UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

FACULTAD DE INGENIERÍA ENSENADA

"Sistema de transferencia de datos a alta velocidad basado en una tarjeta PCI"

TESIS

Que para obtener el grado de:
Ingeniero en Electrónica

Presenta:
Luís Tomas Calvario Velásquez

Director:
M. I. Fernando Quirós Parra

Ensenada, Baja California, Abril 2007

Votos de aprobación de la tesis:

"Sistema de transferencia de datos a alta velocidad basado en una tarjeta PCI"

Presenta:
Luís Tomas Calvario Velásquez

M. I. Fernando Quirós Parra
Director

Dr. Carlos Gómez Agis
Sinodal

MC. Humberto Cervantes Ávila
Sinodal

A mis padres.

Prólogo

Durante el presente trabajo se muestra el desarrollo de un sistema de transferencia de datos punto a punto a alta velocidad basado en una tarjeta PCI: primeramente se estudian algunos principios de funcionamiento, posteriormente se desglosa el sistema, se detalla su integración y se presentaran resultados y sugerencias. El presente documento esta dividido en 6 capítulos:

En el primer capítulo, se presenta los antecedentes de estudio de este trabajo, así como la hipótesis, objetivos, importancia y limitaciones de este análisis. Se explica claramente el objetivo.

En el capítulo 2, se hace una revisión de la literatura ya existente y se discuten los sistemas comerciales que existen en el mercado y sus limitaciones, se analizan sus ventajas y desventajas, ampliando el panorama del campo de estudio de este trabajo.

En el tercer capítulo, se plantean requisitos y algunas cualidades con las que deberá contar el sistema desarrollado, tales como latencias, velocidad de transferencia, interfaz eléctrica etc.

A partir del capítulo 4, se tiene una idea clara del sistema propuesto; finalmente se detallan las características de los elementos que conformaran el diseño y las herramientas a utilizarse.

La quinta parte de este documento contiene información referente a la integración del sistema, las funciones a realizar por el hardware y software y al mismo tiempo se detalla como deben ser utilizados.

Para el capítulo 6, se muestran los resultados obtenidos y un análisis de ellos, se presenta una serie de sugerencias para futuras mejoras al sistema y se comentan las conclusiones finales.

Se añadieron tres apéndices que contienen información técnica para complementar lo expuesto a lo largo de todo el trabajo; o bien, para la implementación de dicho sistema.

Agradecimientos:

A los maestros que me brindaron la formación profesional que poseo, a aquellos que me aconsejaron dentro y fuera del salón de clase.

A todo el equipo de instrumentación que labora en el Instituto de Astronomía. Siempre hubo alguien que me tendiera la mano cuando lo solicite, o alguien que me chamaqueara cuando así lo necesitaba. En especial quiero agradecer a mi asesor, el M.I. Fernando Quiros Parra; quien se encargo de asesorarme durante todo el desarrollo de esta tesis.

Al personal que labora en el Observatorio Astronómico Nacional, Sierra San Pedro Mártir, por sus atenciones, compañerismo y carrillas que me brindaron durante mis estancias de trabajo.

A mis compañeros y amigos, que aunque me aleje durante mucho tiempo de ellos, siempre estuvieron conmigo. Gracias Yohanna, Alejandro, Miguel, Memo, Carol, Diego, Wilson, Paul, Lina. Gracias Mayra.

Principalmente quiero agradecer a mis padres, por el apoyo brindado durante toda mi vida. Siempre estuvieron dispuestos a apoyarme y han sido un ejemplo a seguir. Es por ellos que he llegado aquí y llegare tan lejos como pueda.

Indice

Lista de Figuras

Lista de Tablas

1. Introducción

1.1 Antecedentes

Durante las últimas décadas, los avances en la investigación astronómica nos han brindado una mejor comprensión acerca del universo en que vivimos. Los grandes telescopios y los instrumentos de alta tecnología han jugado un papel muy importante en la comprobación de las hipótesis formuladas por la comunidad de astrónomos alrededor del mundo.

Un instrumento ampliamente utilizado es el CCD (por sus siglas en inglés: Charge Coupled Device). Este dispositivo es un arreglo de celdas capaces de almacenar una carga eléctrica en proporción a la luz que reciben. Al transformar estas cargas en voltajes, que a su vez son convertidos en datos binarios, es posible crear una imagen digital que corresponde a lo que observaría un astrónomo a través del ocular de un telescopio. La ventaja fundamental de contar con esta imagen digital, es la posibilidad de analizar y procesar su contenido mediante el uso de una computadora.

Un sistema de adquisición de imágenes basado en un CCD, esta formado por una serie de dispositivos electrónicos que llevan a cabo diversas funciones dentro del sistema. Entre estos dispositivos se encuentran controladores, secuenciadores, convertidores analógico-digitales y una interfaz hacia una computadora para transferir y almacenar la información que el CCD ha captado. Muchas veces, el desempeño general de un sistema de adquisición de imágenes es limitado por esta comunicación y por lo tanto es de suma importancia contar con una interfaz que resuelva satisfactoriamente el problema de la transferencia de datos entre el sistema de adquisición y el de almacenamiento y procesamiento. Por lo general, este canal de comunicación utiliza algún protocolo ya existente, es decir, el canal de comunicación estará montado en un

ducto o interfaz de computadora, tales como un puerto paralelo, interfaz serial RS-232, ducto ISA, ducto PCI, interfaz de red, fibra óptica, etc. En la siguiente figura se muestra un diagrama sencillo que muestra los elementos más importantes en un sistema de adquisición de imágenes basado en CCD.

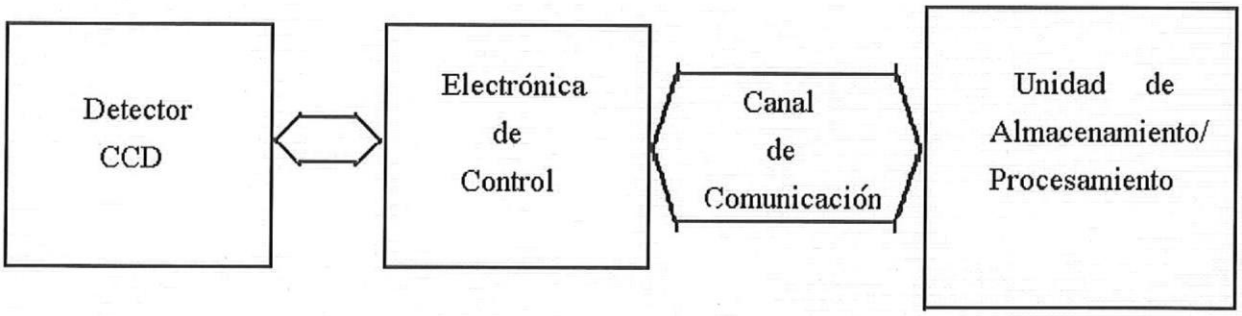


Figura 1. Diagrama general para un sistema de adquisición de imágenes basado en CCD.

La interfaz al canal de comunicación por lo general es creada para un CCD en específico y por lo tanto no puede simplemente ser utilizada por otro sistema. La comunicación se ha llevado a cabo mediante diversos ductos de computadora y conforme al criterio de quien desarrolla el instrumento; por ejemplo, el equipo encabezado por Robert Leach de la SDSU (San Diego State University) ha utilizado principalmente el ducto VME, típico en las computadoras SUN usadas por los astrónomos para quien se diseñaron los sistemas, aunque también existe una adaptación de tipo SCSI implementada sobre una tarjeta PCI. Compañías dedicadas al desarrollo de instrumentos han utilizado otros medios, tal como el Santa Barbara Instrument Group que se ha inclinado por el puerto paralelo y el nuevo ducto USB (Universal Serial Bus) en sus más recientes productos; o la compañía Apogee que se dedica al desarrollo de CCD's, también utiliza el ducto USB e interfaces de red Ethernet en sus diseños.

1.2 Planteamiento del problema

Uno de los principales problemas en la adquisición de imágenes de alta resolución, la cual implica grandes cantidades de información, es el medio de transferencia de dicha información hacia las unidades de almacenamiento y/o procesamiento, debido al gran volumen de información involucrado en la transferencia.

En el área de la astronomía es de suma importancia la velocidad de la transferencia, ya que cada imagen obtenida llega a ocupar hasta 8 MBytes (para un detector de 2000 por 2000 píxeles), y este dato va en aumento con el desarrollo de los nuevos detectores CCD's de mayor tamaño.

1.3 Hipótesis.

Mediante el uso del ducto PCI de una computadora personal (PC), es posible diseñar e implementar una tarjeta que sea capaz de manejar datos de entrada y salida a alta velocidad, en el orden de 3 MBytes por segundo (MB/S).

1.4 Objetivos

Diseñar y construir una tarjeta PCI y toda la programación asociada bajo un sistema operativo LINUX, para el manejo de entrada y salida de datos a alta velocidad.

1.5 Importancia del estudio

La mayoría de las compañías comerciales que producen tarjetas de entrada y salida de datos digitales, no cumplen con las características del sistema que se necesita en el Instituto de Astronomía de la UNAM (IA-UNAM).

El sistema que se propone desarrollar es de suma importancia en el desarrollo de controladores de CCD's, ya que actualmente la transferencia entre el controlador de CCD's y la unidad de procesamiento de imágenes es una de las limitantes para obtener el desempeño óptimo de dicho sistemas. Además, este trabajo es un punto de arranque para futuros proyectos que involucren al ducto PCI como medio de comunicación entre sistemas de procesamiento de información y los instrumentos que apoyan el quehacer científico nacional.

1.6 Limitaciones del estudio

El trabajo se limitará al desarrollo bajo sistemas PC con procesadores compatibles con INTEL, que utilicen sistemas operativos LINUX con Kernel de la familia 2.4 y superiores.

2 Revisión de literatura

2.1 Introducción

La noción de un ducto de computadora evolucionó a principio de la década de 1960, al lado de la minicomputadora. En ese entonces, la minicomputadora fue un avance radical en la arquitectura de computadora. Previamente, la mayoría de las computadoras eran únicas en su clase; es decir, máquinas construidas según las necesidades de una aplicación; y tenían relativamente pocos periféricos: un lector de cintas de papel y una perforadora, un teclado, una impresora de línea y a lo mucho un disco. La lógica de interfaz a los periféricos estaba estrechamente unida a la lógica del procesador.

Con la llegada del circuito integrado (CI), el tamaño de la Unidad de Procesamiento Central (por sus siglas en ingles: Central Processing Unit) se redujo de un gabinete del tamaño de un refrigerador a uno o dos circuitos impresos. La electrónica de las interfaces a dispositivos periféricos se redujo de la misma forma. Fue posible entonces que las computadoras se podían montar en una línea de ensamblaje, pero solo si este ensamblaje fuera eficiente. Los ingenieros de la época rápidamente reconocieron la solución al problema: diseñar todas las tarjetas con un protocolo eléctrico y de interfaz común. Ensamblar una computadora hoy en día es cuestión de conectar tarjetas a una tarjeta madre, la cual cuenta con un CPU y un ducto integrado en la misma tarjeta, dicho ducto consiste en un gran número de pistas paralelas y sus respectivos conectores.

El bus ó ducto de computadora también solucionó un problema de mercadeo. Una sola compañía tiene recursos limitados para orientarse a un grupo reducido de aplicaciones potenciales para la computadora. Los fabricantes resolvieron este problema haciendo públicas las especificaciones de sus buses; motivando a las otras compañías a construir equipo compatible.

2.2 Antecedentes

Fundamentalmente, un bus consiste en un juego de 'líneas' paralelas unidas a varios conectores donde las tarjetas de periféricos pueden ser conectadas. Véase figura 2. Típicamente; el procesador esta conectado en un extremo de estas líneas, aunque a veces la memoria RAM también puede ser conectada al bus.

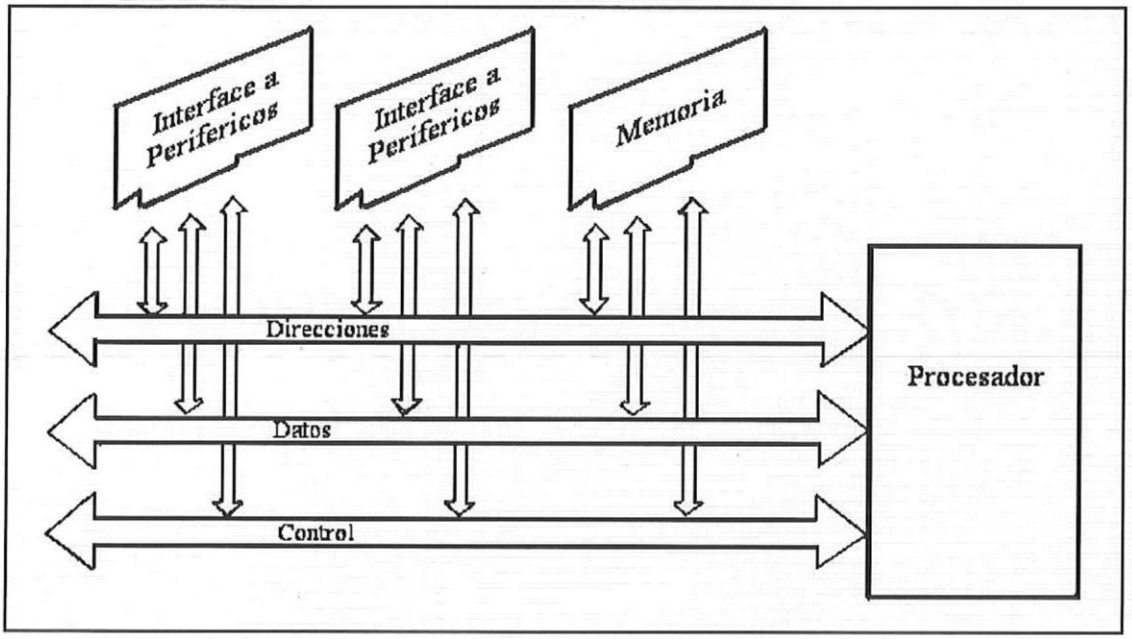


Figura 2. Diagrama Básico de un Ducto de Computadora

Estas líneas pueden ser separadas en varios grupos funcionales:

- Direcciones: Especifican el periférico y registro dentro del periférico que se acceda.
- Datos: La información que se transfiere desde o hacia el periférico.
- Control: Señales que afectan a la operación de transferencia de datos. Es en estas líneas y en como se manipulan donde radica el protocolo del bus.

Mas allá de la transferencia de datos, los buses típicamente incorporan opciones avanzadas, tales como:

- Interrupciones
- Acceso Directo a Memoria (DMA)
- Distribución de potencia

Algunas líneas de control adicionales manejan estas opciones.

El concepto clásico de un bus consiste en un juego de tarjetas conectadas a una tarjeta madre pasiva como la que se muestra en la figura 2. También hay otras implementaciones de buses basados en cables que interconectan componentes separados. El GPIB (General Purpose Interface Bus) es un ejemplo clásico. Un ejemplo contemporáneo de buses de cables incluyen al USB y al IEEE-1394 (marca registrada de Apple Computer bajo el nombre de FireWire). Tampoco las tarjetas madres tienen que ser forzosamente pasivas, como la implementación típica de una tarjeta madre de computadora personal.

2.3 Taxonomía del ducto.

Los ductos de computadora pueden ser clasificados en diferentes familias. Pueden tener una arquitectura serie o paralela. De acuerdo con el tipo de arquitectura, los buses se pueden separar en dos dimensiones binarias: síncronos o asíncronos y multiplexados o no-multiplexados.

En un bus síncrono, todas las operaciones ocurren en un flanco específico de una señal de reloj maestra. En los buses asíncronos las operaciones ocurren en flancos específicos de las señales de control sin depender de un reloj maestro. Los primeros buses tendieron a ser asíncronos, mientras que hoy en día los buses son generalmente síncronos.

En un bus multiplexado los datos y las direcciones comparten las mismas líneas. Las señales de control identifican en que momento el bus común contiene datos o información sobre

direcciones. Un bus no-multiplexado tiene líneas separadas para datos y direcciones.

La ventaja básica de un bus multiplexado radica en tener menos líneas, que a su vez significa menos pins en cada conector, menos circuitos de potencia para cada línea, etcétera. Su principal desventaja es que requiere dos fases para llevar a cabo una sola transferencia de datos: primero una fase de direcciones es enviada, posteriormente la información es enviada. Los buses contemporáneos están repartidos equitativamente entre multiplexados y no-multiplexados.

La tabla 1 enlista algunos de los parámetros cuantificables en el diseño de un bus. Los cuales pueden ser clasificados en términos de las líneas de direcciones y datos. Los buses actuales son típicamente de 32 ó 64 bits para direcciones y datos. No debe sorprendernos que los buses multiplexados tiendan a tener el mismo número de bits para direcciones y datos.

Tabla 1. Algunos parámetros utilizados para clasificar ductos electrónicos

Ancho de direcciones	3,8,16,32,64
Ancho de datos	1,8,16,32,64
Frecuencia	1 MHz a cientos de MHz
Ancho de banda	1 MB/s a cientos de MB/s
Número de dispositivos	1 a muchos

Un elemento clave de cualquier bus es su desempeño. ¿Que tan rápido puede transferir datos? Los primeros buses estaban limitados a unos pocos MHz, que correspondían al desempeño del procesador de aquella época; el problema en los sistemas actuales es que el microprocesador es varias veces más rápido que el bus, convirtiéndose éste en un cuello de botella para el desempeño general del sistema. El largo del bus está relacionado a la velocidad de transferencia. Los primeros buses con velocidades de uno o dos MHz permitían longitudes

máximas de varios metros, pero con velocidades mayores llegaron longitudes más cortas para que el retraso de propagación no impactara negativamente al desempeño.

El número máximo de dispositivos que pueden conectarse a un bus también es restringido por consideraciones para un alto desempeño. Los primeros buses podían tolerar manejadores de alta potencia y relativamente lentos, que podían entonces dar lugar a un gran número de dispositivos conectados. Buses de alto desempeño tales como el PCI limitan la potencia de los manejadores y por ello son severamente limitados en términos del número de dispositivos conectados a un mismo ducto.

2.3.1 Ducto ISA

El bus PCI evolucionó, al menos en parte, en respuesta a las limitaciones del entonces venerado bus ISA (Industry Standar Architecture). A su vez, el bus ISA fue una mejora revolucionaria del bus definido por IBM para su primera computadora personal. Este se acomodaba perfectamente al desempeño del procesador y los requisitos de los periféricos de las primeras computadoras personales.

El bus ISA comenzó a ser insuficiente cerca de 1992 cuando Windows se convirtió en el sistema operativo dominante. Para ser realmente efectiva, la interfaz gráfica requiere mucho más de los 8MB/s de los que es capaz el bus ISA. Su bus de datos de 16 bits es un cuello de botella para los procesadores de 32 bits contemporáneos. También, la caída de los precios de la memoria dinámica (DRAM) y el intenso requerimiento de memoria de la computación gráfica, rápidamente hicieron que el espacio de direcciones de 16MB del ISA se volviera insuficiente.

Otra limitante, fue la manera de configuración de las tarjetas en el bus ISA. Los periféricos ISA dependen principalmente de interruptores para resolver conflictos que involucran

espacios de direcciones de entrada/salida (I/O), asignación de canales de interrupciones y DMA. Una configuración exitosa dependía de una amplia comprensión de los dispositivos y como interactúan. Este nivel de habilidades es el esperado de entusiastas y aficionados, pero es completamente inaceptable en un producto para el mercado masivo.

Este ducto tuvo una extensión de 16 bits: El EISA (Extended ISA), el cual es compatible con el ISA anterior, con lo que se dobló el ancho de banda que este permitía; sin embargo, no cumplió con los requerimientos de su época.

2.3.2 Ducto local VESA

El bus local VESA, puesto en el mercado en 1992 y promocionado por la Video Electronics Standards Association, fue uno de los primeros intentos por superar las limitaciones del ISA. La estrategia del bus VL (VESA Local) es adjuntar el controlador de video, y posiblemente otros dispositivos de alto ancho de banda, directamente al bus local del microprocesador, ya sea directamente o a través de un reforzador. La conexión directa permite un solo dispositivo, mientras que la versión reforzada soporta hasta tres dispositivos. Véase la figura 3 para más detalles.

El bus VL resolvió el problema del ancho de banda (a corto plazo solamente). En un bus de microprocesador a 33Mhz, el VL podía alcanzar 132 Mbytes/s. El VESA además hizo un intento por resolver el problema de configuración, solicitando que todos los dispositivos VL debieran contar con un sistema de configuración automática; desafortunadamente, no se molestaron en desarrollar un protocolo de configuración, por lo que cada fabricante inventó el propio.

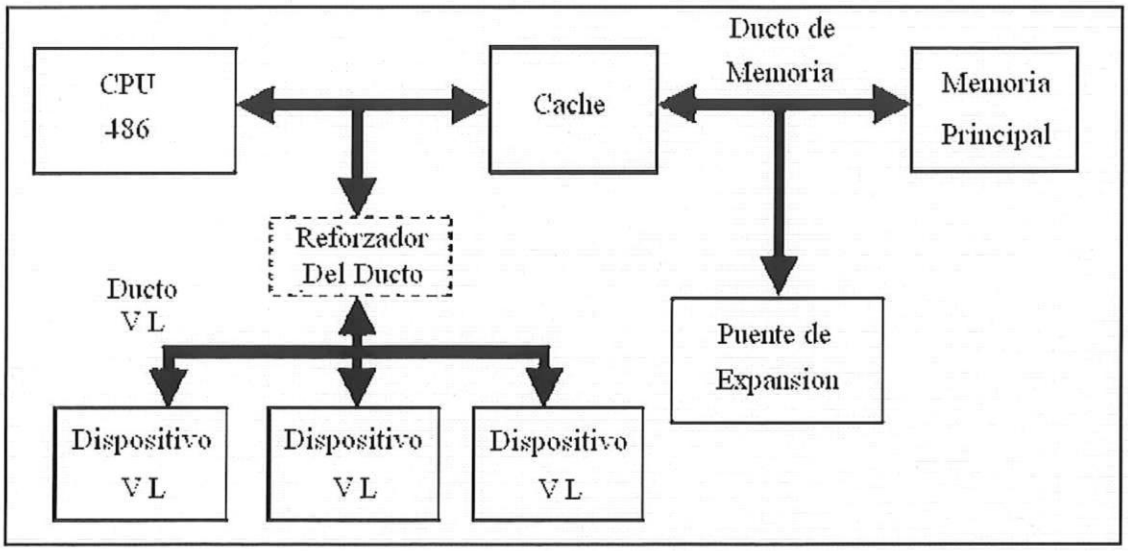


Figura 3. Diagrama de un Ducto VL

El VESA tampoco especifica las características eléctricas de los dispositivos VL, solo se esperaba que fueran compatibles con el bus del 80486. Pero la principal desventaja del bus VL radica en su dependencia del procesador. En cuanto apareció el Pentium, en 1995; perdió relevancia y un año más tarde fue eliminado del mercado junto a la familia 80486.

2.3.3 Ducto PCI

La compañía Intel, desarrolló la especificación original del bus PCI en un intento por poner orden en un mercado caótico. Intel decidió no seguir con el VL, ya que falló en tener una visión a largo plazo de los problemas emergentes y las tendencias en el desarrollo de la arquitectura de la PC.

La revisión 1 de la especificación apareció en junio de 1992. La revisión 2.0 en abril de 1993, seguida de la 2.1 a principios de 1995 y finalmente la versión actual, 2.2, que fue publicada en febrero de 1999.

2.3.3.1 Características

El bus PCI implementa características visionarias que lo mantienen en relevancia aun hoy en día:

- La máxima frecuencia de transferencia de la configuración base es de 132 MB/s. La extensión actual puede acelerar esta velocidad hasta 528MB/s.
- Cualquier dispositivo en el bus puede actuar como el 'bus master' e iniciar transacciones. Una consecuencia de esto, es que no hay necesidad de la noción tradicional de DMA.
- El protocolo de transferencia está optimizado para transferir bloques de datos. Una transferencia sencilla es una transferencia de un bloque con un solo dato.
- Aunque es oficialmente independiente del procesador, inevitablemente refleja sus orígenes en Intel y sus principales aplicaciones en la arquitectura de la PC. Entre otras cosas usa ordenación de bytes de 'little-endian' (el bit menos significativo a la izquierda) tal como los procesadores Intel.
- El bus PCI permite una configuración tipo Plug and Play. Todo dispositivo en un sistema se configura automáticamente cada vez que el sistema es encendido. El protocolo de configuración soporta hasta 256 dispositivos en un sistema.
- Las especificaciones eléctricas estimulan el uso de baja potencia, dando soporte para ambientes de señalización a 3.3 y 5V.

El bus PCI esta embebido en un juego de especificaciones mantenidas por la asociación PCI Special Interest Group, una asociación no corporativa formada por cientos de compañías miembros en todo el mundo representando todos los aspectos de la industria de las microcomputadoras.

2.4 Sumario

Hasta ahora se han revisado algunas de las características de los ductos que han aparecido en el constante desarrollo de las computadoras personales. A modo de resumen, en la tabla 2 se muestra una comparación entre las características de cada uno de ellos.

Aunque en la comparación pueda apreciarse una pequeña diferencia en el ancho de banda entre el ducto local VESA y entre el ducto PCI, es importante recordar que el ducto local VESA fue diseñando específicamente para el procesador 80486 de Intel, hoy en día obsoleto y fuera del mercado.

Tabla 2. Comparación entre diferentes ductos de computadora.

	Ducto ISA-EISA	Ducto Local VESA	Ducto PCI
Frecuencia de reloj	4.77-8.33 MHz	Reloj del cpu (486,50MHz max)	33Mhz, 66Mhz
Ancho de datos (bits)	16	32	32,64
Ancho de banda máximo	16MB/s	200MB/s max	132MB/s-528MB/s
Ancho de direcciones (bits)	24 (hasta 20 para I/O)	30	32,64
Número de dispositivos	8	1-3	256
Alimentación	+5 V, -5 V, +12 V, -12 V	5V	5V, 3.3V

Se ha omitido en esta comparación los ductos serie USB e IEEE1394 (FireWire™) y el ducto paralelo GPIB debido a que generalmente el hardware anfitrión de estos ductos se encuentra a su vez conectado a otro ducto anfitrión; hoy en día, al ducto PCI.

Por todo lo anterior, la mejor opción para desarrollar una tarjeta de entrada y salida de datos a alta velocidad es la utilización del ducto PCI.

3 Requerimientos mínimos para la tarjeta de adquisición

3.1 Introducción

Se hizo mención durante la introducción del presente documento, de la necesidad de un sistema de adquisición de datos a alta velocidad, con una interfaz flexible y adecuada a los CCD's y otros instrumentos posibles utilizados en el Instituto de Astronomía. Además, el resultado final de esta investigación deberá proveer una opción de comunicación rápida y confiable para instrumentos futuros, razón por la cual, la interfaz al exterior de nuestra tarjeta debe tener una arquitectura estandarizada para una adaptación sencilla.

3.2 Interfaz al exterior

El sistema debe contar con un canal de datos de 16 bits bidireccionales, un medio para seleccionar entre diferentes dispositivos o instancias de algún periférico, ya sea en forma de bits de direcciones o un grupo de señales tipo CHIP SELECT o ambas. Además, se deberá proporcionar señales de escritura (WR#) y de lectura (RD#). Los niveles de voltaje deberán ser compatibles con lógica TTL, es decir, utilizar niveles de voltaje discretos de 0 y 5 Volts.

3.3 Velocidad de transferencia

Se desea que la velocidad de transferencia supere a los sistemas de comunicación utilizados actualmente por los instrumentos utilizados en el Observatorio Astronómico Nacional. El medio mas rápido utilizado hasta la fecha ha sido una interfaz de red tipo Ethernet, que trabaja a 100Mb/s. Si bien esto parece ser una velocidad de transferencia bastante rápida, en realidad se

trata de aproximadamente 12 Mbytes/s, de los cuales una buena parte de ellos es destinado a encabezados de paquetes, direcciones, etc.

3.4 Tiempos muertos

Debido a que la tarjeta además de transferir datos, deberá enviar información de control a los periféricos, el tiempo muerto entre una operación y otra debe ser minimizado. Un ejemplo en que el tiempo muerto tiene un serio efecto negativo, es en el caso de controlar la sección de conversión analógico-digital de un CCD: la carga almacenada en una fotocelda del CCD es integrada durante un tiempo definido, siendo el resultado de esta operación capturado por el convertidor A/D. Si se controla dicho tiempo de integración, una variación en tiempo entre una fase de control y otra podría resultar en un muestreo erróneo si el tiempo de integración se prolonga más de lo necesario.

3.5 Latencias

Latencia es el tiempo máximo en ciclos de reloj; en que un dispositivo debe contestar a una petición y llevarla acabo. Los tipos más importantes de latencia son el de espera de respuesta por parte del dispositivo solicitado y el del tiempo en ciclos de reloj como máximo que una transacción puede durar; ya que el ducto no puede ser utilizado por un solo dispositivo por mucho tiempo. El controlador de ducto PCI S5935 de la compañía AMCC puede responder a una petición antes de 8 ciclos de reloj y realizarla en menos 8, ubicandose en la categoría de dispositivos rápidos (uno lento lo hace en 32 ciclos). Algunos problemas pueden surgir en caso de lecturas a periféricos, si estos no pueden responder rápidamente a las peticiones hechas. Se desea que nuestro sistema pertenezca a la categoría de dispositivos rápidos o de baja latencia.

4 Arquitectura propuesta

4.1 Introducción

Se propone desarrollar un sistema de adquisición de datos a alta velocidad, basado en una tarjeta de desarrollo del controlador de ducto PCI y lógica de acoplamiento con el fin de proveer al exterior una interfaz tipo IDE (Integrated Drive Electronics).

Esta interfaz IDE (también conocido como ATA; AT Attachment) fue concebido originalmente para conectar dispositivos de almacenamiento tales como discos duros, unidades de disco compacto, unidades de disco ZIP, etcétera, a las computadoras PC/AT. El IDE cuenta con líneas de control independientes, grupos de líneas de direcciones y datos, y líneas adicionales de direcciones para seleccionar entre diferentes dispositivos conectados a un ducto IDE en común.

La interfaz externa deberá incluir un ducto de datos de 16 bits bidireccionales, 3 bits de direcciones independientes a las líneas de datos, así como 3 líneas típicas de control: RD# (read, lectura), WR# (write, escritura) y CS# (chip select, selección de dispositivo).

El objetivo se perseguirá valiéndose de una tarjeta PCI de desarrollo cuyo elemento más importante es el mismo controlador del ducto PCI: el S5953. Además, se utilizará un dispositivo lógico programable complejo CY37128.

En base a lo señalado en párrafos anteriores, el desarrollo del sistema será orientado hacia un diseño final que cuente con un grado de versatilidad: además de ser capaz de sustraer información de un CCD, también contará con la posibilidad de controlar el mismo, convirtiéndolo en un sistema completo para la adquisición de imágenes.

4.2.- El Ducto PCI

Dado que en el presente documento se detalla el desarrollo de un sistema que opera sobre el ducto PCI, a continuación se explicará de manera general el protocolo que hay que seguir para realizar una transferencia de datos; con lo que posteriormente se logrará una mejor comprensión acerca de la tarea compleja que lleva a cabo el controlador de ducto PCI utilizado.

4.2.1 Señales PCI

La figura 3 muestra las señales definidas en el bus PCI. Una interfaz PCI requiere un mínimo de 47 pins para un dispositivo esclavo y 49 para un maestro. Esto es suficiente para el paso de datos de 32 bits a 33Mhz y es imperante que todos los dispositivos que reclaman ser compatibles con PCI[1]. Pins adicionales definen características opcionales como transferencias de 64 bits, interrupciones y una interfaz JTAG (Joint Test Action Group).

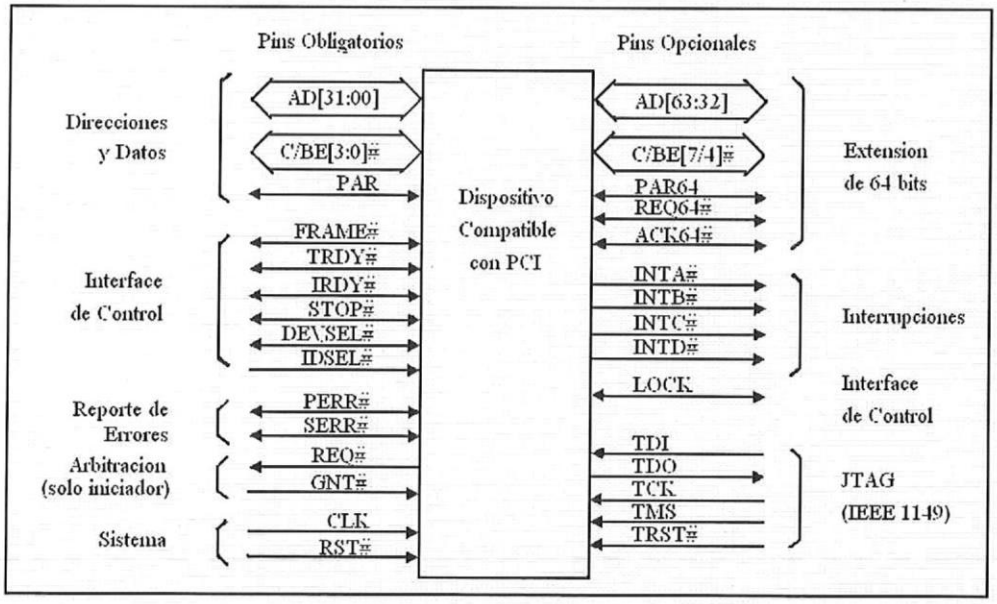


Figura 4. Señales PCI

[1] Para información sobre la notación utilizada, léase el apéndice D.

4.2.2 Grupos de señales

Para propósitos de definición, las señales PCI pueden ser clasificadas en varios grupos funcionales.

4.2.2.1 Sistema

CLK, proporciona temporización para todas las transacciones en el bus y es una entrada para todo dispositivo PCI. Todas las otras señales excepto RST# y de INTA# a INTD# son muestreadas en el flanco de subida de CLK. (in),

RST#, lleva registros específicos PCI, secuenciadores y señales a un estado consistentes. Cuando RST# es activada, todas las señales de salida deben ser llevadas a su estado benigno. En general, esto significa que deban ser puestas en tercer estado. (in),

4.2.2.2 Direcciones y Datos

AD[31::0], Direcciones y datos son multiplexados en el mismo juego de pins. Una transacción PCI consiste en una fase de direcciones y una o más fases de datos. (t/s)

C/BE[3::0], Los comandos para el bus y habilitadores de bytes están multiplexados en los mismos pins. Durante la fase de direcciones de una transacción, C/BE#[3::0] definen un comando para el bus. Durante cada fase de datos, C/BE#[3::0] son usados como habilitadores de bytes para determinar que líneas de bytes llevan datos validos. C/BE#[0] habilita el byte 0 (menos significativo) y C/BE#[3] al byte 3 (más significativo). (t/s).

PAR, Paridad par a través de AD[31::0] y C/BE#[3::0]. Se requiere que todos los agentes PCI generen paridad. (t/s)

4.2.2.3 Interfaz de Control

Frame#, Manejada por el maestro actual para indicar el principio y duración de una transacción. La transferencia de datos continua mientras FRAME# esta activa. Cuando FRAME# es desactivada, la transacción está en su última fase o ha sido completada. (s/t/s),

IRDY#, Initiator Ready (iniciador listo) indica que el bus master es capaz de completar la fase de datos actual. Durante una escritura, IRDY# indica que el master está preparado para aceptar datos. (s/t/s),

TRDY#, Target Ready, Indica que el dispositivo destino es capaz de completar la fase de datos actual. Durante una lectura, TRDY# indica que hay datos validos presentes en AD[31::0]. Durante una escritura, indica que el destinatario esta listo para aceptar datos. Una fase de datos culmina en cualquier ciclo de reloj durante el cual IRDY# y TRDY# están activos. (s/t/s),

STOP#, Indica que el destinatario seleccionada solicita al master para terminar la transacción actual. (s/t/s),

LOCK#, Indica una operación atómica que puede requerir múltiples transacciones para completarse. (s/t/s),

IDSEL, Initializacion Devide Select. Es un chip select durante transacciones de configuración.

DEVSEL#, Device Select Indica que un dispositivo ha decodificado su dirección como el destino de la transacción actual. (s/t/s),

4.2.2.4 Arbitración

REQ#, Request Indica al arbitro central que un agente desea usar el bus. Cada bus master

potencial tiene su propia señal punto a punto REQ#,

GNT#, Grant. Indica a un agente que está activando su señal REQ# que el acceso al bus le ha sido concedido. Cada bus master potencial tiene su propia señal punto a punto GNT#.

4.2.2.5 Reporte de errores

PERR#, Se utiliza para reportar errores de paridad en todas las transacciones excepto durante un Ciclo Especial,

SERR#, System Error. Se utiliza para reportar errores de paridad en direcciones, errores de paridad en datos en comandos de ciclos especiales, y en cualquier otro error de sistema potencialmente catastrófico.

4.2.2.6 Interrupciones (opcionales)

INTA# a INTD# son usadas por un dispositivo para solicitar atención por controlador de dispositivo (device driver, software). Un dispositivo de una sola función solo puede usar INTA#, mientras que dispositivos multifuncionales pueden usar una combinación de señales INTx#. (o/d).

4.2.2.7 Señales Adicionales

Estas señales no son parte del protocolo básico PCI, pero implementan características adicionales que son útiles en ciertos ambientes operativos.

PRSNT[1::2]#. Estos están definidos para tarjetas de expansión pero no para tarjetas madres. Estas señales indican a la tarjeta madre que una tarjeta esta físicamente presente y, si lo está; sus requisitos de potencia totales. Todas las tarjetas están obligadas a aterrizar una o ambas de

estas señales como se muestra a continuación en la tabla 3:

Tabla 3. Señales de requerimientos de potencia para una tarjeta PCI.

PRSNT#1	PRSNT#2	Estado
Abierto	Abierto	Sin tarjeta presente
Tierra	Abierto	Presente, 25 W máximo
Abierto	Tierra	Presente, 15 W máximo
Tierra	Tierra	Presente, 7.5 W máximo

Las tarjetas de expansión deben hacer uso de las señales PRSNT#[1::2], pero son opcionales para las tarjetas madre.

CLKRUN#, Clock URNG. Es una entrada opcional a un dispositivo para determinar el estado de CLK. Es una salida para un dispositivo que desea controlar el estado del reloj. Su activación significa que el reloj está corriendo a su velocidad normal. Su desactivación es una petición para reducir la velocidad o detener al reloj. Esto sirve para implementar un mecanismo de ahorro de energía en sistemas móviles y es descrito más a fondo en el PCI Mobile Design Guide[2]. El conector estándar del bus PCI no tiene un pin para esta señal. (in,o/d,s/t/s).

Cada una de las señales mostradas con anterioridad incluye una variedad de iniciales al final de su descripción (in, out, o/d, etcétera). Estas iniciales indican el tipo de señal. En el apéndice D, se anexa una descripción detallada acerca de estos tipos.

4.2.3 Protocolo del ducto

La esencia de cualquier ducto es el juego de reglas mediante las cuales la información

[2] Consulte la pagina http://www.pci.org

se mueve entre los dispositivos. Este juego de reglas es llamado protocolo. A continuación se describirá el protocolo que controla la transferencia de información dentro de un ducto PCI

4.2.3.1 Comandos del ducto PCI

El comando PCI para una transacción está contenido en las líneas C/BE# durante la fase de direcciones. Cuando C/BE# lleva un comando son activas en alto (nivel alto = lógica 1) mientras que cuando lleva información acerca de habilitadores de bytes es de activación en bajo.

El ducto PCI define 3 espacios diferentes de dirección que con sus correspondientes comandos de lectura y escritura como se muestra en la tabla 4. La principal diferencia entre espacio de entrada-salida y de memoria es que este ultimo es generalmente considerado "prefetchable" (esta memoria puede ser utilizada por el sistema anfitrión) y por lo tanto las lecturas al espacio de la memoria no tienen efectos secundarios. El espacio de direcciones de configuración solo es usado al momento de arranque para configurar la comunidad de tarjetas PCI en un sistema.

Hay algunos comandos adicionales de lectura/escritura que solo afectan al espacio de memoria "prefetchable". El propósito de Memory Read Line es decirle al destinatario que el master quiere leer la mayoría, si no es que toda la línea del cache actual. El destinatario puede ganar algo de ventaja en su desempeño si sabe que se espera del proveer hasta una línea de cache entera. Cuando un master invoca un comando Memory Read Multiple, está diciendo que quiere leer más de una línea de cache antes de desconectarse.

Memory Write and Invalidate es semánticamente idéntica a Memory Write con la adición de que el master se compromete a escribir una línea de cache completa en una sola transacción PCI. Esto es útil cuando una transacción trata de escribir en una línea 'sucia' del cache.

El comando de Interrupt Acknowledge es una lectura implícitamente direccionada al controlador de interrupciones del sistema. El contenido del bus AD es irrelevante durante la fase de direcciones y C/BE# indican el tamaño de el vector regresado durante la fase de datos correspondiente.

Tabla 4. Comandos PCI durante la fase de direcciones.

C/BE#3	C/BE#3	C/BE#3	C/BE#3	Tipo de Comando
0	0	0	0	Interrupt Acknowledge
0	0	0	1	Ciclo Especial
0	0	1	0	I/O Read
0	0	1	1	I/O Write
0	1	0	0	Reservado
0	1	0	1	Reservado
0	1	1	0	Memory Read
0	1	1	1	Memory Write
1	0	0	0	Reservado
1	0	0	1	Reservado
1	0	1	0	Configuration Read
1	0	1	1	Configuratión Write
1	1	0	0	Memory Read Multiple
1	1	0	1	Dual-Address Cycle
1	1	1	0	Memory Read Line
1	1	1	1	Memory Write and Invalidate

El comando Special Cycle provee un mecanismo de difusión de mensajes como una alternativa para separar señales físicas para comunicación en banda lateral. El comando Dual Address Cycle (DAC) es una manera de transferir una dirección de 64 bits sobre una base de 32.

4.2.3.2 Transacciones Básicas de Lectura-Escritura

La figura 5 muestra el diagrama de tiempo de una transacción típica de lectura, es decir, una donde se transfiere información del destinatario al iniciador.

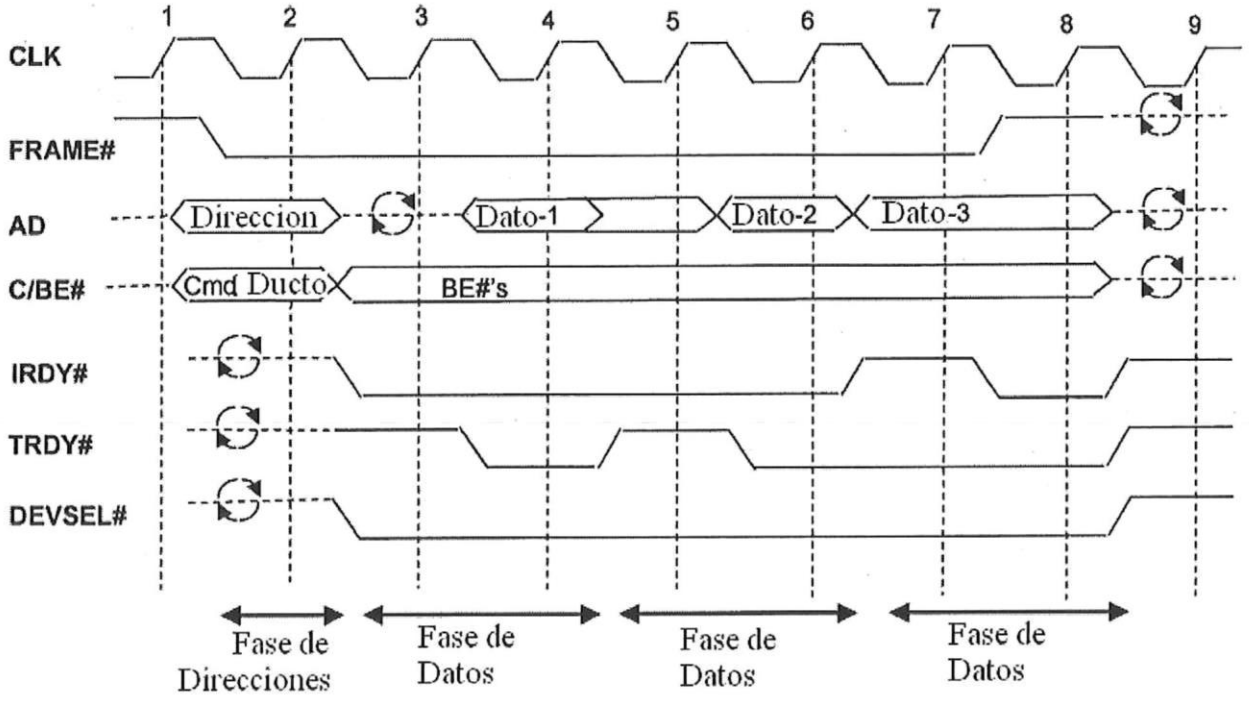


Figura 5. Diagrama de tiempo para una operación de lectura en el ducto PCI.

A continuación, se detallará ciclo a ciclo el protocolo a realizar durante la operación de lectura, especificando el papel de cada señal durante la transferencia.

Ciclo

1 El bus está desocupado y la mayoría de las señales están en tercer estado. El master ha recibido su GNT# por lo que pone FRAME# en alto.

2 Fase de direcciones: El master pone FRAME# en bajo y pone una dirección de destinatario en el bus AD y un comando de bus en C/BE#. Todos los posibles destinos capturan la dirección y el comando.

3 El master activa las líneas apropiadas del bus C/BE# y además activa IRDY#. El dispositivo que reconoce su dirección en el bus AD activa DEVSEL#. Cada vez que más de un dispositivo pueda manejar una línea del ducto PCI, se requiere de una espera de un ciclo, para evitar posibles choques que resulten en picos de ruido y consumo innecesario de energía.

4 El destinatario pone datos en el bus AD y entonces activa TRDY#. El master captura los datos y la transferencia de datos es ejecutada en cualquier ciclo de reloj en la que IRDY# y TRDY# son activadas.

5 El destinatario desactiva TRDY# indicando que el siguiente elemento de datos no esta listo para transferirse.

6 El destinatario ha puesto el siguiente elemento de datos en el bus AD y ha activado TRDY#. IRDY# y TRDY# son activados y el master captura lo que esta en el bus de datos.

7 El master ha desactivado IRDY# indicando que no esta listo para el siguiente elemento de datos.

8 El master ha activado IRDY# nuevamente y desactivado FRAME# para indicar que es el ultimo elemento a transferirse. En respuesta, el destinatario desactiva AD, TRDY# y DEVSEL#. El master desactiva C/BE# e IRDY#. Esta es una terminación iniciada por el master.

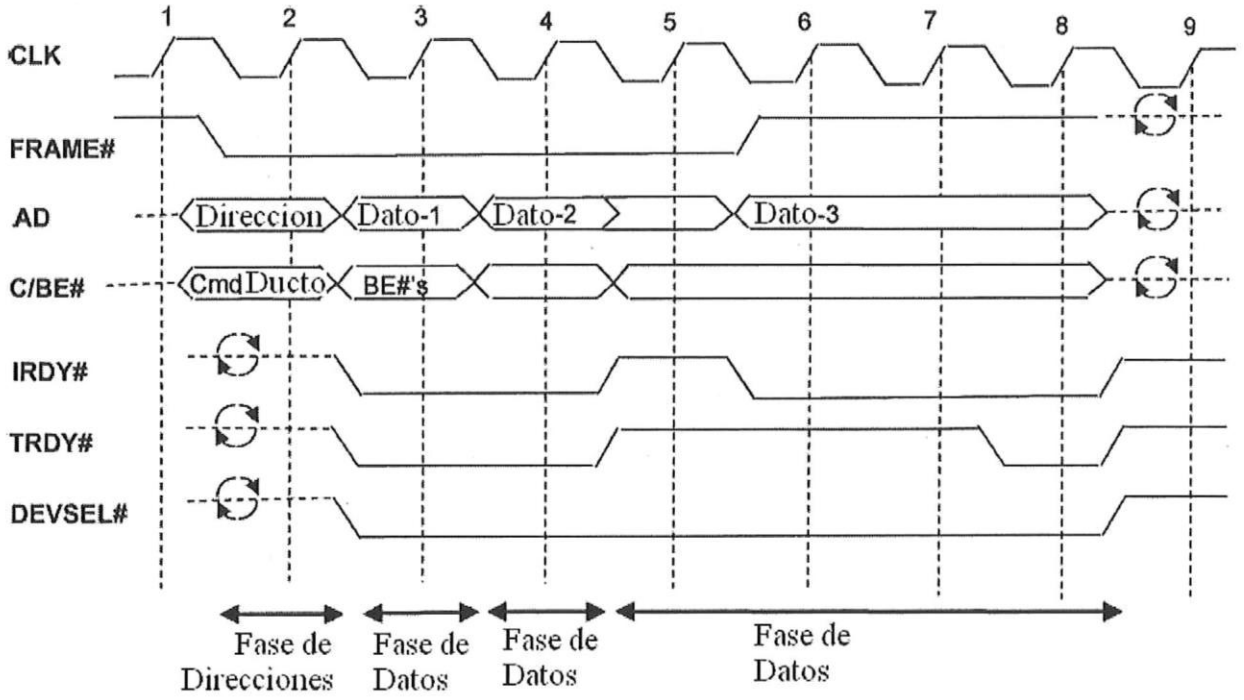


Figura 6. Diagrama de tiempo para una escritura sobre el ducto PCI.

La figura 6 muestra los detalles de una transacción de escritura típica donde los datos son movidos del master al destinatario. La principal diferencia entre una transacción de escritura y la de lectura detallada anteriormente, es que la escritura no requiere del ciclo recursivo entre la fase de direcciones y la de datos ya que el mismo agente está alimentando el bus AD en ambas fases. Debido a la similitud con la operación de lectura, se omitirá la descripción detallada.

4.2.3.3 Habilitación de bytes y su uso

Durante las fases de datos de una transacción, las señales C/BE# indican que líneas de bytes llevan información significativa. El master puede cambiar habilitadores de bytes entre las fases de datos pero deben ser válidos en el ciclo en el que comienza cada fase de datos y

permanecer validos durante la fase de datos entera. El master es libre de utilizar cualquier combinación de habilitadores de bytes continua o discontinua, incluido el caso en que ningún byte sea válido.

Independientemente de los habilitadores de bytes, el agente que maneje al conjunto AD está obligado a llevar las 32 líneas a un valor estable. Esto es para asegurar la generación de paridad y chequeo y para evitar que las líneas AD floten en su estado de alta impedancia.

4.2.3.4 Temporización de DEVSEL#

El destinatario seleccionado está obligado a reclamar la transacción al activar DEVSEL# dentro de los tres ciclos posteriores a la activación de FRAME# como se muestra en la figura 6. Esto lleva a tener 3 categorías para los dispositivos destinatarios basado en su tiempo de respuesta a FRAME#. Estos se clasifican en rápidos, medios y lentos, y su respuesta es en uno, dos y tres ciclos respectivamente. La temporización de un destinatario está codificada dentro del registro de configuración de estado. El destinatario debe activar DEVSEL# antes de que pueda activar TRDY# (o AD en una transacción de lectura).

Si DEVSEL# no es activada después de 4 ciclos después de la activación de FRAME#, el iniciador termina la transacción con un Master Abort. Esto significa que el iniciador trato de acceder una dirección que no existe en el sistema.

4.3 Controlador de ducto PCI

El controlador seleccionado para llevar a cabo la función de puente entre el ducto PCI y la aplicación sobre la que se ha trabajado, es el s5935 de la compañía AMCC. Este poderoso y

flexible controlador para el ducto PCI da soporte a varios niveles de sofisticación en la interface: en el nivel más bajo, puede servir como un destinatario PCI con requerimientos modestos y; en aplicaciones que requieren un alto desempeño, el s5935 puede fungir como el master del bus y alcanzar las capacidades máximas de transferencia de 132 MB/s con un ducto PCI de 32 bits. La figura 7 muestra los principales elementos funcionales que conforman al controlador.

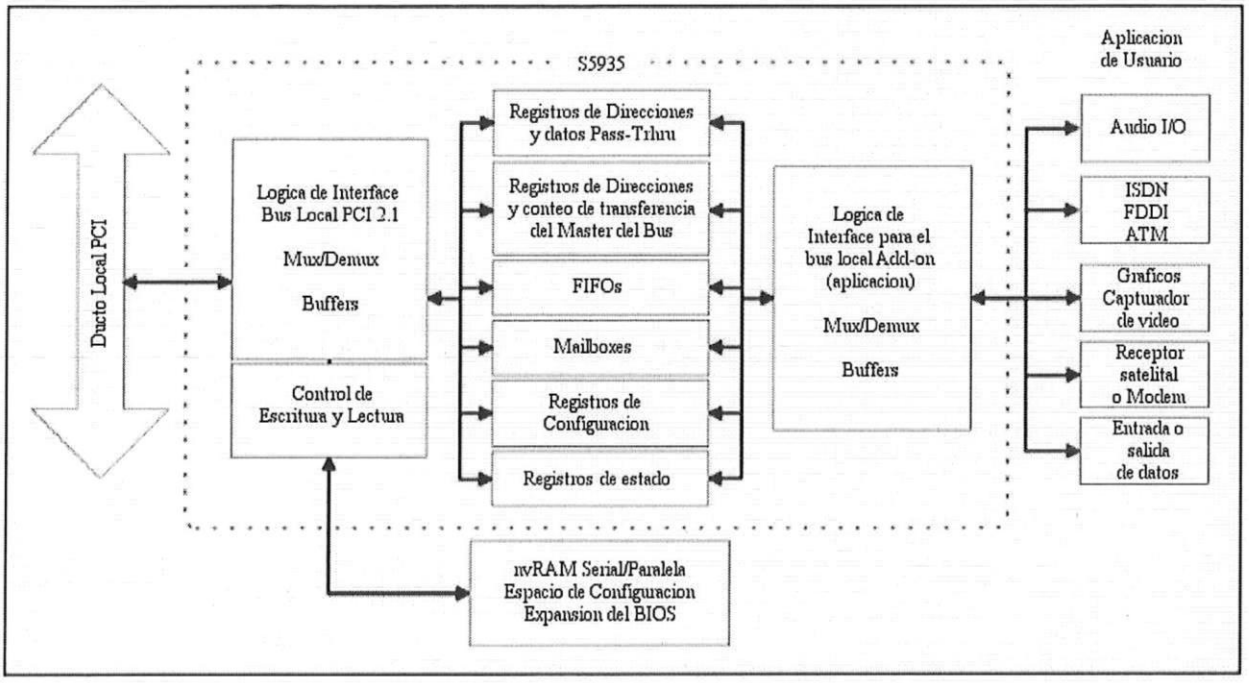


Figura 7.Diagrama a bloques del controlador de ducto PCI S5935.

El S5935 provee 3 interfaces físicas a ductos: el ducto local PCI, el ducto local para el usuario (Add-On) y el ducto para memoria no volátil (serial y paralelo de 8 bits). El movimiento de datos entre los ductos se hace por medio de registros Mailbox o el canal de datos de la FIFO.

Las terminales del S5935 son mostrados en la figura 8. Las señales del ducto local PCI son detalladas en el lado izquierdo, las señales del ducto para el usuario y la interfaz a memorias no volátiles son detalladas en la parte derecha.

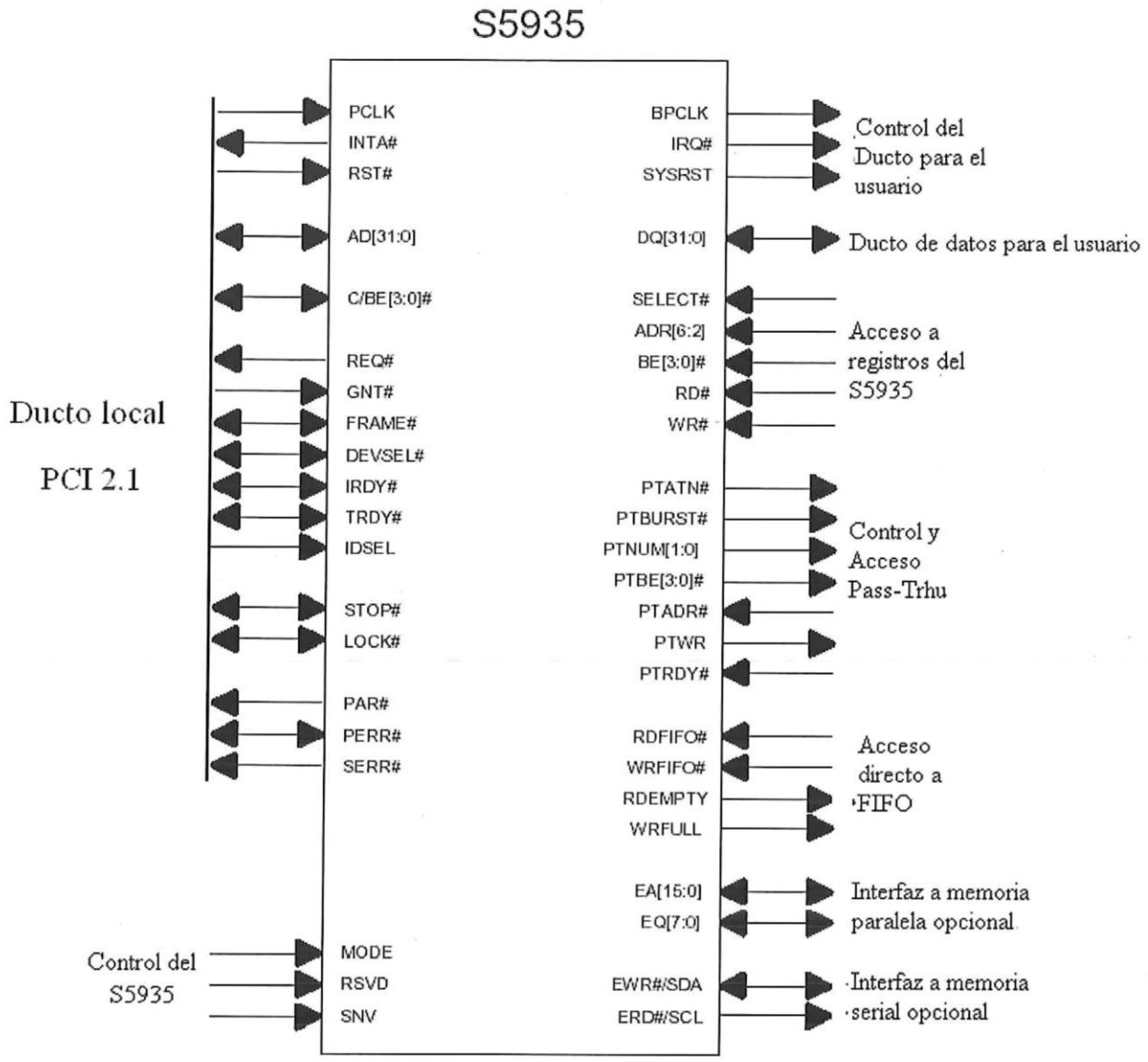


Figura 8. Diagrama de terminales electrónicas del S5935.

El ducto para la memoria no volátil provee un medio para leer datos de configuración de la tarjeta, los cuales son cargados y dados a conocer al anfitrión del ducto local PCI durante el proceso de arranque del sistema.

El controlador hace uso de tres grupos de registros para su configuración y uso. El primer grupo es el que pertenece al espacio de configuración. La tabla 5 presenta la organización de este espacio de configuración.

Tabla 5. Registros contenidos en el espacio de configuración de un dispositivo PCI.

Byte 3	Byte 2	Byte 1	Byte 0	Dirección
ID del Dispositivo		ID Fabricante		00h
Estado PCI		Comando PCI		04h
Clase de Código			ID Revisión	08h
Auto Prueba	Tipo Encabezado	Temporizado de Latencia	Tamaño de línea del cache	0Ch
Registro de Dirección Base 0				10h
Registro de Dirección Base 1				14h
Registro de Dirección Base 2				18h
Registro de Dirección Base 3				1Ch
Registro de Dirección Base 4				20h
Reservado				24h
Espacio Reservado				28h
Espacio Reservado				2Ch
Dirección Base de la ROM de expansión				30h
Espacio Reservado				34h
Espacio Reservado				38h
Latencia Máxima	Concesión mínima	Pin de interrupción	Línea de interrupción	3Ch

Todas las tarjetas PCI tienen un espacio de 256 bytes donde se guarda información acerca de sus capacidades y características, datos acerca del fabricante, versión, etcétera. Esta información se almacena en los primeros 64 bytes de este espacio, los cuales han sido mostrados en la tabla de registros de configuración (tabla 5). Los 196 restantes pueden ser usados para implementar una expansión del BIOS o contener otros datos útiles para la tarjeta.

Tabla 6. Registros de operación del controlador S5935

Registros de operacion PCI	Offset de Direcciones
Registro Mailbox de Salida 1 (OMB1)	00h
Registro Mailbox de Salida 2 (OMB2)	04h
Registro Mailbox de Salida 3 (OMB3)	08h
Registro Mailbox de Salida 4 (OMB4)	0Ch
Registro Mailbox de Entrada 1 (IMB1)	10h
Registro Mailbox de Entrada 2 (IMB2)	14h
Registro Mailbox de Entrada 3 (IMB3)	18h
Registro Mailbox de Entrada 4 (IMB4)	1Ch
Registro del puerto de la FIFO (bidireccional) (FIFO)	20h
Registro de dirección de escritura Master (MWAR)	24h
Registro de conteo de escritura Master (MWTC)	28h
Registro de dirección de lectura Master (MRAR)	2Ch
Registro de conteo de lectura Master (MRTC)	30h
Registro de Mailbox llena/vacía (MBEF)	34h
Registro de control y estado de interrupciones (INTCSR)	38h
Registro de control y estado de Master del ducto (MCSR)	3Ch

El segundo grupo es el de operación PCI y consiste en 16 registros de 32 bits, los cuales, son accesibles al procesador anfitrión del ducto local PCI. Estos son los registros principales a través de los cuales el anfitrión PCI configura la operación del S5935 y la comunicación con el ducto local para el usuario. Estos registros contienen los Mailboxs de entrada y salida, el canal de datos de la FIFO, la dirección del Master y registros de conteo, registros del canal de datos Pass-Thru registros de control y estado del S5935. En la tabla de registros de operación (tabla 6) PCI se hace un listado de estos y sus direcciones.

Tabla 7. Registros de operación Add-on del S5935.

Registros de operación Add-On	Offset de Direcciones
Registro Mailbox de Entrada 1 (OMB1)	00h
Registro Mailbox de Entrada 2 (OMB2)	04h
Registro Mailbox de Entrada 3 (OMB3)	08h
Registro Mailbox de Entrada 4 (OMB4)	0Ch
Registro Mailbox de Salida 1 (IMB1)	10h
Registro Mailbox de Salida 2 (IMB2)	14h
Registro Mailbox de Salida a 3 (IMB3)	18h
Registro Mailbox de Salida 4 (IMB4)	1Ch
Puerto de la F IFO (FIFO)	20h
Registro de dirección de escritura Master (MWAR)	24h
Registro de dirección Pass-Thru (APTA)	28h
Registro de datos Pass-Thru (APTD)	2Ch
Registro de dirección de lectura Master (MRAR)	30h
Registro de Mailbox llena/vacía (AMBEF)	34h
Registro de control y estado de interrupciones (AINT)	38h
Registro de control y estado General (ARCR)	3Ch
Registro de conteo de escritura Master (MRTC)	58h
Registro de conteo de lectura Master (MRTC)	5Ch

El último grupo lo conforman los registros de operación del ducto para el usuario. Como referencia se incluye en tabla de registros Add-on (tabla 7). Este grupo de 18 registros de 32 bits son accesibles al ducto local para el usuario. Estos registros incluyen los Mailbox del ducto para el usuario, su FIFO, sus registros Pass-Thru, registros de estado y control.

4.3.1 El bus para el usuario (Add-On Bus)

Este ducto proporciona un medio para controlar la operación del S5935. La interfaz Add-On es muy similar a los sistemas basados en microprocesador: un ducto de datos de 32 bits, líneas independientes de lectura y escritura, así como otras señales de control utilizadas. El control del S5935 se realiza a través de los Registros de Operación Add-On.

Estos registros son accesados desde la aplicación desarrollada por el usuario e incluyen medios de acceso a la FIFO, operaciones Pass-Thru, los Mailbox y la memoria no volátil. También ofrecen medios de control y de información de estados. Pass-Thru y FIFO, que constituyen los métodos principales de transferencia de información proporcionados por el S5935.

4.3.2 Pass-Thru

Es un medio para realizar directamente operaciones PCI o acceder recursos en tiempo real sobre el ducto local del usuario. Se pueden definir hasta 4 regiones Pass-Thru, ya sean como memoria RAM sobre la tarjeta, o bien como puertos de entrada y salida de datos. Estas regiones son definidas dentro del espacio de configuración almacenado en la memoria no volátil y el ducto local asignará sus direcciones físicas durante el proceso de inicialización. Las direcciones de estas regiones siempre tendrán el mismo offset desde la dirección base de la tarjeta. Un aspecto importante de este método de acceso al ducto local del usuario, es la posibilidad de enviar direcciones para seleccionar alguna localidad de memoria o puerto en especial. Por ejemplo, se puede asignar una región de 8 puertos cuya dirección seria Dirección base de la tarjeta + Dirección de inicio de Región + Dirección de uno de los puertos. Es posible definir el ancho de la unidad de datos que se manejará, es decir, si se utilizarán 8, 16 o 32 bits

para describir un solo dato. Esto influye a su vez en la separación en direcciones de una localidad de memoria u otra. Por ejemplo, si se define una región de memoria con localidades de 16 bits, no es correcto escribir o leer a una localidad 0x01, puesto que no es una dirección par.

4.3.2.1 Señales de control/estado Pass-Thru

Además, el S5935 provee líneas dedicadas que indican el estado del Pass-Thru, es decir, si se realiza una transferencia, es posible leer líneas físicas que indican su ocurrencia y su tipo (lectura, escritura, en bloque o sencilla). Estas señales son mostradas en la tabla 8.

Tabla 8. Señales de control y estado Pass-Thru.

PTATN#	Esta señal de salida indica que una operación Pass-Thru esta ocurriendo.
PTBURST#	Esta señal de salida indica si es un acceso PCI en bloque.
PTNUM [1:0]	Estas señales de salida indican que región Pass-Thru se ha decodificado.
PTBE# [3:0]	Estas señales de salida que bytes contiene datos validos (escritura) o son solicitados (lectura)
PTWR	Esta salida indica si la operación es de lectura o escritura
PTADR#	Esta señal de entrada sirve para extraer la dirección Pass-Thru directamente.
PTRDY#	Esta señal de entrada sirve para indicar que la transacción se ha realizado.
BPCLK	Es una versión reforzada del reloj PCI. Se utiliza para sincronizar operaciones con las aplicaciones externas.

Dentro de este grupo de señales se inlcuyen medios para responder ante situaciones de escritura o lectura, tal es el caso de la línea PTADR#, la cual al ser activada, internamente se selecciona el registro de dirección Pass-Thru y se extrae al bus de datos Add-On la dirección destino de la transferencia.

4.3.3 FIFO

La FIFO integrada es quizá el método más sencillo de transferencia de datos, ya que solo hay que escribir o leer a la dirección de ésta para realizar la transferencia. Esta FIFO tiene una profundidad de 8 localidades. Se cuenta con un registro de estados de la FIFO, en el cual se indica la presencia o ausencia de datos en la misma. Desde el ducto del usuario se cuenta con líneas dedicadas que indican estos estados, lo cual simplifica las transferencias, ya que no es necesario acceder directamente al registro de estados para saber si hay uno o varios datos disponibles para leer, o bien si se ha vaciado y es posible escribir mas datos. Este método permite hacer transferencias en chorro, es decir, que permite la salida de datos con cada ciclo de reloj, alcanzando una velocidad de 132 MBytes/s. A pesar de que esto suena muy atractivo, tiene una desventaja, la imposibilidad de envío de direcciones a través de este canal.

4.3.3.1 Señales de Control/Estado de la FIFO

Así como el método Pass-Thru ofrece señales de control y estado en el Bus Add-On, la FIFO también tiene sus propias señales con las cuales el controlador se comunica hacia la aplicación exterior. Estas señales se describen en la tabla 9.

Tabla 9. Señales de Control/Estado de la FIFO

RDEMPTY	Indica que la FIFO PCI al Add-On esta vacía.
WRFULL	Indica que la FIFO Add-On al PCI esta llena.
FRF	Indica que la FIFO PCI al Add-On esta llena1.
FWE	Indica que la FIFO Add-On al PCI esta vacía[3].
RDFIFO#	Lee datos de la FIFO PCI alAdd-on.
WRFIFO#	Escribe datos a la FIFO Add-On al PCI.
FRC#	Limpia los apuntadores y los indicadores de estado de la FIFO PCI al Add-on[1].
FWC#	Limpia los apuntadores y los indicadores de estado de la FIFO Add-On al PCI[1].
AMREN	Habilita el control maestro del ducto para lecturas iniciadas por la aplicación externa[1].
AMWEN	Habilita el control maestro del ducto para escrituras iniciadas por la aplicación externa[1].

4.4 Características del Controlador de ducto PCI S5935

El S5935 presenta una serie de características que lo respaldan como un controlador que cumple con los requisitos esenciales para la aplicación que se desarrolla y explica en este documento. Entre ellas se puede hacer mención de las siguientes:

- Compatible con PCI 2.1 como Master y Slave
- Razón de transferencia completa de 132 MBytes por segundo
- Respaldo a los nuevos chipsets 440GX/BX de Intel
- Soporte para WinNT con service pack 2 y 3
- Frecuencia de operación del ducto PCI de CC a 33 MHz
- Bus para el usario de 8/16/32 bits
- Cuatro canales Pass-Thru definibles

[3] Estas señales solo están disponibles cuando se usa la memoria no-volátil y el S5935 se configura para que el Add-On pueda iniciar transferencias como Bus Master.

- Dos FIFO's internas de 32bits con DMA
- Operación sincronía del bus para el usuario
- Registros MailBox con nivel de estado por byte
- Pins de interrupciones para PCI y en el bus del usuario
- Cargado de inicialización opcional en memoria no volátil
- Código de expansión para BIOS/POST opcional

En base a este juego resumido de características, este controlador; además de la aplicación que se presenta, también puede ser usado para desarrollar otros dispositivos, como:

- Redes de alta velocidad
- Aplicaciones de video digital
- Comunicaciones por puertos de entrada / salida
- Adquisición de datos a alta velocidad
- Encriptación / decriptacion de datos
- Entrada / salida de audio

4.4.1 Función del controlador de ducto PCI en la aplicación

El S5935 se usa como un puente entre el ducto PCI local de una PC y un dispositivo exterior diseñado para acomodarse a las necesidades de transferencia de datos a alta velocidad. Su tarea principal es la de llevar a cabo los diálogos y negociaciones con el ducto PCI y de proveer una interfaz para la aplicación externa. Dada la complejidad e importancia de esta función, Luego se dará a conocer como funciona y lleva a cabo el movimiento de información hacia el exterior mediante el bus para el usuario Add-On.

4.5 Dispositivo Lógico Programable Complejo (CPLD)

El dispositivo lógico programable usado en esta aplicación es el CY37128, que pertenece a la familia ULTRA37000 de la compañía Cypress Semiconductor. Este dispositivo está diseñado para llevar la flexibilidad, facilidad de uso y desempeño del 22v10 a CPLD's de alta densidad. La arquitectura de este CPLD está basada en un número de bloques lógicos que están conectados por una Matriz de Interconexión Programable (Programmable Interconnect Matriz, PIM). Cada bloque lógico tiene su propio arreglo de términos de productos, su colocador de términos de productos y 16 macroceldas. La PIM distribuye las señales de salida del bloque lógico y los pins de entrada a las entradas del bloque lógico. Este dispositivo se puede borrar eléctricamente y permite su reprogramación en sistema ISR (In-System Reprogrammable) mediante una interfaz JTAG.

4.5.1 Características principales

- Voltaje de operación de 5v
- 128 macroceldas
- 64 pins de entrada y/o salida
- 5 entradas dedicadas o de reloj
- Capacidad de Bus-Hold
- Slew rate programable(lento/rápido)
- Encapsulado PLCC de 84 pins

4.5.2 Desarrollo de Software y Programación

Para programar este dispositivo se utilizó el sistema de desarrollo WARP[3], que lo proporciona la misma compañía que lo produce. En dicho sistema se incluye un editor de texto para el lenguaje de descripción de hardware VHDL. También provee funciones optimizadas de síntesis y colocación mediante el reemplazo de circuitos genéricos básicos por otros previamente optimizados para el dispositivo destino, por medio de la implementación de lógica en memoria sin usar, y la perfecta comunicación entre la síntesis y la colocación sobre el CPLD.

Para la programación se hizo uso de una computadora personal con el software y cable de programación 37000 UltraISR. Este método de programación consiste en la colocación de un conector que lleva los pins ISR de los dispositivos del sistema. El cable de programación se conecta al puerto paralelo de una PC y a este conector. Un archivo simple de configuración instruye al software acerca de las operaciones de programación a llevarse a cabo en cada uno de los dispositivos ULTRA37000 que integren al sistema. Después, el software ISR realiza todas las manipulaciones de datos necesarias para completar la programación, verificación, etc.

En la figura 9 se muestra un diagrama que resume los últimos dos párrafos.

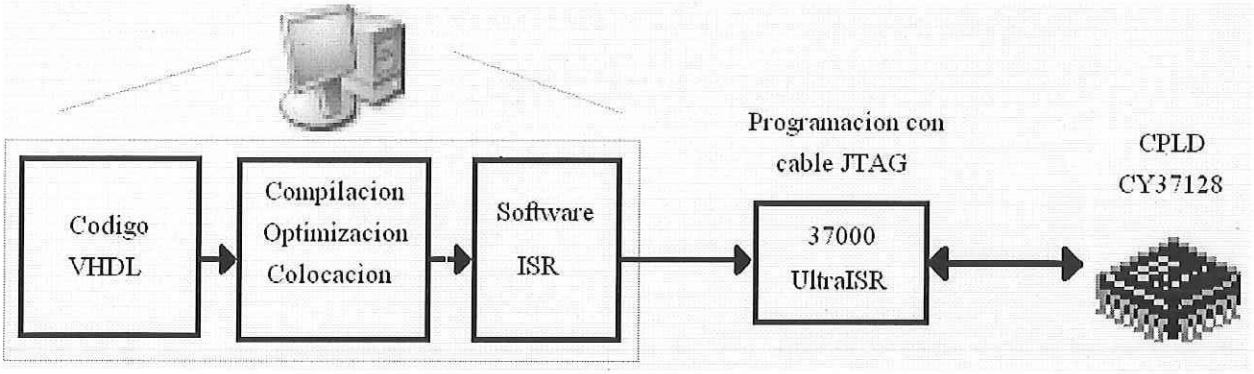


Figura 9. Programación del dispositivo lógico programable complejo

[3] Consulte la página Http://www.cypress.com para más información.

4.5.3 Función del CPLD CY37128 en la aplicación

El CY37128 funciona como un puente entre el bus del usuario proporcionado por el controlador de ducto PCI y el ducto tipo IDE hacia el exterior. Por el lado del S5935, realiza el protocolo necesario para llevar a cabo lecturas y escrituras a través del ducto para el usuario Add-On, el cual es parecido al protocolo del ducto PCI. Dentro del esquema manejado en esta aplicación, la iniciativa en transferencias, ya sea de escritura como lectura, es controlada desde el software de la PC, por lo que el trabajo a realizarse por el CPLD es totalmente en calidad de esclavo; es decir, el CPLD deberá responder a las ordenes que sean recibidas por el ducto del usuario Add-On.

4.6 Interconexión con la aplicación exterior

Debido a que se desarrolla una tarjeta para transferencias de datos de una manera general, es preciso que la tarjeta pueda proveer medios de interconexión con aplicaciones externas.

4.6.1 Descripción

La conexión al exterior se llevará a cabo mediante 22 señales, de las cuales 16 están dedicadas exclusivamente D[15::0] para datos, 3 bits para direcciones A[2::0] y 3 líneas de control WR#, RD# y CS#. Es requisito para las aplicaciones externas que se conecten a esta tarjeta contar con este juego de líneas, así como tener la capacidad de responder a tiempo a las peticiones hechas desde la PC. Es muy importante tener en cuenta que las líneas de control son salidas únicamente, por lo que las lecturas y escrituras así como sus tiempos dependen solo del sistema residente y no de los periféricos.

5 Integración y caracterización

5.1 Funcionamiento general

En un intento por lograr una rápida y clara comprensión en lo que respecta a la integración de los componentes que conforman nuestro sistema, se expone el siguiente diagrama (figura 10) que describe de manera gráfica y general el trabajo propuesto.

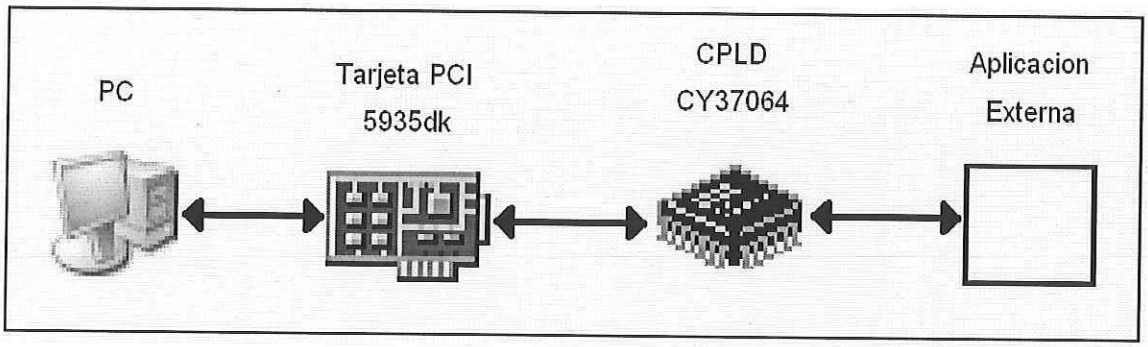


Figura 10. Diagrama general del diseño de la tarjeta de transferencia de datos.

En su descripción más básica, el sistema desarrollado consta de 4 componentes principales: una computadora personal, una tarjeta de desarrollo PCI que a su vez está conectada a un circuito impreso que contiene el CPLD CY37128. Este último es el encargado de proveer la interfaz electrónica para la aplicación externa. No se darán a conocer más detalles acerca de la caja negra que representa la aplicación externa, ya que como se ha mencionado, si bien nuestro sistema de adquisición de datos está enfocado a CCD's, no se limita a estos. Cabe señalar, que aunque no se dan especificaciones concretas acerca de la aplicación externa, esta debe tener la capacidad de responder a tiempo a las peticiones que le haga el CPLD, ya sean de escritura o lectura.

Como primer componente, se encuentra la computadora personal. El único requisito obligatorio para este componente es contar con una ranura PCI libre dentro de la placa madre. Durante el desarrollo del sistema se han utilizado dos computadoras, esto con el fin de hacer comparaciones entre el desempeño de la tarjeta en diferentes sistemas. La primer computadora (PC_1)es una PC con procesador Celeron 600MHz, 64 MB RAM. La segunda computadora (PC_2) es una PC con procesador Pentium IV 3.2 GHz y 1 GB RAM. Ambas computadoras cuentan con un sistema operativo Linux Slackware 10, con un Kernel 2.4. En la sección de pruebas y resultados del presente documento, se presentaran gráficas de los resultados obtenidos y se hará una comparación entre los propios de las computadoras anfitrión.

5.2 Kit de desarrollo para el controlador S5935

El kit de desarrollo consta de cuatro circuitos impresos: una tarjeta que incluye el controlador de ducto S5935, dos tarjetas piggyback con áreas de desarrollo tipo tarjeta perforada, auxiliares en el desarrollo de hardware y una tarjeta ISA para la observación de la tarjeta con el controlador de ducto PCI.

La tarjeta principal, que incluye al controlador, posee dos conectores para memorias no volátiles: una FLASH paralela 29C512 (64K x 8) y una EEPROM serial 24C16 (2K x8). Se puede hacer uso de solo una memoria de estas para suplir el espacio de configuración por omisión de la tarjeta PCI con uno específico para una aplicación dada. Además estas memorias pueden contener código que funja como una expansión ROM del BIOS. La tarjeta cuenta además con dos conectores para PLD's 22v10, jumpers para fijar las líneas PRSNT# [2::1], y varios conectores que llevan las líneas de datos y control utilizadas en el bus para el usuario. En el

apéndice X se detalla la organización y distribución de estas líneas entre los diversos conectores.

Dentro del kit de desarrollo también se incluyen 2 programas como herramientas para el desarrollo de aplicaciones con esta tarjeta. El primero es un visualizador-editor de registros usado para diagnóstico y búsqueda de fallas, llamado AMCCDIAG.exe. La segunda herramienta, AMCCPCI3.EXE, es un editor-visualizador-programador para la memoria no volátil. Ambos programas fueron creados para ser utilizados en un sistema operativo DOS.

Como entre los requisitos figura la necesidad de utilizar direcciones para la aplicación externa, fue necesario realizar las pruebas de desarrollo haciendo uso del método Pass-Thru. Para ello, fue necesario programar la memoria no volátil de la tarjeta PCI de desarrollo. A continuación una breve descripción acerca de la función de esta memoria en nuestra aplicación y como se configura esta para nuestra aplicación.

5.2.1 Función y programación de la memoria no volátil del controlador de ducto PCI

La memoria no volátil funciona como una máscara para el espacio de configuración de la tarjeta PCI (capítulo 4.3). El archivo que contiene la información acerca de los requerimientos de la tarjeta hacia su sistema anfitrión, ha sido creado utilizando la herramienta de AMCCPCI3.EXE y almacenado en archivo HEX. Posteriormente, estos fueron programados en la memoria no volátil utilizando un programador universal GALEP-4[4]. El proceso entero de edición del contenido de la memoria y su programación, fueron realizados en una tercera computadora: una computadora personal tipo portátil (laptop), con procesador Pentium a 100Mhz, 15 Mbytes en RAM, 1 GB de disco duro y sistema operativo Windows 95. La principal razón de haberlo hecho de esa manera, es la falta de una plataforma donde ejecutar la aplicación

[4] Para mayor información, puede consultar el sitio en Internet http://www.galep.com

de programación original, puesto que requiere un sistema operativo DOS con accesos a hardware de bajo nivel.

En la memoria no volátil de la tarjeta de desarrollo PCI, se han definido regiones de puertos de entrada y salida de datos, que son llamados por el libro de especificaciones del S5935 como regiones Pass-Thru. Estas regiones también pueden ser declaradas como regiones de memoria pero en nuestro caso se tratarán como puertos. Para fines de pruebas, se estableció una región de puertos de 32 bytes, con un ancho de palabra de 32 bits. Esta región puede dividirse en 8 registros de 4 bytes con que se cumple uno de los requisitos para la conexión con la aplicación externa al solo ser necesarios 3 bits para identificar estos registros. Esta región fue definida de esta manera ya que la mayoría de los registros internos son de 32 bits, y la tarjeta de desarrollo no provee un conector para acceder direcciones que utilicen los dos bits menos significativos, resultando en la imposibilidad de seleccionar registros de 16 bits. Esto no representa un problema, ya que simplemente se descartan los dos bytes más significativos del registro de datos. Durante el proceso de inicio del sistema, la tarjeta PCI recibe una dirección base asignada por el anfitrión, dentro del mismo proceso, las regiones de puertos definidas y solicitadas por el S5935 reciben también una dirección dentro del sistema. Es entonces cuando nuestra tarjeta queda lista para realizar operaciones ante el sistema que lo hospeda.

5.2.2 Integración detallada entre el CPLD y kit de desarrollo para el controlador de ducto PCI S5935

En el diagrama siguiente (figura 11), se muestra como se encuentran interconectados el controlador PCI y el CPLD.

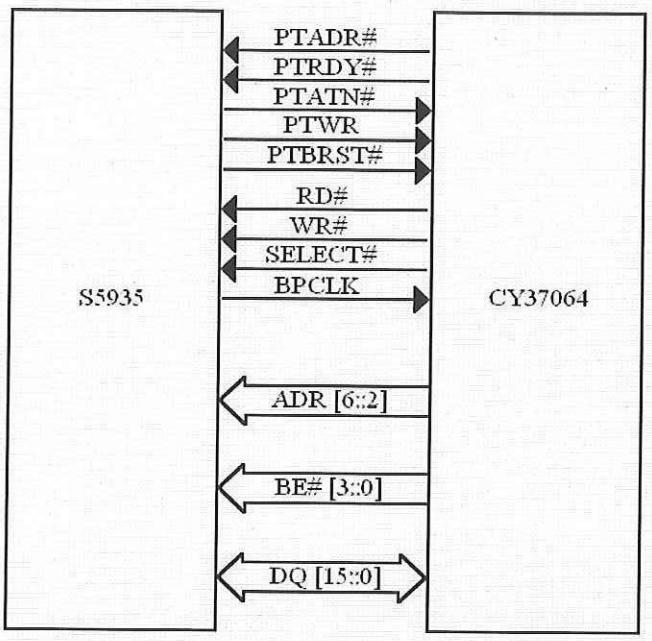


Figura 11. Interconexión entre S5935 y CY37128.

El sistema desarrollado presenta dos medios para la entrada y salida de datos y direcciones, o bien comandos de control y chequeo de estados; esto dependerá de la aplicación o instancia en específico del sistema. Estos medios son el Pass-Thru y FIFO. A continuación se detallara en que consiste cada uno y la manera de utilizarse.

La conexión física entre estos dos dispositivos se realiza mediante cables planos que conectan las placas impresas del kit de desarrollo con la del CPLD.

5.3 Entrada y salida de datos mediante Pass-Thru

Para llevar a cabo una transferencia de datos a un puerto definido dentro de una región Pass-Thru, el controlador identifica la dirección del puerto como una dentro de su rango y la almacena en el registro de direcciones Pass-Thru. Al mismo tiempo, identifica el tipo de

transferencia y notifica esto al exterior mediante las líneas de estado Pass-Thru correspondientes. Estas líneas de estado y control son monitoreadas por el CPLD para actuar conforme a la operación llevada a cabo.

A continuación se profundizará en el tema, detallando cada uno de los pasos para realizar una transferencia de información haciendo uso de este medio.

5.3.1 Escritura

En caso de tratarse de una operación de escritura a un puerto, será necesario ejecutar una función de salida de datos a un puerto desde el software que controle la tarjeta (función outw() para el caso del compilador de C). El sistema operativo en conjunto con el anfitrión PCI se encargará de notificar el intento de acceso a un puerto registrado por el S5935. De ser posible, es decir, en caso que la tarjeta pueda aceptar la transferencia (sin datos previos) el controlador inmediatamente captura la dirección y el dato en los registros Pass-Thru. Inmediatamente después son activadas las líneas de estado/control en el ducto del usuario y se espera una respuesta por parte del CPLD para completar la transferencia. En el siguiente diagrama se expone el protocolo que deben llevar a cabo S5935 y CY37128 para llevar a cabo esta operación. Después se detallará que ocurre en cada ciclo y como interactúan los dispositivos involucrados.

Ciclo 0	La información del ciclo de direcciones PCI es decodificada y almacenada en el registro de direcciones Pass-Thru del S5935.
Ciclo 1	La dirección PCI ha sido identificada como un acceso a la región Pass-Thru 0. EL dato en el bus PCI es almacenado en el registro de datos Pass-Thru. Se activa la señal PTATN# para indicar que un acceso Pass-Thru esta ocurriendo.
Ciclo 2	Las señales de estado Pass-Thru son activadas e indican a la lógica Add-On que acción debe ejecutar. Estas señales solo son validas mientras

PTATN# esta activada.
PTNUM[1:0] : El acceso es a la región Pass-Thru 0.
PTWR: Activada. Se trata de una operación de escritura
PTBE#[3:0]: Todos en cero. Es una operación de 32 bits.
La señal de entrada PTADR# es activada por el CPLD para extraer la dirección del registro de direcciones Pass-Thru y es colocada inmediatamente en el bus de datos Add-On. También las señales BE# [3:0], SELECT# y las direcciones son modificadas por el CPLD.

Ciclo 3 — Las señales de entrada SELECT#, BE#[3:0] y direcciones permanecen validas durante este ciclo. RD# debe ser activada para colocar el dato sobre el bus de datos Add-On

Ciclo 4 — Si PTRDY# es activada antes del flanco de subida de este ciclo, inmediatamente PTATN# es desactivada y el ciclo se habrá completado.

Ciclo 5 — Si la lógica requiere mas tiempo para leer el registro de datos Pass-Thru, el CPLD puede retrasar la activación de PTRDY#.

Ciclo 6 — PTATN# y PTBURST# están desactivadas y el ciclo de escritura ha sido terminado.

La figura 12 presenta un diagrama de tiempo de las operaciones descritas.

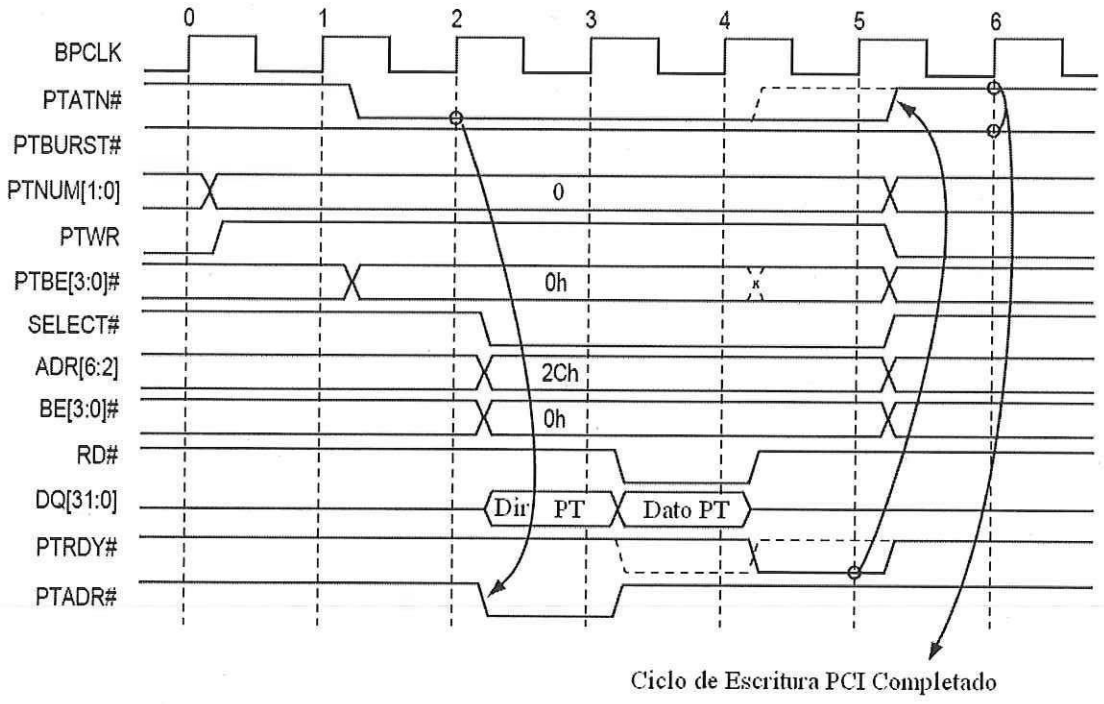


Figura 12. Diagrama de tiempo para una escritura mediante Pass-Thru.

5.3.2 Lectura

Al tratarse de una operación de lectura a un puerto dentro de una región Pass-Thru, debe hacerse un llamado a una función de entrada de datos a través de un puerto físico (inw() en el case del lenguaje C en Linux) en la dirección deseada dentro del rango disponible o deseado. El sistema operativo pasa esta información al anfitrión PCI y éste a su vez hace llegar la petición al S5935. Cuando el controlador S5935 puede atender la petición, la dirección es inmediatamente almacenada en el registro de direcciones y en esta ocasión, dada la naturaleza de la operación, el registro de datos deberá ser llenado por un agente externo que se encuentre en el puerto que es accedido. De manera similar a la operación de escritura, las líneas de control/estado Pass-Thru son activadas para comenzar el protocolo. En la figura 13, se muestra un diagrama de tiempo que describe de manera gráfica el protocolo llevarse a cabo para esta operación. Más delante se explicará detalladamente que sucede en cada ciclo de reloj.

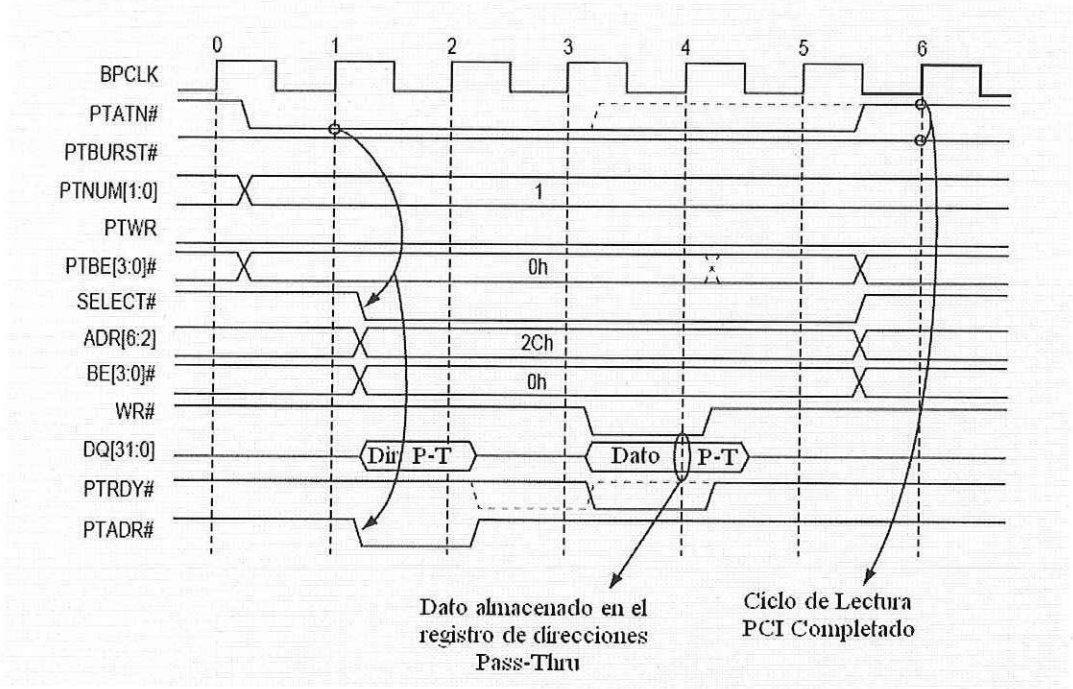


Figura 13. Diagrama de tiempo para lectura mediante Pass-thru.

Ciclo 0 La información de direcciones PCI es almacenada por el S5935 y es reconocido como un acceso a la region Pass-Thru 1. PTATN# es activada .

Ciclo 1 Las señales de estado Pass-Thru son activadas.

PTBURST#	Desactivada. El acceso es una transferencia sencilla.
PTNUM[1:0]	El acceso es a la región Pass-Thru 1
PTWR	Desactivada. Se trata de una operación de escritura
BE#[3:0]	Todos en cero. Es una operación de 32 bits.

La señal de entrada PTADR# es activada por el CPLD para extraer la dirección Pass-Thru y es colocada en el bus de datos Add-On.

Ciclo 2 Este ciclo de reloj es necesario para evitar contencion en el ducto DQ (datos y direcciones Add-On). Un poco de tiempo debe esperarse después de que PTADR# sea desactivada para que las salidas DQ floten antes de que el CPLD intente escribir sobre el registro de datos Pass-Thru.

Ciclo 3 SELECT#, BE#[3:0] y las direcciones de entrada permanecen validas Si WR# es activada durante el flanco de subida del ciclo 3, el dato en el bus DQ sera capturado.

Ciclo 4 Si PTRDY# es activada durante el flanco de subida de este ciclo, la operación se habrá completado.

Ciclo 5 PTATN# y PTBURST# están desactivadas y el ciclo de escritura ha sido terminado.

5.4 Transferencia sencilla de Datos utilizando la FIFO

La FIFO es una forma sencilla de realizar operaciones de lectura y escritura sobre los periféricos conectados a la tarjeta PCI, ya sean transferencias sencillas o en chorro. EL controlador S5935 cuenta con dos FIFOs internas: una para leer (Add-on to PCI FIFO) y una para escribir (PCI to Add-on FIFO). Cualquiera de estas FIFOS cuenta con una profundidad de 8 DWORD's. La FIFO es controlada y supervisada desde un registro de estado y control en el S5935, mientras que los datos fluyen a través de un solo registro (Direccion_base+0x20). Dependiendo del tipo de operación (lectura o escritura), la FIFO se selecciona automáticamente, es decir, una operación de lectura siempre accederá la Add-on -PCI FIFO, mientras que a una escritura se le concede acceso a la PCI-Add-on FIFO. El registro de control y estado de la FIFO provee bits de estado para diferentes condiciones de la FIFO: PCI to Add-on FIFO llena, Add-on

to PCI FIFO vacía, FIFO medio llena (4 lugares ocupados) etc. Además de este registro, también existen líneas que contienen información relevante a los periféricos que hacen uso de la FIFO. En este caso, el CPLD utilizado checa estas líneas para responder de manera oportuna a las operaciones requeridas.

Desde el punto de vista del sistema operativo, esta dirección es un puerto, que se encuentra en un offset de 0x20 de la dirección base de la tarjeta del S5935, por lo tanto, el acceso a la FIFO se hace mediante el uso de las funciones de entrada y salida de datos por un puerto físico (inw() o outw()).

Desde el CPLD, existen dos maneras de completar una transferencia de lectura o escritura desde/hacia la FIFO. Se puede realizar un procedimiento similar al descrito en la operación Pass-Thru: primero se selecciona el registro de la FIFO con las líneas de dirección, se habilitan bytes, el CPLD se encargara de manejar las lineas RD# o WR# según sea el caso. O bien, se hace uso de las líneas especificas RDFIFO# o WRFIFO#, las cuales controlan de manera interna la dirección, los bytes por habilitar y las líneas RD# y WR#, gracias a esto, solo es necesario activar una sola línea para realizar la operación requerida y así llenar o vaciar la FIFO según convenga.

5.4.1 Transferencias en chorro utilizando la FIFO

Es posible realizar transferencias a chorro con el controlador de ducto PCI s5935. Durante el desarrollo de esta investigación se exploró esta posibilidad haciendo uso de un mecanismo provisto por el controlador utilizado: el canal FIFO.

Como ya se explicó con anterioridad, una transferencia a chorro consta de una fase de dirección y múltiples fases de datos (a diferencia de una transacción sencilla, que cuenta con una

fase de dirección por cada fase de datos), de manera que después del primer dato escrito en el ducto, un nuevo dato deberá colocarse en el ducto cada ciclo de reloj posterior hasta que todos los datos hayan sido transferidos o exista alguna condición que impida el flujo y la transacción se vea interrumpida. Para que una transacción en chorro pueda llevarse a cabo, hace falta definir dos elementos importantes: una dirección física (destino) en el ducto para el primer dato y el número de datos a transferirse. Los dispositivos destino deben contar con mecanismos para transferir los nuevos datos a su lugar correspondiente utilizando, por ejemplo, un contador que además funcione como registro de direcciones que se incremente con cada ciclo de reloj.

La FIFO cuenta con un registro de Estado y Control, sobre el cual es posible controlar la manera en que los datos fluyen a través de la FIFO. Además de este registro, existen cuatro registros que adquieren importancia si se desea realizar transferencias a chorro: dos registros de direcciones base para lecturas o escrituras y dos registros contadores para establecer el número de datos a transferirse. Estos registros se actualizan con cada ciclo de reloj para llevar la cuenta del progreso de la transacción si ésta se ve interrumpida. Todos estos registros son accesados mediante una instrucción de lectura o escritura a puertos a la dirección base de la tarjeta más el offset en direcciones correspondiente al registro que se desea acceder.

Para realizar una transferencia en chorro utilizando la FIFO, los pasos siguientes deben llevarse a cabo desde el software que controle a la tarjeta PCI:

1.- Establecer el tipo de manejo de la FIFO: Si las transacciones serán iniciadas desde el ducto PCI (software) o desde la aplicación externa (hardware externo).

2.- Habilitar las interrupciones necesarias del S5935.

3.- Restablecer las banderas del estado de la FIFO. Esto no es necesario, pero es muy recomendable para asegurar una transacción sin problemas.

4.- Definir la prioridad de lecturas contra escrituras. Esto se hace alterando los bits

correspondientes en el registro de Control y estado de la FIFO.

5.- Definir la dirección fuente/destino para la transacción.

6.- Definir el número de bytes a transferirse en el registro correspondiente, ya que existen un registro contador para lecturas y otro para escrituras.

7.- Habilitar el Bus Mastering. Una vez que éste se habilite, la transacción comenzara y terminara hasta que los registros de conteo se encuentren en ceros. Si la operación se ve interrumpida, ya sea porque la FIFO se ha llenado o exista una condición que en el ducto que impida continuar, el controlador intentara continuar cuando dicha condición haya cesado.

La figura 14 contiene un diagrama que ayudará a visualizar los pasos anteriores.

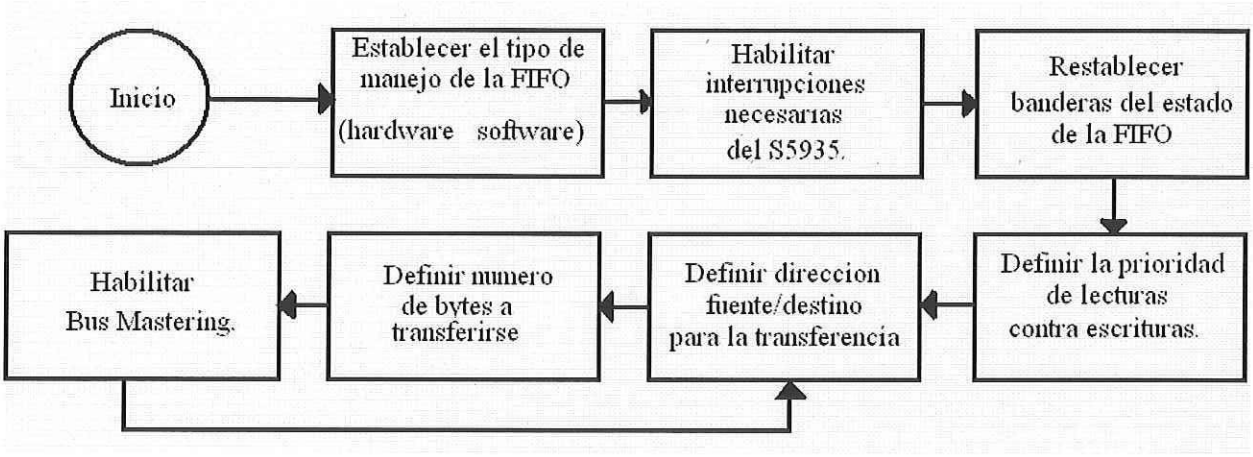


Figura 14. Diagrama ilustrativo de los pasos a seguir para realizar una transferencia a chorro.

El orden de estos pasos no es muy importante, pero se requiere que al menos los últimos 3 sean llevados a cabo. Es posible habilitar el Bus Mastering y después escribir un valor diferente de cero en cualquiera de los contadores para comenzar con la transacción. La transacción tendrá lugar mientras Bus Mastering este habilitado y los registros de conteo alberguen un valor diferente de cero.

5.5 Desarrollo de la programación asociada

Para la programación del software controlador de la tarjeta desarrollada, se utilizó el lenguaje C en un sistema operativo Slackware Linux. Se utilizó un editor sencillo de texto llamado EMACS, cuando solo era posible trabajar en una consola; y en el editor KEDIT, cuando fue posible tener el ambiente gráfico KDE. El compilador utilizado para nuestros programas es el llamado gcc (GNU Compiler Collection) versión 2.95.3.

El programa escrito durante el desarrollo y pruebas de nuestra tarjeta PCI, cuenta con una serie de rutinas, dispuestas en un menú de opciones; entre las que destacan la escritura y lectura a puertos mediante Pass-Thru, la FIFO, así como opciones para leer y escribir algunos registros de estado y control del S5935.

Para llevar a cabo una escritura a un puerto de entrada y salida de datos, basta con ejecutar una instrucción o comando de escritura o lectura a puertos físicos en el software que controla a la tarjeta. En nuestro caso, dichas funciones se incluyen en la biblioteca que contiene estas y otras funciones similares, y que se encuentra en */asm/io.h.*

La función para leer un puerto físico de 16 bits en lenguaje C, tiene la siguiente estructura:

outw(Dato, Direccion);

outw es una instrucción de escritura de una variable WORD (16 bits) a un puerto. El primer argumento es el dato en sí, y la dirección dependerá del medio seleccionado para realizar la operación. Si se utiliza Pass-Thru, la dirección será una dentro del rango de direcciones de puertos asignados durante el arranque del sistema. Si la FIFO es seleccionada como medio para la transferencia, la dirección tendrá la forma BADR+0x20; donde BADR es la dirección base de la

tarjeta asignada durante el arranque, y el 0x20 es el offset donde se encuentra el registro de la FIFO.

El caso de una operación de lectura, la instrucción utilizada es parecida a la de escritura, tiene la forma:

dato = inw(Direccion);

dato es una variable WORD (16bits), la instrucción inw() es una lectura a puerto, donde el dato a leerse será del tamaño de *DATO*, y la dirección tiene las mismas características que en el caso del comando de escritura.

Estas instrucciones de acceso de bajo nivel a puertos, fueron diseñadas para usarse en espacio de Kernel principalmente, pero pueden ser utilizadas por programas ejecutados en espacio de usuario. En caso de que el software que controla nuestro sistema se ejecute en espacio de usuario, se deberá incluir una de las funciones *ioperm()* o *iopl()* al principio del programa, para otorgar el acceso a puertos.

6 Resultados, mejoras al sistema y conclusiones

6.1 Resultados experimentales

El sistema ha sido probado en 2 computadoras distintas, esto con el fin de conocer la dependencia respecto al sistema anfitrión. La primera computadora (PC1) es una PC con procesador Celeron 600MHz, 64 MB RAM. La segunda computadora (PC2) es una PC con procesador Pentium IV 3.2 GHz y 1 GB RAM. Es importante tener en consideración que en el ducto local PCI de la primera existen solo 3 elementos dados de alta en el sistema, mientras que en la PC2 existen varios componentes que pasan a través de el ducto local PCI, lo cual puede tener repercusiones en el desempeño de la tarjeta si un elemento de ellos trata de utilizar el ducto mientras se ejecutan operaciones con la tarjeta. En ambas computadoras, la frecuencia de reloj del ducto PCI es de 33Mhz, o bien, cada ciclo tiene una duración de 33 nanosegundos.

Haciendo uso de un osciloscopio Tektronix TDS3000 se capturaron formas de onda correspondientes a las líneas PTATN, el CS# y WR# del S5935 con el fin de medir el tiempo que lleva realizar una transacción de escritura a un puerto. La frecuencia de muestreo es de 4ns. Solo se muestran resultados de operaciones de escritura para fines de demostración, dado que las operaciones de lectura no difieren considerablemente en cuanto a su duración.

En ambos casos, la duración total del un ciclo de escritura fue de 146 nanosegundos, alrededor de 5 ciclos de reloj, lo cual era de esperarse.

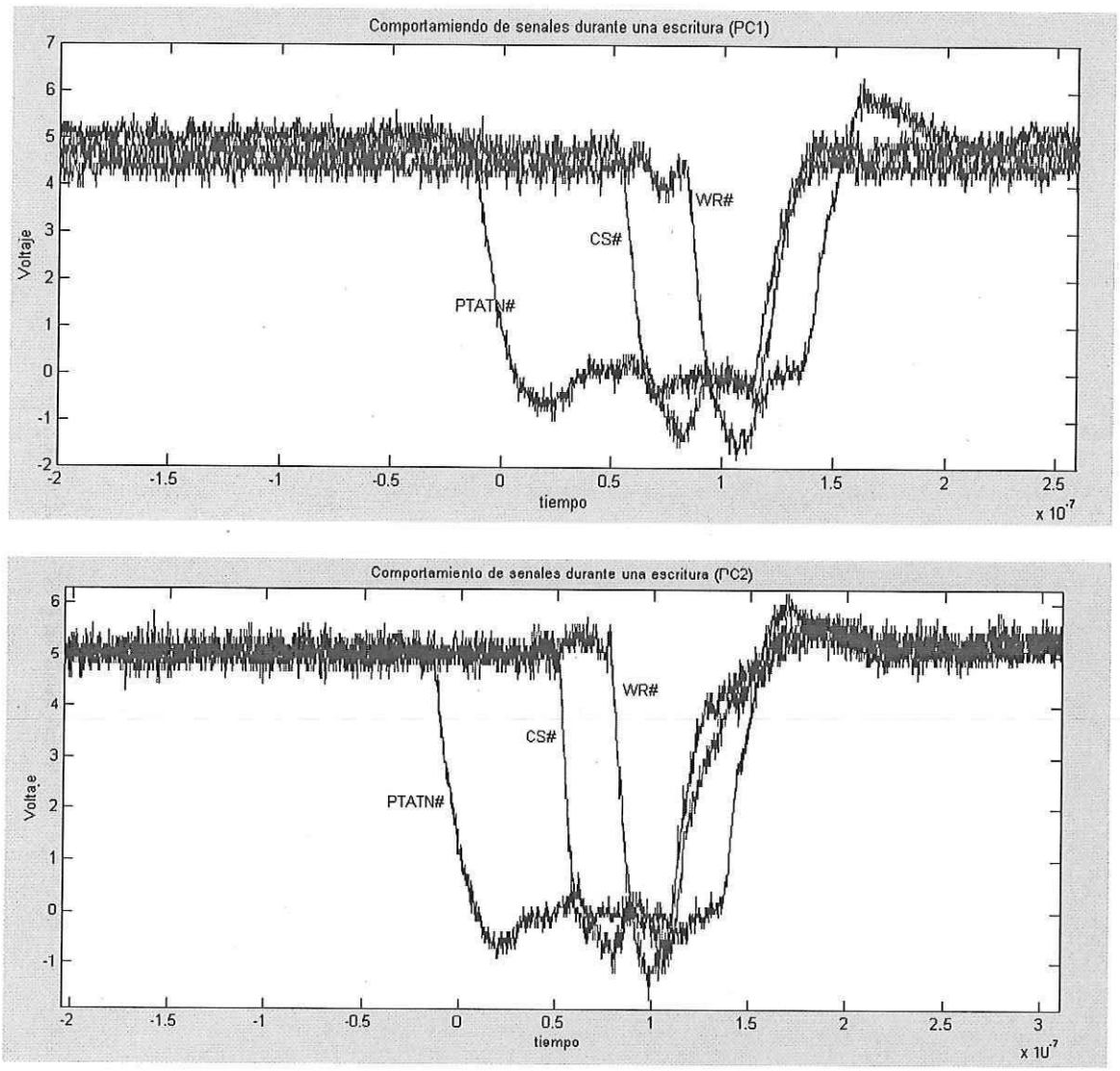


Figura 15. Formas de onda de señales durante operaciones de escritura Pass-thru.

Posteriormente se hicieron pruebas haciendo 32 escrituras consecutivas al mismo puerto, esto con el fin de buscar variaciones en la duración de estos ciclos, así como medir tiempos muertos. Se capturó la forma de onda de la señal PTATN y el análisis se realizó mediante un programa escrito en Matlab. Se determina el ancho en tiempo de los estados lógicos registrados, contando la duración de estos respecto al número de muestras en que se mantiene

dicho estado. Se almacenan los datos en un par de matrices para ser analizados posteriormente por las funciones estadísticas incluidas en la paquetería de Matlab. Se considera tiempo muerto cuando hay un nivel lógico en alto entre dos niveles bajos (transferencia en proceso). Cabe mencionar que para el procesamiento de los datos se redujo el número de muestras en un 50% para minimizar la carga en el programa. Además, para evitar falsos cambio de nivel lógico, dentro del programa se considera el cambio de estado lógico si la señal rebasa los 2.8 volts en un flanco de subida, mientras que para ser considerado bajo se aplicó un criterio de 1.4 volts.

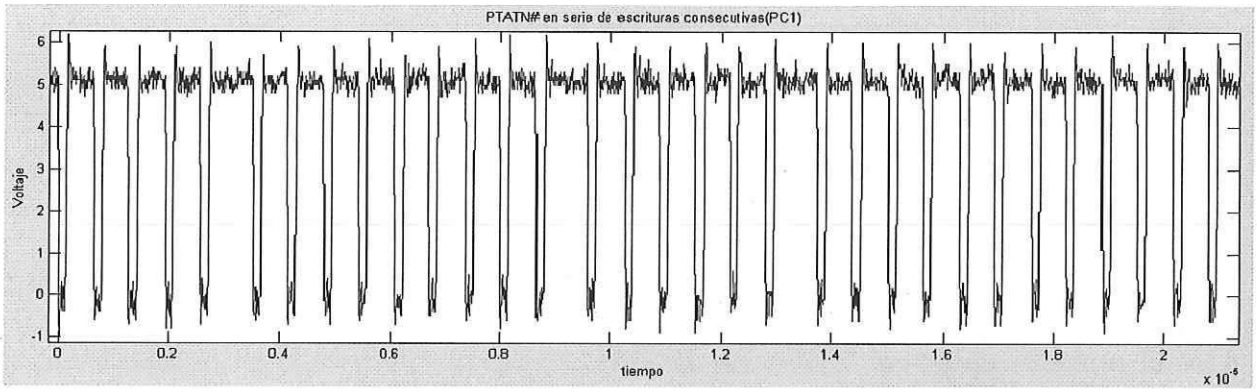


Figura 16. Señal PTATN durante 32 operaciones de escritura consecutivas (PC1)

En la figura 16, se muestra la señal PTATN con el sistema instalado en la PC1, la duración promedio de las transacciones fue de 146 nanosegundos, con una mediana de 144 nanosegundos y una desviación estándar de 3.86 nanosegundos. Los tiempos muertos obtuvieron una duración promedio de 524 nanosegundos, mediana de 512 nanosegundos con desviación estándar de 86.7 nanosegundos. Este último dato, a primera vista alarmante y probablemente erróneo, se debe a dos tiempos muertos excesivamente largos, ubicados cerca del centro de la grafica correspondiente. En la figura 17 se muestra el resultado obtenido al instalar la tarjeta en la PC2.

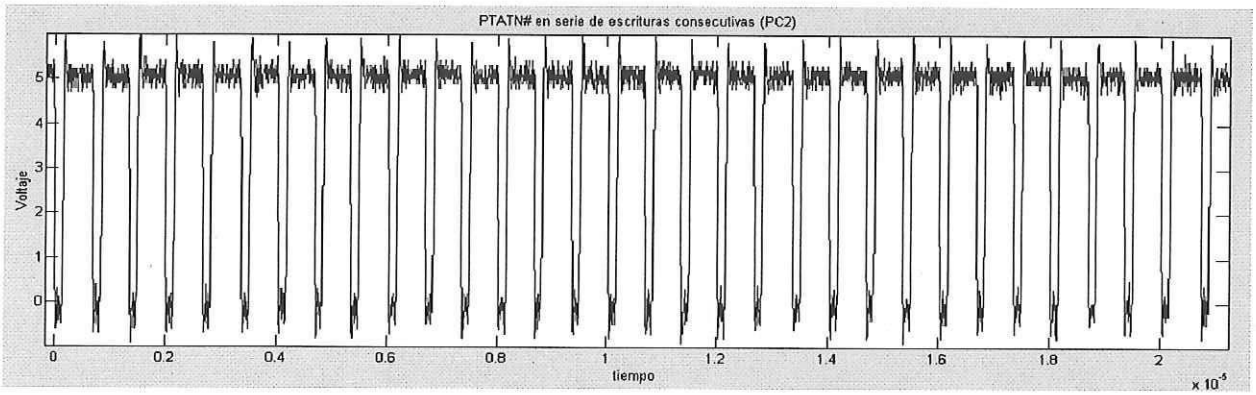


Figura 17. Señal PTATN durante 32 operaciones de escritura consecutivas (PC2)

Para la PC2, el tiempo promedio de las transacciones fue de 146 nanosegundos, con una mediana de 144 nanosegundos una desviación estándar de 3.51 nanosegundos. Los tiempos muertos obtuvieron una duración promedio de 523 nanosegundos, mediana de 520 nanosegundos y desviación estándar de 16.2 nanosegundos.

De este análisis se concluye que no hay diferencias considerables en el desempeño del sistema si es instalado en otra computadora. En parte es debido a que gran parte del manejo de datos que circulan por el ducto PCI se realiza sin la intervención del microprocesador del sistema, el cual es uno de los fines buscados cuando se desarrolló dicho ducto: liberar al microprocesador de la tarea de mover datos entre dispositivos, sobre todo cuando esta información solo interesa a los involucrados en la transferencia.

6.2 Mejoras al sistema

Durante una transferencia en rafaga, desde el punto de vista de la aplicación externa, la FIFO solo parecerá llenarse/vaciarse demasiado rápido, para ello, es recomendable que se provea un medio para resolver este problema y evitar que las transferencias se interrumpan constantemente. El CPLD utilizado en el desarrollo de la presente investigación, es capaz de leer la FIFO cada dos ciclos de reloj, por lo que las transacciones se verán interrumpidas varias veces si se transfiere un número de datos mayor a la profundidad de la FIFO. Esto supone una grave desventaja, aunque en la realidad, los sistemas anfitriones PCI de las placas madre no permiten largas transferencias en rafaga sin interrupciones; esto con el fin de distribuir el tiempo de ducto de manera justa y eficaz entre los diferentes dispositivos PCI que lo comparten.

Para solucionar esto, se recomienda reemplazar el CPLD por otro similar con tiempos de propagación menores. Si bien nuestro CPLD cuenta con un tiempo de propagación de las entradas a las salidas de 7 nanosegundos y la señal de reloj utilizada para sincronización tiene un periodo de 33 nanosegundos; en algunas ocasiones es necesaria una retroalimentación de las salidas a las entradas y forzosamente la información tiene que circular por el CPLD mas de una vez, de manera que el tiempo en que el CPLD responde a un estado en especial puede ser varias veces el especificado por las hojas de datos. Una vez que sea posible completar todas las iteraciones necesarias dentro del CPLD, se deberá modificar el código en VHDL del programa que reside en el CPLD. Específicamente la sección que maneja los datos que fluyen a través de la FIFO, de manera que pueda transferir un dato por ciclo de reloj, en otras palabras, que sea capaz de manejar transferencias de información en rafagas. Esto fue corroborado en una prueba que se hizo generando un acceso en rafaga a memoria desde la tarjeta PCI: efectivamente, los datos fueron recuperados cada segundo ciclo de reloj, pero no siempre fue posible mantener

constante la salida de datos. En promedio cada 8 datos recuperados existía una interrupción en la rafaga, lo cual se atribuye a un llenado de la FIFO lo cual origina una desconexión temporal del ducto PCI. Lo anterior se traduce en un aumento en la velocidad de transmisión de datos; sin embargo, se pierde control sobre los tiempos muertos (latencias) de los datos.

Una vez que el hardware de la tarjeta sea capaz de transferir datos con el máximo ancho de banda permitido, forzosamente el software debe estar diseñado para manejar grandes cantidades de información eficientemente.

Durante el desarrollo, se exploraron dos modalidades para el controlador de nuestro sistema de adquisición: el programa en espacio de usuario revisado con anterioridad; y un módulo montable sobre el Kernel durante el tiempo de ejecución. La programación de un módulo, fue descartada debido a que no existía una diferencia notable en cuanto al desempeño del hardware se refiere durante las primeras pruebas. Cuando un módulo es montado sobre el Kernel, su código es añadido y su ejecución afecta al desempeño global del sistema; si este módulo contiene errores y falla, todo el sistema lo hace también. La solución a esto es presionar el botón RESET de la computadora y hacer un chequeo de la superficie del disco duro para buscar errores y posibles daños. El programa en espacio de usuario no presenta esta última desventaja, ya que si éste falla, solo será necesario detener el proceso en ejecución, corregir el error e intentar de nuevo. El mayor problema encontrado con la programación en espacio de usuario, es la imposibilidad de obtener la dirección física en memoria de una variable o bloque de datos utilizados en espacio de usuario, ya que estas direcciones son virtuales. La razón de esto es la misma protección del sistema operativo, al separar el espacio de memoria asignado al Kernel y el de espacio de usuario. Una dirección física es necesaria para realizar transferencias a chorro utilizando la FIFO, puesto que el S5935 intentará acceder una dirección física desde el ducto PCI.

La solución a esto es: crear un módulo, reservar un espacio de memoria continua con dirección lógica en el espacio de direcciones del Kernel y recuperar su dirección física para después ser transferida al registro correspondiente para la transferencia en rafaga, pero debido a limitaciones de tiempo, no fue explorada.

En un futuro será necesaria la elaboración de un módulo manejador que resida en el Kernel y solicite una sección de memoria durante la inicialización del sistema. Dicha porción de memoria deberá ser continua para poder ser accesada eficazmente por la tarjeta PCI en una transferencia en chorro a la primer dirección de ese bloque de memoria.

Por último, es integrar todos los elementos del sistema en una solo circuito impreso, con el fin de eliminar las conexiones mediante cables planos y reducir el espacio que actualmente ocupa el sistema.

6.3 Conclusiones

En base a lo expuesto en el presente documento, se concluye que es posible crear un sistema de transferencia de datos suficientemente rápido para controlar un instrumento de uso científico, tal como un CCD. El sistema actualmente es capaz de realizar operaciones de lectura y escritura sobre un dispositivo externo capaz de responder a dichas operaciones. Además, este primer sistema desarrollado en los laboratorios del Instituto de Astronomía de la UNAM, sirve como punto de arranque para desarrollar otros sistemas con interfaces diseñadas para algún instrumento en particular. También es importante hacer notar que un sistema de transferencia de datos que utiliza el ducto PCI, tendrá un mejor desempeño si en lugar de transferir dato por dato, los dispositivos externos son capaces de proveer/aceptar grandes cantidades de datos sin interrupciones.

Una buena cantidad de los problemas encontrados fueron resueltos satisfactoriamente, mientras que otros no pueden ser resueltos mediante las herramientas disponibles actualmente, tal es el caso del CPLD o el tiempo de uso del ducto que un anfitrión PCI asigna a un dispositivo en particular.

7.- Referencias y bibliografía

Doug Abbot. PCI Bus Demystified. LLH Technology Publishing. 2000.

S5935 PCI Product Databook. Applied Micro Circuits Corporation.1999.
http://www.amccc.com/

S5935 PCI MatchMaker Developer's Kit Technical Reference Manual. Applied Micro Circuits Corporation. 2001.
http://www.amccc.com/

Ultra37000 CPLD Family. Cypress Semiconductor Corporation. Marzo 2004.
http://www.cypresssemiconductor.com/

Alessandro Rubini, Jonathan Corbet. Linux Device Drivers, Second Edition 2001.
http://www.oreilly.com/catalog/linuxdrive2/

J. Mirkowski, M Kapustka. EVITA – Enhanced VHDL Tutorial with Applications. Rev 2.0,1998
http://www.aldec.com/products/tutorials/

Apéndice A. Rutinas de programación realizadas

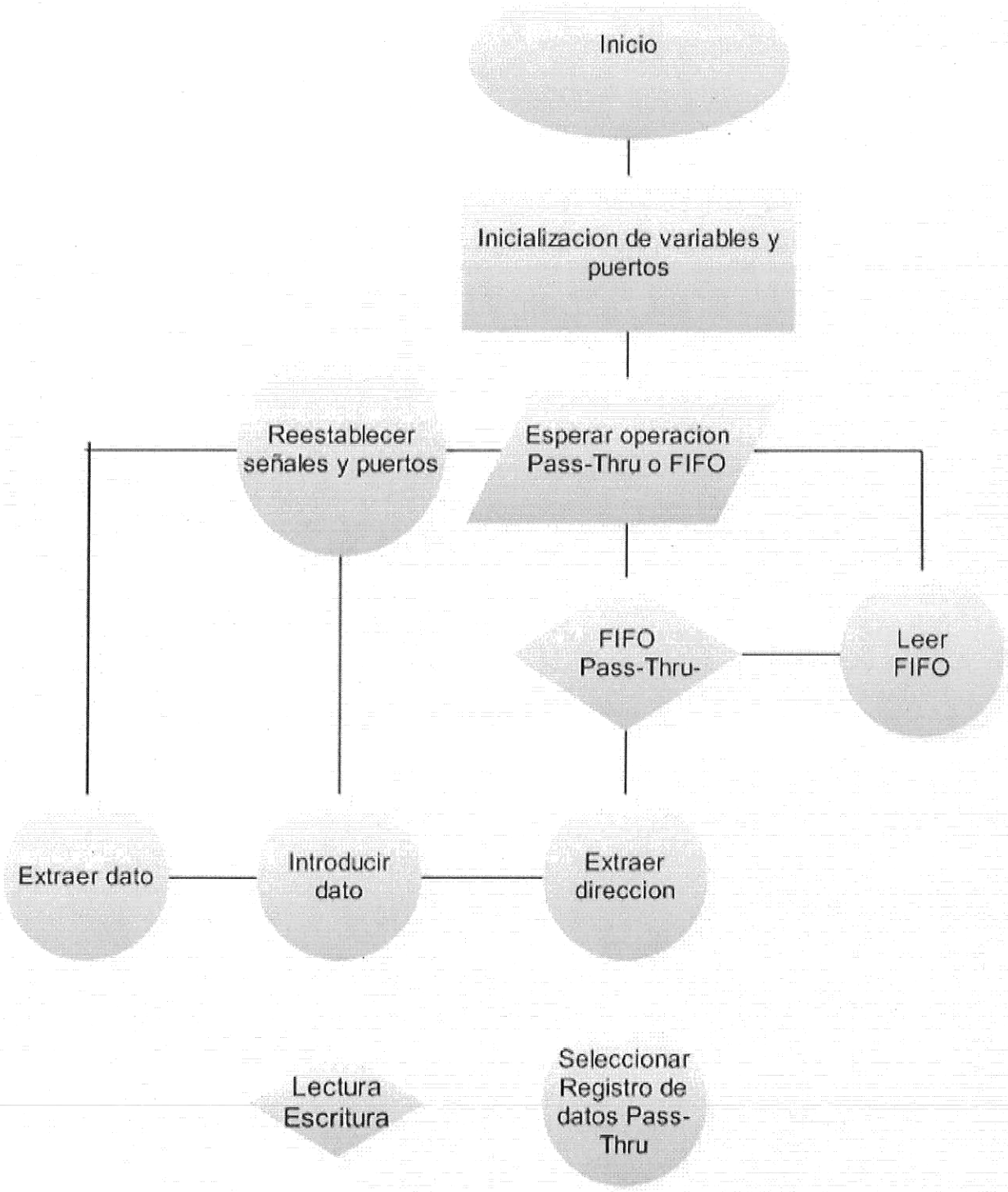


Diagrama de flujo para el programa en el CPLD

Listado del programa en VHDL para el CPLD CY37128

```
--      Los comentarios en este listado son aquellos
--      que aparecen después de dos guiones seguidos
--      en una línea como en este caso.

--      Programa para la interfaz Bus de usuario del
--      controlador de ducto PCI S5935 hacia una
--      interfaz tipo IDE al exterior. El CPLD lleva
--      a cabo operaciones tipo Pass-Thru y operaciones
--      de escritura de la FIFO (PCI to Add-on FIFO).


--      Se define la librería básica a utilizarse por
--      nuestro programa

library IEEE;
use             IEEE.STD_LOGIC_1164.all;


--      Se definen la entidad y los puertos que contiene
entity amcc128 is
port(
        clk                             :in std_logic;
        ptatn,ptwr,ptbrst       :in std_logic;
        rdempty                         :in std_logic;
        ibytel,ibyteh                   :inout std_logic_vector(7 downto 0);
        pbytel,pbyteh                   :inout std_logic_vector(7 downto 0);
        rdfifo                          :out std_logic :='1';
        ptrdy,ptadr                     :out std_logic :='1';
        be3,be2,be1,be0                 :out std_logic :='0';
        padr                            :out std_logic_vector (6 downto 2):="01011";
        rd,wr                           :out std_logic :='1';
        sel                             :out std_logic :='0';
        idewr,iderd,idecs       :out std_logic :='0';
        ideadr2,ideadr1,ideadr0         :out std_logic :='0'
        );

--      En las siguientes líneas se asignan puertos
--      de la entidad a pins físicos en el CPLD

 attribute pin_numbers of amcc64: entity is
 " clk:62 " &
 " rd:59" &
 " wr:58" &
 " sel:60" &
 " be3:71" &
 " be2:73" &
 " be1:70" &
 " be0:72" &
 " padr(6):69" &
 " padr(5):61" &
 " padr(4):66" &
```

```
 " padr(3):67" &
 " padr(2):68" &

 " ptwr:12 " &
 " ptatn:16 " &
 " ptbrst:15 " &
 " ptadr:13" &
 " ptrdy:14" &

 " pbytel(0):10" &
 " pbytel(1):9" &
 " pbytel(2):8" &
 " pbytel(3):7" &
 " pbytel(4):6" &
 " pbytel(5):5" &
 " pbytel(6):4" &
 " pbytel(7):3" &

 " pbyteh(0):82" &
 " pbyteh(1):81" &
 " pbyteh(2):80" &
 " pbyteh(3):79" &
 " pbyteh(4):78" &
 " pbyteh(5):77" &
 " pbyteh(6):76" &
 " pbyteh(7):75" &

 " iderd:27" &
 " idewr:28" &
 " idecs:26" &
 " ideadr2:29" &
 " ideadr1:30" &
 " ideadr0:31" &
 " rdfifo:56" &
 " rdempty:54" &

 " ibytel(7):33" &
 " ibytel(6):34" &
 " ibytel(5):35" &
 " ibytel(4):36" &
 " ibytel(3):37" &
 " ibytel(2):38" &
 " ibytel(1):39" &
 " ibytel(0):40" &

 " ibyteh(7):45" &
 " ibyteh(6):46" &
 " ibyteh(5):47" &
 " ibyteh(4):48" &
 " ibyteh(3):49" &
 " ibyteh(2):50" &
 " ibyteh(1):51" &
 " ibyteh(0):52"
  ;
```

```
end     amcc128;

architecture arqide128 of amcc128 is
type estados is (es0,es1,es2,es3,es4,es5);
signal presente  : estados bus:= es0;
signal siguiente : estados bus:= es0;

--      Inicio de la descripción y procesos que se
--      llevan a cabo en este sistema.

begin

--      Este es el proceso principal. Se trata de una
--      maquina de estados finitos que actúa conforme
--      al estado que se encuentre y en algunos casos
--      la respuesta de este agente dependerá de las
--      entradas.
--      El sistema se encargara de hacer la tarea
--      correspondiente al estado en que se encuentre
--      y de acuerdo a lo que le ordenen sus entradas.

proceso1: process(presente)
        begin
                case presente is
                        when es0 =>
                                pbytel<="ZZZZZZZZ";
                                pbyteh<="ZZZZZZZZ";
                                rd<= '1';
                                wr<= '1';
                                sel<= '1';
                                ptrdy<= '1';
                                pbyteh<="ZZZZZZZZ";
                                pbytel<="ZZZZZZZZ";
                                ibyteh<="ZZZZZZZZ";
                                ibytel<="ZZZZZZZZ";

                                rdfifo<='1';
                                idecs<='1';
                                idewr<='1';
                                siguiente <= es0;
                                if (rdempty='0') then
                                        rdfifo<='0';
                                        idecs<='0';
                                        siguiente<=es5;
                                end if;
                                if (ptatn = '0') then
                                                siguiente <= es1;
                                                be3<='0';
                                                be2<='0';
                                                be1<='0';
                                                be0<='0';
                                                sel<= '0';
                                                padr(6)<='0';
```

```
                padr(5)<='1';
                padr(4)<='0';
                padr(3)<='1';
                padr(2)<='1';
                ptadr<='0';
        end if;

when es1 =>
        ideadr2<=pbytel(4);
        ideadr1<=pbytel(3);
        ideadr0<=pbytel(2);
        siguiente<=es2;

when es2 =>
        if ptwr = '0'then
                wr<='0';
        else
                rd<='0';
        end if;
        ptadr<='1';
        idecs<='0';
        pbyteh<="ZZZZZZZZ";
        pbytel<="ZZZZZZZZ";
        siguiente<=es3;

when es3 =>
        if ptwr = '1'then
                ibytel<=pbytel;
                ibyteh<=pbyteh;
                idewr<='0';
        else
                pbytel<=ibytel;
                pbyteh<=ibyteh;
                iderd<='0';
        end     if;
        ptrdy<='0';
        siguiente<=es4;
        if ptbrst = '0' then
        siguiente<=es3;
        end if;

when es4 =>
        ptrdy<='1';
        pbyteh<="ZZZZZZZZ";
        pbytel<="ZZZZZZZZ";
        ibyteh<="ZZZZZZZZ";
        ibytel<="ZZZZZZZZ";
        rd<='1';
        wr<= '1';
        sel<='1';
        idecs<='1';
        idewr<='1';
        iderd<='1';
        be3<='1';
```

```
                    be2<='1';
                    be1<='1';
                    be0<='1';
                    padr(6)<='1';
                    padr(5)<='1';
                    padr(4)<='1';
                    padr(3)<='1';
                    padr(2)<='1';
                    ptadr<='1';
                    siguiente <= es0;

                when es5 =>
                    ibytel<=pbytel;
                    ibyteh<=pbyteh;
                    idewr<='0';
                    rdfifo<='1';
                    siguiente<=es0;
                end case;
end process proceso1;

--      El siguiente proceso se encarga de hacer que
--      la maquina de estados finita avance cuando
--      exista un cambio de 0 a 1 en clk.

avance : process (clk)
        begin
        if(clk'event and clk = '1') then
                presente <= siguiente;
                end if;
        end process avance;

end arqide128;
```

Listado del programa escrito en lenguaje C para el control de la tarjeta.

```
/*      El siguiente programa fue escrito con la finalidad de hacer pruebas

        de funcionamiento y desempeño del sistema de transferencia de datos
        presentado. Las funciones presentadas fueron creadas con el fin de
        caracterizar el sistema, y no necesariamente manejan información útil        */


#include <fcntl.h>              /* open */
#include <unistd.h>             /* exit */

/* Librerías adicionales necesarias para esta aplicación*/
#include <stdio.h>
#include <stdlib.h>
#include <asm/io.h>

 main()
{

/* Se declaran e inicializan las variables a utilizar en el programa */
  int i,j;
  j=0;
  char *b[32];
  long dato=0,dati=0;
  long fifoin,fifoout;
  long conf;
  long conf2;
  long conf3;
  long contr;
  long fifoadr;
  long registro;
  long badr = 0xc000;
  long ptr2=0xc400;
  long ptr1=0xc800;
  long badr2=0xc000;
  int ptadroffset;
  iopl(3);
  j=0;
  conf2=inl(badr+0x3c);
  printf("%04x  ",conf2);
  printf("%04x  ",(conf2)|(0xa000));
  outl((conf2)|(0xa000),badr+0x3c);
  conf=inl(badr+0x38);
  printf("\n el conf 38 tiene %x",conf);
  conf=(conf)&(0x00000000);
  printf(" y luego un %x",conf);
  outl(conf,badr+0x38);

   /* La siguiente instrucción obtiene permisos para realizar operaciones a puertos*/

   if( iopl(3) == -1)
  printf("Error al abrir puertos\n");
```

```
/* Se presenta un menú con las diferentes opciones a escoger para probar
 alguna función de la tarjeta */

 while (i != 0 )
 {
        conf = inl(badr+0x3c);
        printf ( "\n \n el registro tiene %x \n",conf);
        printf("\n Escoje una opcion : \n\n");
        printf("  1) Sacar un dato por la fifo \n");
        printf("  2) Sacar un chorro por la fifo\n");
        printf("  3) sacar un dato por un puerto pass-thru \n");
        printf("  4) leer  un dato pass-thru \n");
        printf("  5) Sacar una serie de datos por la fifo (no chorro)  \n");
        printf("  6) sacar varios datos por pass thru  I/O \n");
        printf("  0) Salir \n");
        printf("\n                    Opcion :  ");
        scanf("%d",&i);

        switch (i)
                {

        /* La primer opción, saca un dato si hay un espacio libre en la FIFO, si esta llena
        imprime un mensaje de error y se aborta la operación*/

                case 1:
                        printf("\npon el dato a sacar \n");
                        scanf("%d",&fifoout);
                        if (conf & 0x01)
                                {
                                printf (" no se saco nada porque la fifo esta llena\n")
                                ;break;
                                }
                        else
                                {
                                printf(" fifoout = %x   \n ",fifoout);
                                conf = inl(badr+0x3c);
                                printf ( "\n \n el registro tiene %x \n",conf);
                                outl(fifoout,badr+0x20);
                                if (conf & 0x01) printf("la fifo de salida se ha llenado\n");
                                }
                        break;
                        }

        /* La segunda opción prepara y manda hacer una transferencia en chorro iniciada por la tarjeta,
        según se describe en el capitulo ... */

                case 2:
                        {
                        conf3=inl(badr+0x3c) & 0xffffb3ff;                      //se deshabilitan transferencias
                        outl(conf3,badr+0x3c);
                        fifoadr=badr2+0x38;
                        b[0]=1;
                        printf("\n la direccion es %x \n el registro tiene %x  %x\n ",fifoadr,conf3,0x804a1e8);
                        conf3=inl(badr+0x3c);
```

```
conf3=(conf3)|(0x06000000);                    // se reestablecen banderas
printf("\n con mod conf = %x",conf3);
outl(fifoadr,badr+0x2c);
fifoadr=1;
outl(conf3,badr+0x3c);
outl(32,badr+0x30);
contr=inl(badr+0x30);
conf3=inl(badr+0x3c);
fifoadr=inl(badr+0x2c);
printf("\n la direccion  ahora tiene %x  y el contador %x",fifoadr,contr);
outl(conf3|(0x00004000),badr+0x3c);            //se habilita transferencia
wait(10000);
contr=inl(badr+0x30);
conf3=inl(badr+0x3c);
printf("\n reg = %x contador = %x",conf3,contr);
dato=inl(badr2+0x38);
printf("\n dato es : %x",dato);
break;
}
```

/* La tercera opción pide introducir un numero que posteriormente sacara por un puerto Pass-Thru, este puerto será un numero múltiplo de 4 entre 0 y 28 */

```
case 3 :
        printf("\n introduce un numero para sacar     ");
        scanf("%d",&dato);
        printf("\n por que puerto ?  (1,2,3)    -----> ");
        scanf("%d", &ptadroffset);
        printf("\n \n     %d    %x",dato,ptr1+ptadroffset);
        outl(dato,ptr1+ptadroffset);
        break;
```

/* Esta cuarta rutina lee mediante una operación Pass-Thru el contenido en bus para el usuario (Add-On) */

```
case 4 :
        printf("\n\n leyendo dato pass-thru\n");
        dati=inw(ptr1);
        printf("\n El dato es %x \n\n ",dati);
        break;
```

/* La siguiente opción saca una serie de datos en cadena por la FITO, pero no es una transferencia en chorro */

```
case 5 :
        printf("\n sacando un chorro    ");
        outsw(ptr1+ptadroffset,b,32);
        break;
```

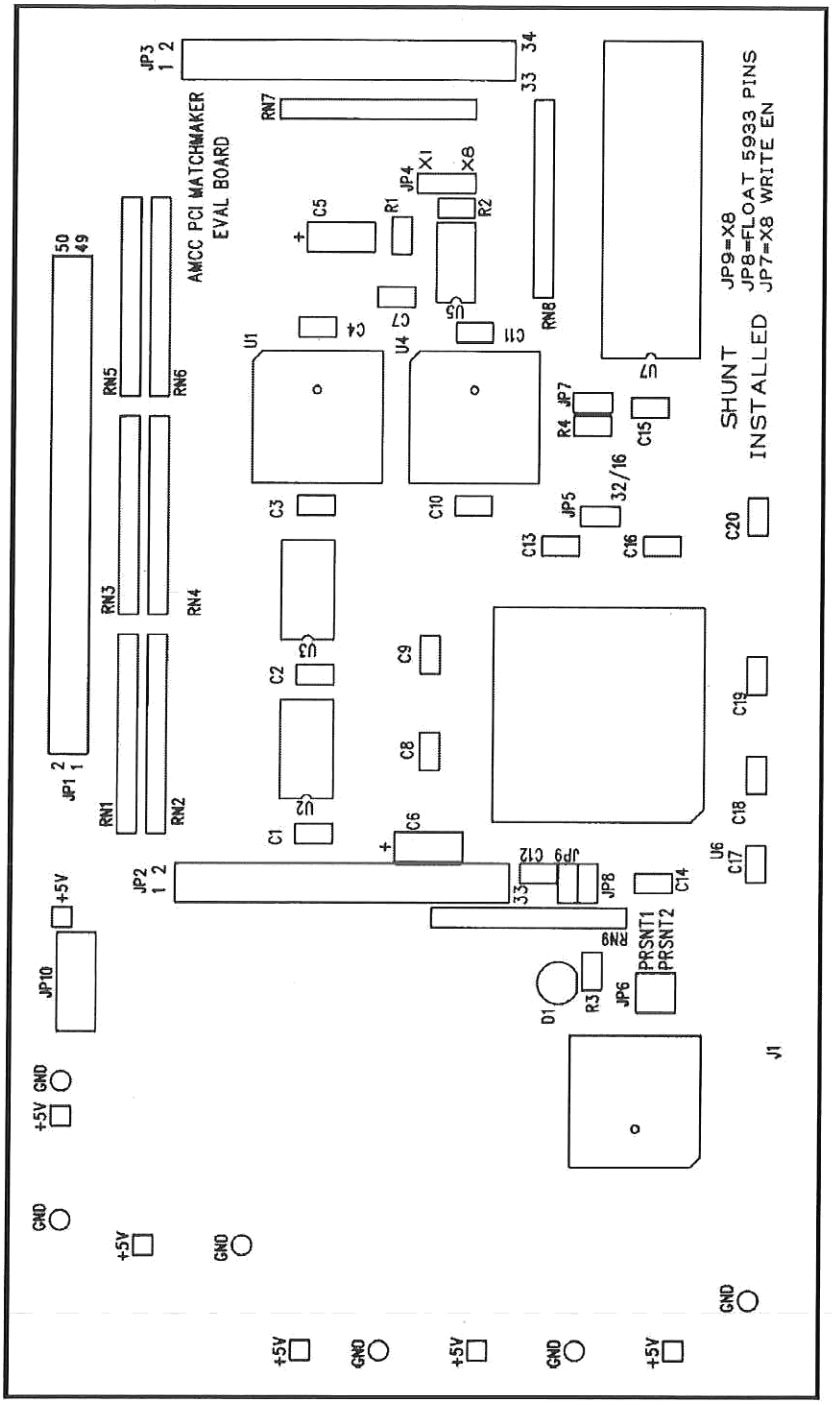
/* La siguiente opción saca una serie de datos en cadena por el ultimo puerto Pass-Thru seleccionado, pero no es una transferencia en chorro */

```
case 6:
        outl(dato,ptr1+ptadroffset);
```

```
                    outl(dato,ptr1+ptadroffset);
                    outl(dato,ptr1+ptadroffset);
                    outl(dato,ptr1+ptadroffset);
                    break;

                default : break;
            }
    }
return 0;
}
```

Apéndice B. Diagramas esquemáticos y distribución de componentes



Distribución de componentes en la tarjeta PCI del kit de desarrollo S5935DK

Los componentes más importantes son:

(1) Controlador AMCC S5935 PCI Matchmaker (U6)

(1) Memoria Flash 29C512 64K x 8 (U7)

(1) EEPROM Serial 24C16 2K x 8 (U5)

(2) Latch Octal 74F374 (U2, U3)

(2) PLDs 22V10 o 22CV10-10 (U1, U4)

Los circuitos integrados U1, U2, U3, U4 fueron removidos puesto que la función que desempeñan por defecto la realiza el CPLD utilizado en el desarrollo de este trabajo de tesis.

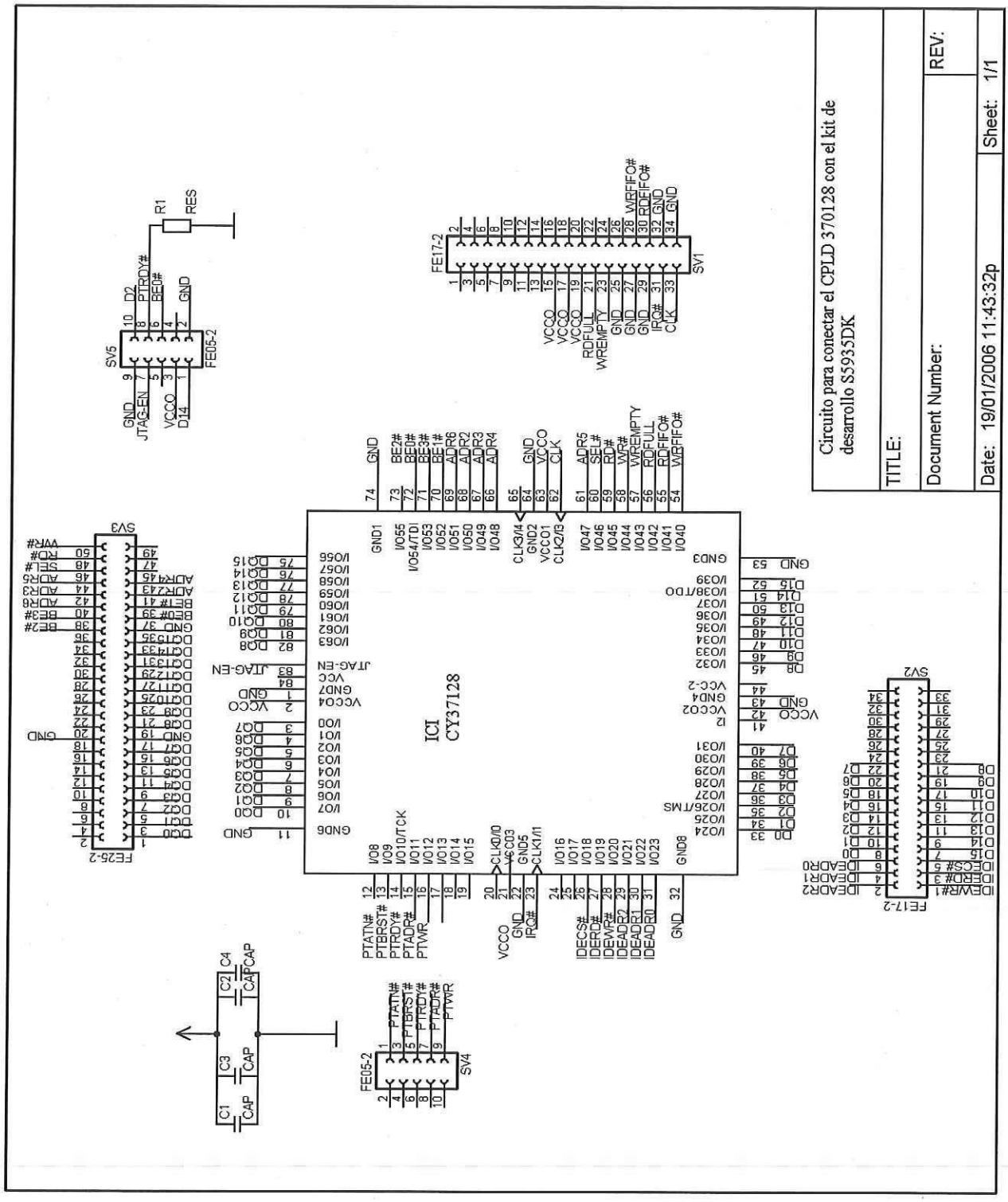
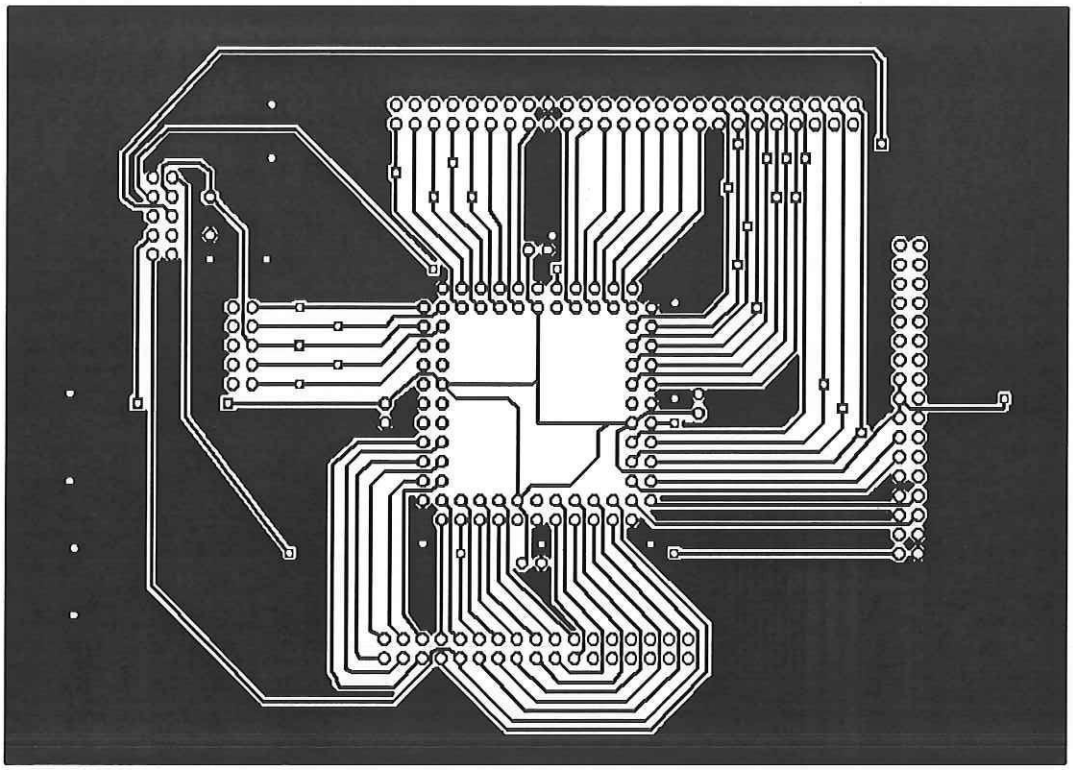
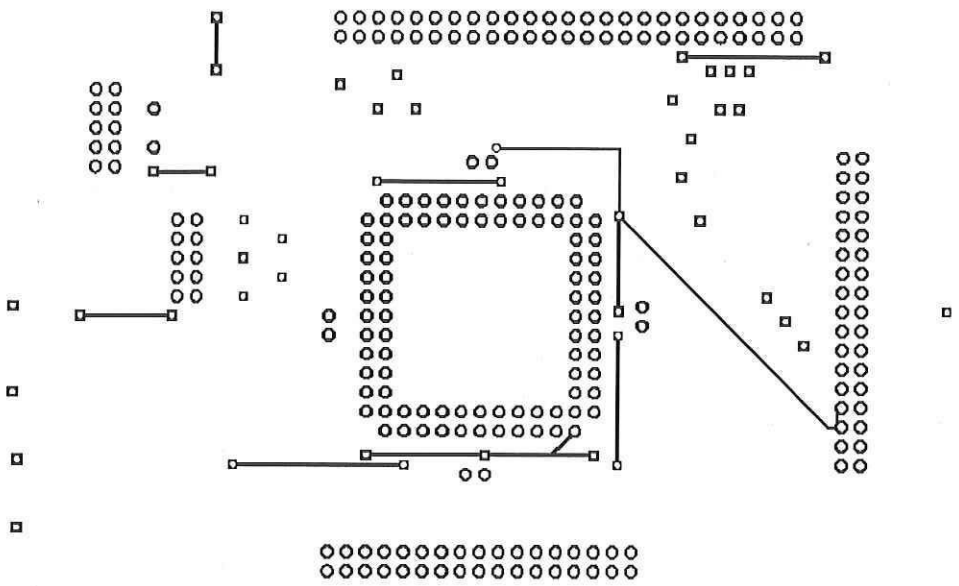


Diagrama esquemático de la tarjeta auxiliar que contiene al CPLD CY37128.



Mascara en negativo cara inferior para el circuito impreso correspondiente al diagrama esquemático del circuito impreso con el CPLD.



Mascara en negativo cara superior para el circuito impreso correspondiente al diagrama esquemático del circuito impreso con el CPLD.

Notación:

Un signo # al final de un nombre de señal, indica que esta señal es activada o asserted en el estado de voltaje bajo. Las señales sin # son activadas en su estado de voltaje alto. La notación [n::m], donde n y m son enteros tal que n es mayor que m, representa un arreglo de señales con n-m+1 elementos. Entonces, AD[31::0] representa al bus de datos de 32 bits que contiene las señales AD[0] hasta AD[31] con AD[0] como el bit menos significativo.

Tipos de Señales

in: Entrada únicamente

- CLK, RST#, IDSEL, TCK, TDI, TMS, TRST#, PRSNT[1:2] [1], CLKRUN#,M66EN,3.3VAux

out: Salida únicamente. Activa, tótem-pole.

- TDO

t/s: Entrada-salidas bidireccionales con tercer estado.

- AD[31::0], C/BE[3::0], PAR, REQ#,GNT#,CLKRUN#

s/t/s: Tercer estado sostenido. Controlado por un dueño a la vez. Nótese que todas las señales s/t/s son activadas en bajo. El dueño debe manejar la señal en alto, es decir, en su estado inactivo, por un ciclo de reloj antes de ponerla en tercer estado. Cualquier otro agente no debe manejar una señal s/t/s antes de un ciclo de reloj después de que el previo dueño la haya puesto en tercer estado. Las señales s/t/s requieren pull-up para mantener el estado inactivo hasta que otro agente la controle. Este pull-up debe ser proporcionado por la fuente principal, por ejemplo, la tarjeta madre.

- FRAME#, TRDY#, IRDY#, STOP#,LOCK#,PERR#

o/d : Open drain, wire-OR's permiten que múltiples dispositivos activen la señal simultáneamente. Se requiere que un pull-up sostenga la señal mientras ningún agente lo este manejando. Este pull-up debe ser proporcionado por la fuente principal, por ejemplo, la tarjeta madre.

- SERR#, INTA#-INTD#, CLKRUN#, PME#