAM far *)mem_red;
tra=(char far *)mem_bx_red;
while(1)
/* debido a que no existe un protocolo IIE, el mensaje se transcribe */
 {
 *(tra+size)=*rec;
 if(*(tra+size)==CR)
  break;
 size++;
 rec++;
 }

/*-----------------------------------------------------------------
 Interpretacion de comandos ADAM para obtener tamaño de la respuesta
-----------------------------------------------------------------*/
 ptr=ptr+(ascii_byte(*(tra+1),*(tra+2)));
 switch(*tra)
  {
  case '%':
   /* si el comando tiene la longitud correcta se actualiza la red */
   if(*(tra+11)==CR)
    {
    ptr=(ADAM far *)mem_red;
    ptr=ptr+(ascii_byte(*(tra+3),*(tra+4)));
    ptr->type=ascii_byte(*(tra+7),*(tra+8));
    ptr->baud=ascii_byte('0','7');
    switch(ascii_byte(*(tra+9),*(tra+10)))
     {
     case 0:
     case 1:
      ptr->format=7;
      break;
     case 2:
      ptr->format=4;
     }
    }
   size=4;
   break;

  case '$':
   if(*(tra+1)=='^' && *(tra+2)=='^' && *(tra+3)==CR)
    {
    size=0;
    break;
    }
   if((*(tra+3)==CR) && (ptr->type==0))
    {
    size=ptr->format+2;
    break;
```

```
    }
   if((ptr->type==1)||(ptr->type==2))
    {
    if((ptr->format==7)&&(*(tra+10)==CR))
     {
     size=2;
     break;
     }
    if((ptr->format==4)&&(*(tra+7)==CR))
     {
     size=2;
     break;
     }
    }
   size=4;
   break;

  case '$':
   if(*(tra+4)==CR)
    {
    switch (*(tra+3))
     {
     case '2':
      size=10;
      break;
     case '3':
      if(ptr->type == 0)
       {
       size=9;
       break;
       }
      size=4;
      break;
     case '4':
      if(ptr->type==0)
       {
       size=ptr->format+5;
       break;
       }
      if(ptr->type==2)
       {
       size=9;
       break;
       }
      size=4;
      break;
     case '5':
      if(ptr->type!=0)
       {
       size=5;
       break;
       }
      size=4;
      break;
     case '6':
      if(ptr->type == 1)
       {
       size=ptr->format+4;
       break;
       }
      if(ptr->type == 2)
       {
       size=8;
       break;
       }
      size=4;
      break;
     case '8':
      if(ptr->type == 1)
```

```
        {
         size=ptr->format+4;
         break;
         }
        size=4;
        break;
       case '0':
       case '1':
       case '9':
        size=4;
       }
      break;
      }
     size=4;
     break;

    case '@':
     if(ptr->type == 0)
      {
      if(*(tra+5)==CR && *(tra+3)=='R')
       {
       switch(*(tra+4))
        {
        case 'H':
        case 'L':
         size=11;
         break;
        case 'E':
         size=9;
        }
       break;
       }
      if(*(tra+5)==CR && *(tra+3)=='D' && *(tra+4)=='I')
       {
       size=9;
       break;
       }
      }
     size=4;
     break;
   }
  return size;
}

/*****************************************************************
 Adam_IIE
*****************************************************************/
unsigned int red_iie()
{
/* debido a que no existe protocolo IIE se transcribe la respuesta */
char far *p1,*p2;
unsigned int size=0;

p1=(char far *)mem_rx_red;
p2=(char far *)mem_bc;
while(1)
 {
 *p2=*p1;
 size++;
 if(*p2==CR)
   break;
 p2++;
 p1++;
 }
return size;
}
```

C.6 ARCHIVO FLASH.C

```
/*****************************************************************************
 FLASH.C  Programa para borrar y programar la memoria FLASH
*****************************************************************************/
#include <i8086.h>
#include <c:\stbc\header\libjbox.h>
#include <c:\stbc\header\libio.h>
#include <c:\stbc\header\libcalc.h>
#include <c:\stbc\header\stasjbox.lit>

/*****************************************************************************
 Variables globales
*****************************************************************************/
extern unsigned long int dir_flash = 0x20000000;/*apuntador*/
extern unsigned long int dir0_ram = 0x00000;/*direccion en hexa*/
extern unsigned long int dir0_flash = 0x20000;/*direccion en hexa*/
extern unsigned long int dir0_rom = 0x00000;/*direccion en hexa*/
extern unsigned long int tam_flash =0x1ffff; /* tamaños de las memorias */
extern unsigned long int tam_ram =0x1ffff;
extern unsigned long int tam_rom =0xffff;

/*****************************************************************************
 Rutina para leer codigo y protecciones
*****************************************************************************/
void sector_protection()
{
char far *ptr;
char a;

saca_caracter(CLS);
for(a=0;a<2;a++)
  imprime("",1);
imprime(" ---------------------------------------------",1);
imprime("            Sectores protegidos",1);
imprime("",1);

ptr=(char far *)dir_flash;
/***** Comando Autoselect *****/

*(ptr+0x5555)=0xaa;
*(ptr+0x2aaa)=0x55;
*(ptr+0x5555)=0x90;

/***** Se leen el Manufacture Code y el Am29F010 Device Code *****/

if(*(ptr+0x00)==0x01)
  imprime("    Fabricante: Advanced Micro Devices",1);
else
  imprime("    Error en la lectura de [Manufacture Code]",1);

if(*(ptr+0x01)==0x20)
  imprime("    Memoria: Am29F010",1);
else
  imprime("    Error en la lectura de [Device Code]",1);

/***** Se busca cual sector esta protegido *****/

imprime("",1);
  imprime("    Sector 0 (00000h-03FFFh): ",0);
if(*(int_ptr(dir_flash+0x00002))==0x01)
  imprime("protegido",1);
 else
  imprime("no protegido",1);

  imprime("    Sector 1 (04000h-07FFFh): ",0);
if(*(int_ptr(dir_flash+0x04002))==0x01)
  imprime("protegido",1);
 else
```

```
  imprime("no protegido",1);

imprime("",1);
  imprime("   Sector 2 (08000h-0BFFFh): ",0);
if(*(int_ptr(dir_flash+0x08002))==0x01)
  imprime("protegido",1);
 else
  imprime("no protegido",1);

  imprime("   Sector 3 (0C000h-0FFFFh): ",0);
if(*(int_ptr(dir_flash+0x0c002))==0x01)
  imprime("protegido",1);
 else
  imprime("no protegido",1);

imprime("",1);
  imprime("   Sector 4 (10000h-13FFFh): ",0);
if(*(int_ptr(dir_flash+0x10002))==0x01)
  imprime("protegido",1);
 else
  imprime("no protegido",1);

  imprime("   Sector 5 (14000h-17FFFh): ",0);
if(*(int_ptr(dir_flash+0x14002))==0x01)
  imprime("protegido",1);
 else
  imprime("no protegido",1);

imprime("",1);
  imprime("   Sector 6 (18000h-1BFFFh): ",0);
if(*(int_ptr(dir_flash+0x18002))==0x01)
  imprime("protegido",1);
 else
  imprime("no protegido",1);

  imprime("   Sector 7 (1C000h-1FFFFh): ",0);
if(*(int_ptr(dir_flash+0x1c002))==0x01)
  imprime("protegido",1);
 else
  imprime("no protegido",1);

imprime(" ------------------------------------------------------------",1);
for(a=0;a<2;a++)
  imprime("",1);
imprime(" ",0);
pause();

/***** Comando Read/Reset *****/
*(ptr+0x5555)=0xaa;
*(ptr+0x2aaa)=0x55;
*(ptr+0x5555)=0xf0;
}

/*************************************************************************
 Rutina para programar la memoria
*************************************************************************/
void program()
{
char far *ptr;
char far *ptr1;
char far *ptr2;
unsigned int num_bytes,cont;
unsigned long int dir_inI_flash=0;
unsigned long int dir_inI_copy=0;
char a;
char error;

do
{
```

```
saca_caracter(CLS);
error=' ';
for(a=0;a<3;a++)
  imprime("",1);
imprime("  ---------------------------------------------",1);
imprime("       Programacion de la memoria Flash",1);
imprime("",1);
imprime("    Numero de bytes a copiar (max 64k): ",0);
num_bytes=obten_num4();
imprime("",1);

   imprime("    Direccion fuente (memoria RAM/ROM/Flash) (Seg:Off] ",0);
   dir_ini_copy=(obten_num4()&dir_ini_copy)<<16;
   imprime(":",0);
   dir_ini_copy=obten_num4()|dir_ini_copy;

   ptr1=(char far *)dir_ini_copy;
   if (ptr_int(ptr1)<dir0_ram || ptr_int(ptr1)>(dir0_ram+tam_ram))
    {
    if (ptr_int(ptr1)<dir0_flash || ptr_int(ptr1)>(dir0_flash+tam_flash))
     {
     if (ptr_int(ptr1)<dir0_rom || ptr_int(ptr1)>(dir0_rom+tam_rom))
      {
      imprime("",1);
      imprime("    Direccion invalida",1);
      error='e';
      }
     }
    }

imprime("",1);

   imprime("    Direccion destino (memoria Flash) (Seg:Off] ",0);
   dir_ini_flash=(obten_num4()&dir_ini_flash)<<16;
   imprime(":",0);
   dir_ini_flash=obten_num4()|dir_ini_flash;

   ptr2=(char far *)dir_ini_flash;
   if (ptr_int(ptr2)<dir0_flash || ptr_int(ptr2)>(dir0_flash+tam_flash))
    {
    imprime("",1);
    imprime("    Direccion fuera de Flash",1);
    error='e';
    }

imprime("",1);
imprime("",1);

/***** Chequeo de dimensiones *****/

imprime("    Chequeo de dimensiones...",1);
/*if (ptr_int(ptr1)+num_bytes>dir0_ram+tam_ram)
  {
  error='e';
  imprime("    El numero de bytes dado rebasa el tama&o de la RAM",1);
  }*/

if (ptr_int(ptr2)+num_bytes>dir0_flash+tam_flash)
  {
  error='e';
  imprime("    El numero de bytes dado rebasa el tama&o de la Flash",1);
  imprime("    ",0);
  pausa();
  imprime("",1);
  }

imprime("    Deseas continuar con la programacion? (s/n): ",0);
a=obten_dato();
saca_caracter(a);
```

```
imprime("",1);
if (a=='n' || a=='N')
  return;
}while(error=='e');

/***** Programacion ****/

ptr=(char far *)dir_flash;

imprime("",1);
imprime("    Direccion inicial (RAM/ROM/Flash):  ",0);
saca_ptr(ptr1);
imprime("",1);
imprime("    Direccion inicial en Flash: ",0);
saca_ptr(ptr2);
imprime("",1);
imprime("    Numero de bytes: ",0);
saca_int(num_bytes);
imprime("",1);
imprime("",1);
imprime("    kbyte: ",0);
cont=0;
a=0;
while(cont<=num_bytes)
{
 /***** Comando Byte Program *****/
 *(ptr+0x5555)=0xaa;
 *(ptr+0x2aaa)=0x55;
 *(ptr+0x5555)=0xa0;
 *(ptr2+cont)=*(ptr1+cont);

 while(*(ptr2+cont)!=*(ptr1+cont));

 if (cont%0x400==0)
   {
   saca_byte(a);
   saca_caracter('h');
   saca_caracter(BS);
   saca_caracter(BS);
   saca_caracter(BS);
   a++;
   }

if (cont==0xffff) break;
cont++;
}
imprime("",1);
imprime("",1);
imprime("            Programacion finalizada con exito!",1);
imprime("  ------------------------------------------------",1);
imprime("",1);
imprime("    ",0),
pause();
}

/*************************************************************************
 Rutina para borrar la memoria
*************************************************************************/
void chip_erase()
{
char far *ptr;
char error,a;

saca_caracter(CLS);
for(a=0;a<5;a++)
  imprime("",1);
imprime("  ------------------------------------------------",1);
imprime("            Borrado de la memoria Flash",1);
imprime("",1);
```

```
imprime("    AGUAS!! Esta opcion borra la memoria completamente",1);
imprime("    Deseas continuar? (s/n): ",0 );
error=c_bten_dato();
seca_caracter(error);
if (error!='s' && error!='S')
  return;

imprime("",1);
imprime("",1);
imprime("",1);
imprime("    Borrando la memoria Flash...",0);

ptr=(char far *)dir_flash;
*(ptr+0x5555)=0xaa;
*(ptr+0x2aaa)=0x55;
*(ptr+0x5555)=0x80;
*(ptr+0x5555)=0xaa;
*(ptr+0x2aaa)=0x55;
*(ptr+0x5555)=0x10;

while((*(ptr)&0x80)==0)
  {
  seca_caracter('.');
  seca_caracter(BS);
  }
imprime("",1);
imprime("    ...memoria Flash borrada completamente",1);
imprime("",1);
imprime("  ----------------------------------------",1);
imprime("",1);

imprime("    ",0);
pausa();
}

/*****************************************************************************
 Rutina para borrar la memoria por sectores
*****************************************************************************/
void sector_erase()
{
char i,a,error;
char array[8]={' ',' ',' ',' ',' ',' ',' ',' '};
char far *ptr;
char far *ptr0,*ptr1,*ptr2,*ptr3,*ptr4,*ptr5,*ptr6,*ptr7;
unsigned long int data_poll;

do
{
error=' ';
seca_caracter(CLS);
for(a=0;a<3;a++)
  imprime("",1);
imprime("  ----------------------------------------",1);
imprime("          Borrado por sectores de la memoria Flash",1);
imprime("",1);
  imprime("    Sector 0 (00000h-03FFFh): ",0);
  imprime("    Sector 4 (10000h-13FFFh): ",0);
imprime("",1);
  imprime("    Sector 1 (04000h-07FFFh): ",0);
  imprime("    Sector 5 (14000h-17FFFh): ",0);
imprime("",1);
  imprime("    Sector 2 (08000h-0BFFFh): ",0);
  imprime("    Sector 6 (18000h-1BFFFh): ",0);
imprime("",1);
  imprime("    Sector 3 (0C000h-0FFFFh): ",0);
  imprime("    Sector 7 (1C000h-1FFFFh): ",0);
imprime("",1);
imprime("",1);
imprime("    Que numero de sector(es) quieres borrar?: ",0);
```

```
a=0;
while(a<=7)
{
  do
  {
  error=' ';
  array[a]=obten_dato();
  if (array[a]==CR) break;
  if (array[a]<'0' || array[a]>'7')
    error='e';
  }while(error=='e');
  if (array[a]==CR)
    {
    saca_caracter(BS);
    saca_caracter(' ');
    break;
    }
  saca_caracter(array[a]);
  imprime("",0);
  a++;
 }
 saca_caracter(BS);
 saca_caracter(' ');
imprime("",1);
imprime("    Los datos son correctos? (s/n): ",0);
error=obten_dato();
saca_caracter(error);
if (error=='n' || error=='N')
  error='e';
}while(error=='e');

imprime("",1);
imprime("    Desees continuar? (s/n): ",0 );
error=obten_dato();
saca_caracter(error);
 if (error=='n' || error=='N') return;

/***** Comandos de borrado *****/
imprime("",1);
imprime("",1);
imprime("    Borrando la memoria Flash por sectores...",0);
ptr=(char far *)dir_flash;
ptr0=(char far *)(dir_flash+0x0000);
ptr1=(char far *)(dir_flash+0x4000);
ptr2=(char far *)(dir_flash+0x8000);
ptr3=(char far *)(dir_flash+0xc000);
ptr4=(char far *)(dir_flash+0x10000000);
ptr5=(char far *)(dir_flash+0x10004000);
ptr6=(char far *)(dir_flash+0x10008000);
ptr7=(char far *)(dir_flash+0x1000c000);

*(ptr+0x5555)=0xaa;
*(ptr+0x2aaa)=0x55;
*(ptr+0x5555)=0x80;
*(ptr+0x5555)=0xaa;
*(ptr+0x2aaa)=0x55;

a-=1;
i=0;
while(i<=a)
{
 switch(array[i])
  {
  case '0':
  data_poll=dir_flash+0x0000;
  *(ptr0)=0x30;
  break;
  case '1':
```

```
  data_poll=dir_flash+0x4000;
  *(ptr1)=0x30;
  break;
  case '2':
  data_poll=dir_flash+0x8000;
  *(ptr2)=0x30;
  break;
  case '3':
  data_poll=dir_flash+0xc000;
  *(ptr3)=0x30;
  break;
  case '4':
  data_poll=dir_flash+0x10000;
  *(ptr4)=0x30;
  break;
  case '5':
  data_poll=dir_flash+0x14000;
  *(ptr5)=0x30;
  break;
  case '6':
  data_poll=dir_flash+0x18000;
  *(ptr6)=0x30;
  break;
  case '7':
  data_poll=dir_flash+0x1c000;
  *(ptr7)=0x30;
  break;
  }
i++;
}

while(*(int_ptr(data_poll))&0x80==0)
 {
 saca_caracter('.');
 saca_caracter(BS);
 }
imprime("",1);
imprime("    ...memoria Flash borrada por sectores",1);
imprime(" ------------------------------------------------------------",1);
imprime("",1);
pause();
}

/*****************************************************************
 Rutina para mostrar memoria
*****************************************************************/
void memo()
{
char far *ptr;
unsigned long int direccion=0;
unsigned char error,e;

do
 {
 saca_caracter(CLS);
 for(e=0;e<3;e++)
   imprime("",1);
 imprime(" ------------------------------------------------------------",1);
 imprime("                    Memoria",1);
 imprime("",1);
   imprime("   Esta opcion muestra memoria por segmento (max 64k) en bloques",1);
 imprime("",1);
   imprime("   Direccion inicial [Seg:Off] ",0);
 direccion=(obten_num4()|direccion)<<16;
 imprime(":",0);
 direccion=direccion|obten_num4();
 ptr=(char far *)direccion;
 saca_caracter(CLS);
 saca_caracter(CR);
```

```
 mostrar_mem(ptr,127);
 imprime("",1);
 imprime("    Desees continuar viendo memoria? (s/n) ",0);
 error=obten_dato();
 seca_caracter(error);
 } while(error=='s' || error=='S');
return;
}

/*************************************************************************
 Main
*************************************************************************/
void main(void)
{
char selec,a;

ini_jbox();
/* Programacion de GCS2 */
outword(dir_base_PCB + gcs2st,0x2000);
outword(dir_base_PCB + gcs2sp,0x400a);

do
{
 do
  {
  seca_caracter(CLS);
  for(a=0;a<5;a++)
    imprime("",1);
  imprime("  ------------------------------------------------",1);
  imprime("      Utilerias para memoria Flash:",1);
  imprime("",1);
  imprime("              1. Mostrar sectores protegidos",1);
  imprime("              2. Programar",1);
  imprime("              3. Borrar sectores",1);
  imprime("              4. Borrar completamente",1);
  imprime("              5. Mostrar memoria",1);
  imprime("              6. Salir",1);
  imprime("",1);
  imprime("  ------------------------------------------------",1);

    for(a=0;a<6;a++)
          imprime("",1);
  imprime("      Eleccion:_",0);
    selec=obten_dato();
    seca_caracter(selec);
  } while((selec<'1') || (selec>'6'));

  switch(selec)
  {
   case '1':
          sector_protection();
          break;
   case '2':
          program();
          break;
   case '3':
          sector_erase();
          break;
   case '4':
          chip_erase();
          break;
   case '5':
     memo();
     break;
  }
 } while(selec!='6');
}
```

C.7 ARCHIVO ADAMPC.C

```
//*****************************************************************
// ADAMPC.C   Programa para simular módulos ADAM en una PC
//*****************************************************************
#include <dos.h>
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>

#define CR     0x0D
#define LF     0x0A
#define BS     0x08
#define CLS    0x1a
#define ESC    0x1b
#define BELL   0x07

#define COM1  0x3f8
#define STS   0x3fd

#define rx_listo 0x01
#define tx_listo 0x40

extern char rec_pte=0;

unsigned char ascii[16]={'0','1','2','3','4','5','6','7',
                         '8','9','A','B','C','D','E','F'};

unsigned char adam[16];
unsigned char cam[16];
unsigned char type=0;
int mem=0x400;

//*****************************************************************
// Puerto serie
//*****************************************************************
void saca_caracter(unsigned char c)
{
 char status;
 status=inp(STS);
 if ((status & rx_listo) == rx_listo)
   rec_pte=status;
 while ((status & tx_listo) != tx_listo)
   {
   status=inp(STS);
   if ((status & rx_listo) == rx_listo)
   rec_pte=status;
   }
 outp(COM1,c);
}

unsigned char obtan_dato()
{
 while (((inp(STS)|rec_pte) & rx_listo) == 0);
 rec_pte=0;
 return(inp(COM1));
}

int far imprime(char *p,int a)
{
 int i=0,
 while(*p)
 {
  saca_caracter(*p);
  p++;
  i++;
 }
```

```
 if (a==1)
   saca_caracter(CR);
 return i;
}

//*****************************************************************
// Inicializacion de UART
//*****************************************************************
void ini_UART()
{
// Inicializacion del 8250 UART (COM1)
outp(0x3fb,0x80); // Seleccion del registro BRG
outpw(0x3f8,6);   // Divisor 6 para obtener 19.2 bps
outp(0x3fb,3);    // Normal UART operation, Parity N, 8 bits, 1 stop
outp(0x3f9,0);    // Interrupciones deshabilitadas
inp(ST6);  // Clear LSR
inp(COM1); // Clear RxREG (COM1)
}

//*****************************************************************
// Manda y recibe comandos
//*****************************************************************
unsigned char recibe()
{
unsigned char i=0;
while(1)
 {
 edam[i]=obten_dato();
 if (edam[i]==CR)
   break;
 i++;
 }
 return i;
}

void transmite()
{
unsigned char i=0;
while(1)
 {
 saca_caracter(cam[i]);
 if (cam[i]==CR)
   break;
 i++;
 }

i=0;
do
 {
 putch(cam[i]);
 if (cam[i]==CR)
   break;
 i++;
 }while(1);
putch(LF);
}

/*-----------------------------------------------------------
 Rutina para leer 2 caracteres ASCII y convertirlos en un byte
-----------------------------------------------------------*/
unsigned char ascii_byte(char alfa,char beta)
{
char byte=0;

 if (alfa>='0' && alfa <='9')
   {
   byte=(alfa-48)&0x0f;
```

```
 if (a==1)
   saca_caracter(CR);
 return i;
}


//*****************************************************************
// Inicializacion de UART
//*****************************************************************
void ini_UART()
{
// Inicializacion del 8250 UART (COM1)
outp(0x3fb,0x80); // Seleccion del registro BRG
outpw(0x3f8,6);   // Divisor 6 para obtener 19.2 bps
outp(0x3fb,3);    // Normal UART operation, Parity N, 8 bits, 1 stop
outp(0x3f9,0);    // Interrupciones deshabilitadas
inp(ST5);  // Clear LSR
inp(COM1); // Clear RxREG (COM1)
}


//*****************************************************************
// Manda y recibe comandos
//*****************************************************************
unsigned char recibe()
{
unsigned char i=0;
while(1)
 {
 adam[i]=obten_dato();
 if (adam[i]==CR)
   break;
 i++;
 }
 return i;
}

void transmite()
{
unsigned char i=0;
while(1)
 {
 saca_caracter(cam[i]);
 if (cam[i]==CR)
   break;
 i++;
 }

i=0;
do
 {
 putch(cam[i]);
 if (cam[i]==CR)
   break;
 i++;
 }while(1);
putch(LF);
}

/*-----------------------------------------------------------
 Rutina para leer 2 caracteres ASCII y convertirlos en un byte
-----------------------------------------------------------*/
unsigned char ascii_byte(char alfa,char beta)
{
char byte=0;

 if (alfa>='0' && alfa <='9')
   {
   byte=(alfa-48)&0x0f;
```

```
  byte<<=4;
  }
 if (alfa>='a' && alfa <='f')
  {
  byte=(alfa-87)&0x0f;
  byte<<=4;
  }
 if (alfa>='A' && alfa <='F')
  {
  byte=(alfa-55)&0x0f;
  byte<<=4;
  }
 if (beta>='0' && beta <='9')
  byte=byte | ((beta-48)&0x0f);
 if (beta>='a' && beta <='f')
  byte=byte | ((beta-87)&0x0f);
 if (beta>='A' && beta <='F')
  byte=byte | ((beta-55)&0x0f);
 return byte;
}

//*****************************************************************
// Genera las respuestas de dispositivos tipo ADAM
//*****************************************************************
char data(unsigned char i,unsigned char type)
{
char a,b;
switch(type)
 {
// Engineering Units
 case 0:
  if (random(2)==0) cam[i]='-';
   else cam[i]='+';
  a=(char)random(6);
  a++;
  for (b=1;b<=6;b++)
   {
   if (a==b)
    {
     cam[i+b]='.';
     continue;
    }
   cam[i+b]=ascii[(char)random(10)];
   }
  a=7+i;
  break;

// FSR
 case 1:
  if (random(2)==0) cam[i]='-';
   else cam[i]='+';
  cam[i+1]=ascii[(char)random(10)];
  cam[i+2]=ascii[(char)random(10)];
  cam[i+3]=ascii[(char)random(10)];
  cam[i+4]='.';
  cam[i+5]=ascii[(char)random(10)];
  cam[i+6]=ascii[(char)random(10)];
  a=7+i;
  break;

// TWO'S COMPLEMENT
 case 2:
  for (b=0,b<=3,b++)
   cam[i+b]=ascii[(char)random(16)];
  a=4+i;
  break;

// ADDRESS
 case 3:
```

```
    cam[i]='I';
    cam[i+1]=adam[1];
    cam[i+2]=adam[2];
    a=3+i;
    break;
   }
  return a;
}

//*********************************************************************
// Revise que la longitud de un mensaje sea adecuada, mensajes error
//*********************************************************************
char rev_long(unsigned char lm,unsigned char a)
{
if (lm!=a+1)
  {
  printf(" Error: longitud del mensaje incorecta\n");
  imprime("???",1);
  cam[0]='?';
  cam[1]='?';
  cam[2]='?';
  cam[3]=CR;
  return 0;
  }
return 1;
}

void errtype()
{
imprime("???",1);
printf("Error: invalid data type \n");
cam[0]='?';
cam[1]='?';
cam[2]='?';
cam[3]=CR;
}

void errcom()
{
printf("Error: invalid command for this module\n");
imprime("???",1);
cam[0]='?';
cam[1]='?';
cam[2]='?';
cam[3]=CR;
}

//*********************************************************************
// Estructura ADAM
//*********************************************************************
extern struct adam_disp
  {
  char type; // Analog input: 0; Analog output: 1; Digital I/O: 2
  char format; // Engineering: 0, %FSR: 1; 2's: 2; Ohm: 3
  char range;
  };

//*********************************************************************
// main()
//*********************************************************************
main()
{
unsigned char a,b,lm;
struct adam_disp struc[256];
unsigned int i;

ini_UART();
printf("Hola! Simulación de dispositivos ADAM\n");
```

```
// Inicia ADAM
for(i=0;i<=0xff;i++)
 {
 struc[i].format=0x00;
 struc[i].type=0x00;
 struc[i].range=0x05;
 }

do
 {
 for(a=0;a<=15;a++)
  adam[a]=0;
 lm=recibe();

 printf("\n");
 printf(" Comando: ");
 for(a=0;a<=lm;a++)
  putch(adam[a]);
 putch(CR);
 putch(LF);

 // Codigo valido
 if (adam[0]!='%' && adam[0]!='@' && adam[0]!='#' && adam[0]!='$')
  {
  printf("Error: not a command \n");
  imprime("???",1);
  cam[0]='?';
  cam[1]='?';
  cam[2]='?';
  cam[3]=CR;
  continue;
  }

 a=ascii_byte(adam[1],adam[2]);
 printf("Type:   ");
 switch(struc[a].type)
  {
  case 0:
   printf("Analog Input  ");
   break;
  case 1:
   printf("Analog Output ");
   break;
  case 2:
   printf("Digital I/O   ");
   break;
  }
 printf("   Data: ");
 switch(struc[a].format)
  {
  case 0:
   printf("Engineering\n");
   break;
  case 1:
   printf("%FSR\n");
   break;
  case 2:
   printf("2's Complement\n");
   break;
  }

 switch(adam[0])
  {
 // %
  case '%':
   if(!rev_long(lm,10)) continue;
   a=ascii_byte(adam[3],adam[4]);
   struc[a].type=ascii_byte(adam[7],adam[8]);
   struc[a].range=ascii_byte(adam[5],adam[6]);
```

```
    struc[a].format=ascii_byte(adam[9],adam[10]);

    printf(" Command name: Configuration \n");
    printf("  Last Address: %c%c   New Address: %c%c\n",
          adam[1],adam[2],adam[3],adam[4]);
    printf("  Input Range:  %c%c   Type: ",adam[5],adam[6]);
    if(struc[a].type == 0)
     printf("Analog Input \n");
    if(struc[a].type == 1)
     printf("Analog Output \n");
    if(struc[a].type == 2)
     {
     printf("Digital Input/Output \n");
     struc[a].format=2;
     }
    printf("  Parameters:   %c%c   Baud: 07 \n",adam[9],adam[10]);
    cam[0]='!';
    cam[1]=adam[3];
    cam[2]=adam[4];
    cam[3]=CR;
    printf("Wait 7 seconds ...\n");
    break;

  // #
   case '#':
    if(adam[1]=='*' && adam[2]=='*' && adam[3]==CR)
     {
     if(!rev_long(im,2)) break;
     printf("All modules sampling ...\n");
     printf(" Command name: Synchronized Sampling \n");
     continue;
     }
    printf(" Address:  %c%c\n",adam[1],adam[2]);
    if(adam[3]==CR)
     {
     if(struc[a].type != 0)
      {
      errcom();
      break;
      }
     printf(" Command name: Analog Data In \n");
     cam[0]='>';
     cam[data(1,struc[a].format)]=CR;
     break;
     }
    if(adam[3]!=CR)
     {
     if(struc[a].type==0)
      {
      errcom();
      break;
      }
     if((struc[a].format==0 || struc[a].format==1)&&(adam[10]!=CR))
      {
      errtype();
      break;
      }
     if((struc[a].format==2)&&(adam[7]!=CR))
      {
      errtype();
      break;
      }
     if(struc[a].type == 1)
      printf(" Command name: Analog Data Out \n");
     if(struc[a].type == 2)
      printf(" Command name: Digital Data Out \n");
     cam[0]='>';
     cam[1]=CR;
     break;
```

```
    }
   break;

// $
  case '$':
   printf(" Address:  %c%c\n",adam[1],adam[2]);
   if(adam[3]=='3' && adam[6]==CR && struc[a].type==1)
    {
    printf(" Command name: Trim Calibration \n");
    printf(" Number of counts: %c%c\n",adam[4],adam[5]);
    cam[data(0,3)]=CR;
    break;
    }
   if(!rev_long(lm,3)) break;
   switch (adam[3])
    {
    case '0':
     if(struc[a].type == 2)
      {
      errcom();
      break;
      }
     if(struc[a].type == 0)
      printf(" Command name: Span Calibration \n");
     if(struc[a].type == 1)
      printf(" Command name: 4 mA Calibration \n");
     cam[data(0,3)]=CR;
     break;
    case '1':
     if(struc[a].type == 2)
      {
      errcom();
      break;
      }
     if(struc[a].type == 0)
      printf(" Command name: Offset Calibration \n");
     if(struc[a].type == 1)
      printf(" Command name: 20 mA Calibration \n");
     cam[data(0,3)]=CR;
     break;
    case '2':
     printf(" Command name: Configuration Status Request \n");
     b=data(0,3);
     cam[b]=ascii[(struc[a].range>>4)&0x0f];
     cam[b+1]=ascii[(struc[a].range)&0x0f];
     cam[b+2]='0';
     cam[b+3]='7';
     cam[b+4]=ascii[(struc[a].format>>4)&0x0f];
     cam[b+5]=ascii[(struc[a].format)&0x0f];
     cam[b+6]=CR;
     break;
    case '3':
     if(struc[a].type != 0)
      {
      errcom();
      break;
      }
     printf(" Command name: CJC Status \n");
     cam[0]='>';
     cam[data(1,0)]=CR;
     break;
    case '4':
     if(struc[a].type==0)
      {
      printf("Command name: Read Synchronized Data\n");
      b=data(0,3);
      cam[b]=ascii[(char)random(2)];
      b++;
      cam[data(b,struc[a].format)]=CR;
```

```
        break;
        }
      if(struc[a].type==1)
        {
        printf("Command name: Start-Up Output Current/Voltage Configuration\n");
        cam[data(0,3)]=CR;
        break;
        }
      if(struc[a].type==2)
        {
        printf(" Command name: Read Synchronized Data \n");
        cam[0]='>';
        cam[1]=ascii[(char)random(2)];
        b=data(2,2);
        cam[b]='0';
        b++;
        cam[b]='0';
        b++;
        cam[b]=CR;
        break;
        }
    case '5':
      if(struc[a].type == 0)
        {
        errcom();
        break;
        }
      printf(" Command name: Reset Status \n");
      b=data(0,3);
      cam[b]=ascii[(char)random(2)];
      b++;
      cam[b]=CR;
      break;
    case '6':
      if(struc[a].type == 0)
        {
        errcom();
        break;
        }
      if(struc[a].type == 1)
        {
        printf(" Command name: Last Value Readback\n");
        b=data(0,3);
        cam[data(b,struc[a].format)]=CR;
        break;
        }
      if(struc[a].type == 2)
        {
        printf(" Command name: Digital Data In \n");
        cam[0]='I';
        b=data(1,2);
        cam[b]='0';
        b++;
        cam[b]='0';
        b++;
        cam[b]=CR;
        break;
        }
    case '8':
      if(struc[a].type != 1)
        {
        errcom();
        break;
        }
      printf(" Command name: Current Readback \n");
      b=data(0,3);
      cam[data(b,struc[a].format)]=CR;
      break;
    case '9':
```

```
      if(struc[a].type != 0)
       {
       errcom();
       break;
       }
      printf(" Command name: CJC Offset Calibration \n");
      cam[data(0,3)]=CR;
      break;
     }
    break;

// @
   case '@':
    printf(" Address:  %c%c\n",adam[1],adam[2]);
    if(struc[a].type != 0)
     {
     errcom();
     break;
     }
    if(adam[6]==CR && adam[3]=='E' && adam[4]=='A')
     {
     printf(" Command name: Enable Alarm \n");
     if(adam[5]=='M')
      printf(" Momentary Alarm State \n");
     if(adam[5]=='L')
      printf(" Latching Alarm State \n");
     cam[data(0,3)]=CR;
     break;
     }
    if(adam[12]==CR)
     {
     if(adam[3]=='H' && adam[4]=='I')
      {
      printf(" Command name: Set High Alarm \n");
      cam[data(0,3)]=CR;
      break;
      }
     if(adam[3]=='L' && adam[4]=='O')
      {
      printf(" Command name: Set Low Alarm \n");
      cam[data(0,3)]=CR;
      break;
      }
     errcom();
     break;
     }
    if(adam[7]==CR && adam[3]=='D' && adam[4]=='O')
     {
     printf(" Command name: Set Digital Output \n");
     cam[data(0,3)]=CR;
     break;
     }

    else if(!rev_long(lm,4)) break;
    else if((adam[3]=='h'&&adam[4]=='i')||
      ((adam[3]=='L'||adam[3]=='D')&&adam[4]=='O'))
     {
     errtype();
     break;
     }
    else if(adam[3]=='D' && adam[4]=='A')
     {
     printf(" Command name: Disable Alarm \n");
     cam[data(0,3)]=CR;
     break;
     }
    else if(adam[3]=='C' && adam[4]=='A')
     {
     printf(" Command name: Clear Latch Alarm \n");
```

```
    cam[data(0,3)]=CR;
    break;
    }
  else if(adam[3]=='C' && adam[4]=='E')
    {
    printf(" Command name: Clear Event Counter \n");
    cam[data(0,3)]=CR;
    break;
    }
  else if(adam[3]=='D' && adam[4]=='I')
    {
    printf(" Command name: Read Digital I/O and Alarm Status \n");
    b=data(0,3);
    cam[b]=ascii[(char)random(3)];
    b++;
    cam[b]='0';
    b++;
    cam[b]=ascii[(char)random(4)];
    b++;
    cam[b]='0';
    b++;
    cam[b]=ascii[(char)random(2)];
    b++;
    cam[b]=CR;
    break;
    }
  else if(adam[3]=='R' && adam[4]=='H')
    {
    printf(" Command name: Read High Alarm \n");
    b=data(0,3);
    cam[data(b,0)]=CR;
    break;
    }
  else if(adam[3]=='R' && adam[4]=='L')
    {
    printf(" Command name: Read Low Alarm \n");
    b=data(0,3);
    cam[data(b,0)]=CR;
    break;
    }
  else if(adam[3]=='R' && adam[4]=='E')
    {
    printf(" Command name: Read Event Counter \n");
    b=data(0,3);
    cam[b]=ascii[(char)random(7)];
    b++;
    cam[data(b,2)]=CR;
    break;
    }
  else
    errcom();
  }
 printf("\n");
 transmite();
 }while(1);
}
```

APÉNDICE D. COTAS PARA LOS DIAGRAMAS DE TEMPORIZADO DEL MC68824-10 TOKEN BUS CONTROLLER

Cota	Descripción	Min	Max
1	Setup para entrada asíncrona	20	—
2	/UDS,/LDS inactivas a /CS,/IACK inactivas	—	100
3	CLK bajo (en el que se reconocen /UDS o /LDS, y /CS o /IACK) a datos de salida válidos[a]	—	200
4	/CS o /IACK alto a datos de salida en alta impedancia	—	60
5	Tiempo de sostenimiento de datos de salida después de /LDS/DS alto[b]	0	—
6	/IACK o /CS bajo a /DTACK alto	—	80
7	CLK bajo (en el que se reconocen /UDS o /LDS, y /CS o /IACK) a /DTACK bajo[a]	—	290
8	CLK bajo a /DTACK bajo	—	90
9	Datos de salida válidos a /DTACK bajo	20	—
10	/DTACK bajo a /UDS, /LDS, /CS , /IACK alto (la primera de ellas)	100	—
11	/CS o /IACK o Data Strobes[c] en alto (la primera de ellas) a /DTACK alto	—	60
12	/DTACK alto a /DTACK en alta impedancia (al final de un ciclo de bus)	—	50
13	Tiempo de inactividad de /UDS, /LDS	100	—
14	Tiempo de inactividad de /CS, /IACK	0	—
15	A1-A2 válidas a /UDS, /LDS, /CS bajo (la última de ellas) en un ciclo de escritura	30	—
16	Tiempo de sostenimiento de datos y de A1-A2 después de /DTACK bajo	100	—
17	/UDS o /LDS, /CS o /IACK bajo (la última de ellas) a datos de entrada válidos	—	80
18	R/W válido a /UDS o /LDS, /CS o /IACK bajo (la última de ellas)	20	—
19	/UDS, /LDS alto a R/W alto	0	—
25	/BGACK bajo a /BR en alta impedancia	20	—
26	Setup de /BG en la transición activa/inactiva con respecto a la bajada de CLK	20	—
27	CLK bajo a /BGACK bajo	—	60
30	/BG bajo a /BGACK bajo	220	380
31	/BR en alta impedancia a /BG alto	0	—
32	CLK en el que /BGACK bajo a CLK en el que /AS bajo	150	150
33	CLK bajo a /BGACK alto	—	55
38	CLK alto a direcciones válidas	—	100
39	Direcciones válidas a /AS válida	20	—
40	CLK alto a /AS, /UDS, /LDS bajo	—	50
41	CLK a /AS, /UDS, /LDS alto	—	55
42	/AS alto a direcciones/FC inválidas	20	—
44	CLK a R/W alto[d]	—	55
46	/UDS, /LDS alto a datos de entrada inválidos	0	—

47	/AS, /UDS, /LDS alto (primera de ellas) a /DTACK alto	O	1OO
48	Setup de datos de entrada con respecto a CLK cuando /DTACK satisface la cota 1	1O	—
49	Setup de /DTACK bajo a datos de entrada válidos cuando /DTACK no satisface la cota 1	—	65
50	CLK alto a R/W bajo	—	6O
51	/AS bajo a datos de salida válidos (ciclo de escritura)	—	9O
54	/UDS, /LDS alto a datos de salida inválidos	2O	—
55	Tiempo de sostenimiento de datos de salida con respecto a CLK alto	O	1OO

[a] Los datos y /DTACK se miden a partir del primer ciclo de reloj en el que /CS y algún Data Strobe son reconocidas para un ciclo MPU, y a partir del primer ciclo de reloj en el que /IACK y algún Data Strobe son reconocidas para un ciclo de interrupción

[b] Si /CS o /IACK son desactivadas antes que /UDS o /LDS, es probable que el bus esté en tercer estado antes de la desactivación de estas últimas

[c] Al utilizar un bus de 8 bits de datos sólo se considera a /LDS. Si se utiliza un bus de 16 bits de datos se deben desactivar ambas señales

[d] R/W sube al término de un ciclo de escritura. Si el TBC solicita el bus, entonces R/W se pasa a alta impedancia medio ciclo después. Cuando el TBC toma el bus, R/W está en alta impedancia durante la fase S1 y sube en esa fase