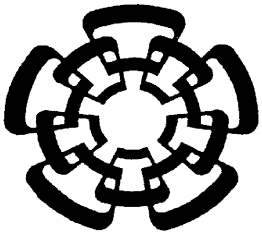
CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO NACIONAL
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA
SECCIÓN DE COMPUTACIÓN



BUFFER AUTOCOMPACTANTE PARA SWITCHES Y RUTEADORES EN SISTEMAS PARALELOS

Tesis que presenta

Armando Jiménez Flores

Para Obtener el Grado de

Doctor en Ciencias

En la Especialidad de

Ingeniería Eléctrica

Opción Computación

Director de la Tesis: Dr. Adriano De Luca Pennacchia

México D. F. Marzo 2006

Resumen

Para resolver grandes problemas, los arquitectos de computadoras buscan satisfacer la creciente demanda de capacidad de cómputo. Aún con el uso del paralelismo, el desempeño de estas máquinas está limitado por los retardos de comunicación entre sus elementos de procesamiento. Por lo que, el diseño de la red de interconexión es un factor decisivo en el desempeño de las computadoras paralelas. Típicamente, el diseño de la red se reduce al diseño del elemento de comunicación, conocido como *switch* o ruteador. Existen condiciones que hacen necesario el almacenamiento temporal de los datos de comunicación entre procesadores. Para ello se utilizan *buffers*, dentro del elemento de comunicación, que mejoran su desempeño. No obstante, si la administración del *buffer* es ineficiente, el desempeño del *switch* o ruteador se decrementa seriamente.

El objetivo de esta tesis es diseñar una memoria especial de alto desempeño que permita reducir los tiempos de comunicación de datos entre los elementos de procesamiento de un sistema paralelo. Esta memoria es utilizada como *buffer* donde se almacenan temporalmente los datos que se transfieren entre los procesadores del sistema, y es aplicable a *switches* y ruteadores en sistemas de procesamiento paralelo para mejorar la latencia de comunicación. Se diseña un modelo de *buffer* que administra dinámicamente los espacios de almacenamiento y permite la concurrencia de operaciones de escritura y lectura. Su modelo se basa en un conjunto de cinco operaciones fundamentales que incluye escrituras o lecturas simples y operaciones paralelas que combinan una escritura con una lectura. Al *buffer* lo hemos denominado *BAC* (*Buffer* Auto-Compactante), porque mantiene compactado su espacio de almacenamiento. El *BAC* es implementado en código VHDL (*Very-High-Speed-Integrated-Circuit Hardware Description Language*) y simulado a nivel lógico con una herramienta computacional basada en dicho lenguaje. Se busca que el modelo del *BAC* administre eficientemente los espacios y tenga propiedades de baja latencia que puedan ser aprovechadas en la implementación.

Para resaltar las propiedades de alto desempeño del *BAC,* se diseñó completamente en hardware, con un tipo de control distribuido paralelo que utiliza una celda de control por cada localidad del *buffer*. La capacidad del *BAC* es expandible: puede crecer en ancho y largo, manteniendo constantes la complejidad de sus celdas de control y el tamaño del bus de direcciones. Cada una de las cinco operaciones fundamentales del *BAC* se realiza durante un solo ciclo de reloj. Estas consideraciones en el diseño reducen los retardos de comunicación debidos a la administración de los *buffers* e incrementan el desempeño de los elementos de comunicación.

Las contribuciones de esta tesis son las siguientes: Un análisis de las posibles configuraciones arquitecturales para un *buffer* de propósito específico desde la perspectiva de su desempeño, un diseño específico completo de un modelo de *buffer* (*BAC*) de baja latencia y acceso concurrente, una implementación en VHDL del *BAC*, un diseño y verificación funcional de los componentes que integran el *BAC*.

Palabras clave: paralelismo, *switch*, ruteador, *buffer*, autocompactante, latencia, VHDL.

Abstract

In order to solve large problems, the goal of computer architects is to satisfy the growing computational demand. Even with the use of parallel computing, the performance of these machines is limited by communication delays among its processing elements. That is why interconnection network design is a decisive factor in parallel computer performance. Typically, the design of the network focusses on the design of the communication element known as switch or router. There are conditions that make necessary to keep communication data in temporary storage. Such storage is provided by buffers inside the communication element to improve network performance. Nevertheless, when buffer management is inefficient, the switch or router performance may seriously deteriorate.

The goal of this thesis is to design a high performance special memory that allows the reduction of data communication delays among the processing elements of a parallel system. This special memory is used as buffer where the data transferred among system processors is temporarily stored and can be used by switches and routers in parallel processing systems to improve communication latency. A buffer model is designed to dynamically manage the storage spaces allowing concurrent writing and reading operations. Such model is based on a group of five fundamental operations that include simple writings or readings and parallel operations that combine one writing with one reading. The buffer is referred to as BAC ("Buffer Auto-Compactante"–Self-Compacting Buffer), because it maintains compacted its storage space. The BAC is implemented in VHDL code (Very-High-Speed-Integrated-Circuit Hardware Description Language) and is simulated at logical level with a CAD tool based on this language. The aim of the BAC model is to manage the storage spaces efficiently and to have low latency properties, that can be taken advantage of by the implementation.

To reach the BAC performance full potential, the buffer was totally designed in hardware with a parallel distributed control type that uses a control cell for each buffer locality. The BAC capacity is expandable: it can grow both in width and length, maintaining constant the complexity of the control cells and the address bus size. Each one of the five BAC's fundamental operations is carried out during a single clock cycle. These design considerations reduce the communication delays caused by buffers' management and increase the performance of the communication elements.

The contributions of this thesis are the following: An analysis of architectural possible configurations for a specific purpose buffer from the perspective of their performance, a complete specific design of a buffer model (BAC) of low latency and concurrent access, a BAC implementation on VHDL, and a design and functional verification of the components that integrate the BAC.

Keywords: parallelism, switch, router, buffer, self-compacting, latency, VHDL.

A mis adorables hijas: Abigail, Diana y Angélica

A mi amada esposa: Mary

A mis queridos padres: Santa e Ignacio

A mis entrañables hermanos: Mina, Nacho y Gaby

GRACIAS

Agradecimientos

Agradezco ampliamente a mi director de tesis Dr. Adriano De Luca Pennacchia, el haberme aceptado como su discípulo en la sugestiva aventura de realizar un doctorado. El tema de investigación que me planteó, así como, su guía y asesoría me permitieron desarrollar este trabajo de tesis.

Igualmente, estoy agradecido con el Dr. Sergio V. Chapa Vergara porque, durante su gestión como jefe de la Sección de Computación del Centro de Investigación y de Estudios Avanzados (CINVESTAV) del Instituto Politécnico Nacional (IPN), me apoyó para ingresar como docente y para ser aceptado como alumno de doctorado. Sus observaciones durante el examen predoctoral y sus recomendaciones finales mejoraron la estructura y el contenido de este documento.

De manera muy especial, agradezco al Dr. Héctor Ruiz Barradas por su permanente interés que, como compañero y profesor, ha mostrado por mi trabajo. Por su valioso tiempo dedicado en la revisión y corrección de este documento. Por la asesoría sobre la notación utilizada y sus invaluables consejos.

Mi profundo agradecimiento a las autoridades, profesores y personal administrativo del CINVESTAV del IPN, así como, a las autoridades del Consejo Nacional de Ciencia y Tecnología, por los apoyos recibidos durante la realización de mis estudios de Maestría y Doctorado.

Reconozco mi deuda con las autoridades del IPN, por todas las facilidades otorgadas. Particularmente, el Ing. Jorge E. Martínez Rodríguez y el Dr. Cornelio Robledo Sosa quienes, durante su gestión como directores de la Escuela Superior de Ingeniería Mecánica y Eléctrica (ESIME) del IPN, depositaron en mí su confianza. Asimismo, agradezco al Dr. José Madrid Flores y al M. en C. Salvador Saucedo Flores por los apoyos otorgados para ingresar a la planta docente de la ESIME.

Debo agradecer a las autoridades de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco (UAM-Azc.), por su comprensión durante la realización de este proyecto. Gracias también al Dr. Armando Maldonado Talamantes, quien me ayudó a ingresar a la planta docente del Departamento de Electrónica de la División de Ciencias Básicas e Ingeniería de la UAM-Azc.

Agradezco al Dr. José G. Delgado Frías sus recomendaciones para mejorar este documento y el interés mostrado en nuestro proyecto desde que se lo presentamos en la Universidad del Estado de Washington. Sus trabajos de investigación han sido para nosotros una valiosa referencia.

Mi agradecimiento al Dr. Maximino Peña Guerrero por el extraordinario entusiasmo con el que me apoyó durante la etapa de pruebas y por su solidaridad como amigo y compañero de equipo. Gracias por sus opiniones para mejorar la redacción.

Agradezco mucho al Dr. Miguel Ángel León Chávez la revisión que hizo a este documento, gracias a la cual se corrigieron errores en la redacción, se mejoró el contenido y se resaltaron fortalezas de la tesis.

Doy las gracias al Dr. José Antonio Moreno Cadenas por sus observaciones y recomendaciones que, a partir de mi examen predoctoral y durante la revisión final, conllevaron a mejorar este documento.

Expreso mi agradecimiento al Dr. Jorge Buenabad Chávez y al Dr. Arturo Díaz Pérez por sus recomendaciones que me ayudaron a mejorar el contenido de este documento.

Muchas gracias a la Lic. Laura Araceli Cedillo Jiménez y al Ing. Oscar Manuel Ortega Ortega por su valiosa colaboración durante la simulación dinámica de eventos. Asimismo, al Lic. José Urbano Cruz Cedillo y al Dr. Juan Jesús Ocampo Hidalgo, por la información técnica proporcionada y sus comentarios sobre el trabajo. Finalmente, a todos mis alumnos, colegas, familiares y amigos que de una u otra forma me apoyaron durante la realización de este trabajo.

ARMANDO JIMÉNEZ FLORES

No es la meta lo más importante,
sino aquello que se vive en el camino hacia ella.
Porque la esencia humana ha sido dotada de esa gran fuerza
que permite levantarse una y otra vez,
jamás sucumbir ante el dolor
y vencer toda adversidad.
Por ese gran espíritu con ímpetu inquebrantable
que prueba no haber estado solo.
Gracias, Dios.

A. JIMÉNEZ F.

Índice General

Índice de Figuras

Índice de tablas

Φ

Capítulo 1

Introducción

1.1 Antecedentes

Un área importante de investigación en computación busca incrementar el desempeño en los sistemas de cómputo, para satisfacer los requerimientos de aplicaciones cada vez más grandes y complejas. El desempeño de una computadora es recíproco del tiempo de ejecución, por consiguiente, a un mayor desempeño corresponde un menor tiempo de ejecución. Existen tres métricas fundamentales del desempeño: *latencia*, tiempo tomado para realizar una operación; *ancho de banda*, razón a la cual se realizan las operaciones; y *costo*, el impacto que estas operaciones tienen sobre el tiempo de ejecución de un programa.

Una alternativa que mejora el desempeño de las computadoras es el uso de tecnologías que permiten incrementar la velocidad de operación de sus circuitos. Sin embargo, al diseñar circuitos con mayor velocidad y densidad de integración, surgen problemas de disipación de calor que limitan los aumentos en la frecuencia de operación.

Otra alternativa para mejorar el desempeño se encuentra en la innovación de arquitecturas de computadoras que explotan la ejecución simultánea y sincronizada de varias tareas y que se conocen como computadoras paralelas. Los sistemas paralelos

mejoran substancialmente la capacidad de cómputo y actualmente se presentan como la clave tecnológica en la computación moderna que exige alto desempeño y costos bajos [1].

Una computadora paralela es una colección de entidades o nodos que se comunican y cooperan para "resolver grandes problemas rápidamente" [2]. Los nodos se comunican mediante un subsistema de comunicación que define las operaciones básicas de comunicación y sincronización y dirige la estructura organizacional que realiza dichas operaciones. La estructura organizacional del subsistema de comunicación, conocida como red de interconexión, está compuesta por interfaces nodo-a-red y enlaces que son el medio para interconectar los nodos. Típicamente, cada nodo contiene un elemento de procesamiento (EP), un elemento de memoria (EM), una interfaz nodo-a-red denominada elemento de comunicación (EC) y, dependiendo del sistema también puede contener otros dispositivos de soporte [36]. En cada uno de estos elementos se han desarrollado técnicas para mejorar su desempeño.

En los últimos años los EPs han sido diseñados con características tales como: procesamiento en línea (*pipeline*), colas de prebúsqueda (*prefetch*), múltiples unidades de ejecución (superescalar), múltiples tareas por ciclo de reloj (supersegmentación), algoritmos de predicción de ramificaciones y otras más. Asimismo, los EMs han sido mejorados con técnicas tales como: separación de datos e instrucciones (arquitecturas *Harvard*), bancos de memoria alternos (*interleaving*), memorias rápidas (*caché*), memorias anchas (VLIW), entre otras [68], [70], [39], [54], [75], [76]. Sin embargo, con el uso de estas técnicas el desempeño de las máquinas paralelas sigue estando limitado por los retardos en la comunicación y no por su lógica de procesamiento o de memoria. Por lo anterior, se puede ver que el diseño del subsistema de comunicación es un factor clave en el diseño de sistemas paralelos, dado que las ineficiencias de éste reducen en forma drástica el desempeño de todo el sistema. Además, la implementación de tales subsistemas es muy

crítica porque es la única parte del sistema paralelo que no se puede desarrollar a partir de componentes comerciales [16], [53]. Éstas han sido las principales motivaciones para desarrollar la presente tesis, en la cual se ofrece una alternativa que ayuda a mejorar el desempeño de las máquinas paralelas.

1.1.1 Arquitecturas de los Sistemas Paralelos

Durante la última década se ha reconocido una taxonomía para las máquinas paralelas MIMD (*Multiple Instruction stream over Multiple Data stream*) en la cual se consideran dos principales tipos de máquinas: *multicomputadora* y *multiprocesador de memoria compartida* [3], [4].

En un sistema multicomputadora los EPs disponen de memorias locales y la comunicación se realiza mediante un mecanismo denominado paso de mensajes. Mientras que, en los sistemas multiprocesador existe una memoria compartida entre los procesadores y la comunicación se realiza mediante operaciones de lectura/escritura a dicha memoria.

Los sistemas multicomputadora tienen la ventaja de ser económicos [35], [40], [41] y de fácil implementación, debido que se pueden construir a partir de una amplia variedad de productos comercialmente disponibles [44], [56]. Sin embargo, la ausencia de memoria compartida a nivel de *hardware* tiene implicaciones en la estructura del *software*. La programación de estos sistemas es complicada porque el programador debe procurar una distribución eficiente de código y datos entre los EP, invocando llamadas a funciones de paso de mensajes cada vez que algún dato es requerido por otros elementos de procesamiento. La complejidad de la programación se reduce mediante memorias compartidas distribuidas, las cuales proporcionan un modelo de programación donde el paso de mensajes es transparente al usuario. Esto provoca un detrimento del desempeño debido a la sobrecarga de los protocolos de coherencia de memoria. Por otro lado, los subsistemas de comunicación de las multicomputadoras de los 80s utilizaron componentes

comerciales, como los de las redes Ethernet [69], por lo que su lentitud se convirtió en *cuello de botella* y contribuyó en la motivación por el desarrollo de redes de alta velocidad.

En los sistemas multiprocesador de escala pequeña y moderada, el acceso a memoria compartida se realiza comúnmente mediante un bus común que ofrece acceso simétrico a toda la memoria desde cualquier procesador. Estos sistemas no son escalables porque el tiempo de acceso a memoria incluye la latencia de red, la cual se incrementa con el tamaño del sistema. Sin embargo, si en un sistema multiprocesador de memoria compartida, se distribuye físicamente la memoria entre los procesadores, entonces es posible incrementar la escalabilidad y obtener sistemas multiprocesador mayores.

1.1.2 El Subsistema de Comunicación

En las máquinas multicomputador y multiprocesador la función de la red de interconexión es crítica para su desempeño, porque la comunicación debe ser realizada con la menor latencia posible y permitiendo el mayor número de transferencias concurrentemente. Además, la red de interconexión debe ser relativamente económica con respecto al resto de la computadora paralela.

Existe una clasificación de redes de interconexión propuesta por Ni [5] con las siguientes cuatro categorías. (1) Redes de medio compartido: el medio de transmisión es compartido por todos los ECs del sistema; (2) Redes directas: enlaces punto a punto que conectan directamente cada nodo a un subconjunto de otros nodos en la red; (3) Redes indirectas: conectan a los nodos por medio de uno o más *switches*; (4) Redes híbridas: combinan mecanismos de redes de medio compartido con mecanismos de redes directas o indirectas.

En redes de medio compartido sólo un dispositivo a la vez puede usar la red. El problema principal en esta primera categoría es la estrategia para resolver los conflictos de acceso a la red. Su principal característica es la habilidad para soportar la transferencia de

unidades de datos indivisibles (emisión atómica) en la cual todos los dispositivos en el medio pueden monitorear las actividades de la red y recibir la información transmitida sobre el medio compartido. La desventaja principal está en su limitado ancho de banda de red.

Las redes directas o de punto a punto conectan a los nodos mediante ECs conocidos como ruteadores, los cuales se encargan de la comunicación entre nodos. Por este motivo, las redes directas se conocen como redes basadas en ruteador.

En las redes indirectas (a diferencia de las directas), los nodos se conectan a otros nodos mediante dispositivos conocidos como *switches*. Estas redes se conocen como *redes basadas en switch*. El elemento de comunicación en este tipo de redes es simplemente un adaptador de red que se conecta a un *switch* de red.

Las redes híbridas incrementan el ancho de banda con respecto a las redes de medio compartido y permiten distancias más cortas entre nodos con respecto a las redes directas e indirectas. Una aplicación típica de red híbrida es la conexión de LANs (*Local Area Networks*) a través de puentes [69]. Sin embargo, para sistemas que requieren muy alto desempeño las redes directas e indirectas alcanzan mejor escalabilidad que las redes híbridas, debido a que los enlaces punto a punto son más simples y rápidos que los *buses* de medio compartido. La mayoría de los sistemas de procesamiento paralelo usan redes directas o indirectas.

El alto desempeño de los sistemas paralelos se alcanza mediante el uso de redes de interconexión que provean un ancho de banda alto y una baja latencia de comunicación entre los EPs. [6], [7]. El desempeño de un sistema paralelo está profundamente influenciado por su escalabilidad. La propiedad de escalabilidad permite evitar los límites inherentes de diseño que existen cuando se agregan más recursos al sistema. Una arquitectura escalable incrementa proporcionalmente su desempeño con el incremento de

nodos. Los tipos de redes con mejores propiedades de escalabilidad, como se dijo en el párrafo anterior, son las redes basadas en *switch* y las basadas en ruteador. Con el fin de que los *switches* y los ruteadores no se conviertan en un *cuello de botella* se debe mejorar su diseño: éste es crítico en cuanto al desempeño de los sistemas paralelos.

1.1.3 El Switch y el Ruteador

Un *switch* consiste de un conjunto de puertos de entrada, un conjunto de puertos de salida, un *crossbar* interno que conecta a cada puerto de entrada con un puerto de salida y una lógica de control para efectuar la conexión entrada/salida en cada punto en el tiempo. Típicamente en cada puerto de entrada y/o de salida se tiene un *buffer* para almacenar los mensajes en tránsito. El *crossbar* es un arreglo de pares de contactos operados individualmente, en el cual hay un par para cada combinación entrada-salida [8]. La lógica de control implementa dos funciones importantes del *switch*: *control de flujo y control de ruta*. El *control de flujo* es un protocolo de sincronización para transmitir y recibir porciones de mensajes, con el fin de maximizar el ancho de banda de la red. La función de *control de ruta* determina la trayectoria más adecuada de los mensajes dentro de la red.

Un ruteador es básicamente un *switch*, el cual contiene un puerto interno bidireccional adicional a los puertos externos de entrada y salida de un *switch* básico y que sirve para conectarse al EP del nodo al cual pertenece.

Desde el punto de vista de desempeño del ruteador hay dos parámetros importantes [9]: *retardo de enrutamiento* y *latencia de control de flujo*. El primero se refiere al tiempo que transcurre desde que llega un mensaje al ruteador hasta que se determina el puerto de salida sobre el cual va a ser enviado. Típicamente, el retardo está constituido por el tiempo necesario para configurar el *crossbar*. El segundo parámetro corresponde a la taza de transmisión a la cual pueden enviarse los mensajes a través del ruteador. Esta taza es determinada por el retardo de propagación a través del *crossbar* y la taza de señalización

para sincronizar la transferencia de datos entre los *buffers* de los puertos de entrada y salida. Los retardos de enrutamiento y de control de flujo determinan, conjuntamente, la latencia de mensaje que se alcanza a través del ruteador [60].

El uso de *buffers* en el ruteador es forzoso para almacenar temporalmente la información que fluye a través de dicho ruteador [33]. Hay tres condiciones básicas donde el almacenamiento es necesario: 1) Cuando el puerto de salida, a través del cual se enrutará el paquete, se encuentra bloqueado por el próximo nodo de la red; 2) Cuando dos paquetes destinados al mismo puerto de salida llegan simultáneamente a diferentes puertos de entrada, pero el puerto de salida puede aceptar solamente un paquete a la vez; y, 3) Cuando los paquetes necesitan mantenerse mientras que el control de enrutamiento determina el puerto de salida para enrutar el paquete.

Además, el uso de *buffers* permite desacoplar los puertos de entrada y salida del ruteador y mejora significativamente el desempeño. Conforme se incrementa el tamaño del *buffer* se disminuye su probabilidad de saturación, mejorando así el desempeño del ruteador. No obstante, si la administración del *buffer* es ineficiente, el desempeño del ruteador puede decaer aún con *buffers* grandes. Por lo tanto, la administración de los *buffers* de un ruteador es crítica en su desempeño y por ende en el desempeño del sistema paralelo.

1.2 Objetivo

Este trabajo de tesis presenta el diseño de una memoria especial de alto desempeño que tiene como objetivo reducir los tiempos de comunicación de datos entre los elementos de procesamiento de un sistema paralelo. Dicha memoria es utilizada como *buffer* para almacenar temporalmente los datos que se transfieren entre los procesadores del sistema.

Para lograr este objetivo se analizan las arquitecturas posibles de los *switches* y ruteadores, que son los elementos encargados de la comunicación. Por lo que, el análisis se

realiza desde la perspectiva de sus *buffers*, comparando sus opciones de administración y complejidad en el diseño para alcanzar el mejor desempeño. Con base en este análisis, se elige un tipo de *buffer* de asignación dinámica y se desarrolla un modelo específico con propiedades de baja latencia y de acceso paralelo que puedan ser resaltadas en la implementación lógica.

1.3 Planteamiento del problema

En esta sección se describe la problemática de nuestro trabajo. Para ello, se hace un breve análisis de las posibilidades arquitecturales de *buffers* para *switches* y ruteadores y se establece un criterio de elección de una de ellas. La elección realizada se toma como referencia para desarrollar una propuesta de solución. Asimismo, se describen las características generales de la solución propuesta.

1.3.1 Administración de los Buffers

Existen tres opciones básicas de administración de *buffers*: *buffer* compartido centralizado, *buffers* independientes en las salidas y *buffers* independientes en las entradas [16, 13]. En la Tabla 1.1 se resumen algunas ventajas y desventajas importantes en cada una de estas opciones.

La opción de *buffer* compartido centralizado consiste en utilizar un espacio común de almacenamiento para todos los puertos de entrada. Esto permite, a todos los puertos de entrada, usar eficientemente el espacio total de almacenamiento disponible en el *switch*, en lugar de dividirlo estáticamente entre los puertos. Cada puerto de entrada puede escribir en el *buffer* independientemente del puerto de salida y cada puerto de salida lee los paquetes del *buffer*. La compartición del *buffer* puede bloquear innecesariamente los puertos de entrada. Así, un solo puerto de salida congestionado puede acaparar la mayor parte del *buffer* e impedir que el resto del tráfico de mensajes se mueva a través del *switch*. Esto

ocasiona una reducción en el desempeño. Por otro lado, su implementación resulta complicada debido a que todos los puertos de entrada necesitan acceder simultáneamente al *buffer* compartido, requiriendo una memoria con ancho de banda muy alto o una memoria multipuerto.

Tabla 1.1 Algunas ventajas y desventajas de las opciones de administración de *buffers*

Opción	Ventajas	Desventajas
Buffer compartido centralizado	Uso eficiente del espacio total disponible en el *switch* o el ruteador.	Bloqueo innecesario de puertos de entrada debido a que un puerto de salida congestionado acapara el *buffer.*
Buffers independientes en las salidas	No hay bloqueo innecesario de puertos de entrada. No hay acaparamiento del *buffer*. No hay canales de salida ociosos.	El *switch* debe operar tan rápido como la suma de velocidades de los puertos de entrada. Se requieren *buffers* multipuertos.
Buffers independientes en las entradas	No requieren *buffers* multipuerto. Evitan acaparamiento del *buffer*. Facilitan el manejo de paquetes de longitud variable.	Aparición de canales de salida ociosos debido a paquetes bloqueados innecesariamente.

La opción de *buffers* independientes en las entradas se basa en el uso de un espacio de almacenamiento independiente para cada puerto de entrada. Esta forma de administración tiene la ventaja de que sólo un paquete a la vez llega a un puerto de entrada, de modo que sólo se requiere un puerto de escritura. Así, se evita la necesidad de una memoria con mayor ancho de banda o del tipo multipuerto, como la requerida en la administración con *buffer* centralizado compartido. Además, los *buffers* independientes en cada puerto evitan el acaparamiento del espacio de almacenamiento por parte de algún puerto de salida congestionado, tal como sucede con el *buffer* centralizado compartido. Por otro lado, si los *buffers* son manejados como colas FIFO, entonces es muy fácil tratar con paquetes de longitud variable. Por estas razones, muchos *switches* existentes usan colas FIFO en los puertos de entrada [10], [11], [12]. El problema con esta forma de administración de *buffers*

es la aparición de canales de salida ociosos debido a paquetes bloqueados innecesariamente: Cuando un paquete en el encabezado de la cola, cuyo puerto de salida destino se encuentre ocupado, éste puede bloquear a todos los demás paquetes de la cola que está siendo transmitida, aunque su puerto de salida destino esté desocupado.

La administración con *buffers* independientes en las salidas se apoya en el uso de un espacio de almacenamiento independiente en cada puerto de salida. Esta opción ofrece la ventaja de generar longitudes de colas más cortas que las que se crean en los sistemas con *buffers* independientes en las entradas [13]. Esto se debe a que, con *buffers* en las entradas, los paquetes destinados a puertos de salida desocupados pueden ser encolados detrás de los paquetes con destino a puertos de salida ocupados. El uso de *buffers* independientes en las salidas resuelve los problemas de bloqueo innecesario de puertos de entrada y el acaparamiento del *buffer*, como sucede con la administración compartida centralizada. Asimismo, se resuelve el problema de canales de salida ociosos, que se presenta con el uso de *buffers* independientes en las entradas. El problema con los *buffers* en los puertos de salida es que el *switch* debe operar tan rápido como la suma de las velocidades de los puertos de entrada, o los *buffers* deben tener tantos puertos de escritura como puertos de entrada al *switch*, para permitir el arribo simultáneo de paquetes. La implementación de *buffers* con múltiples puertos de escritura, incrementa su tamaño y reduce su desempeño. Además, se dificulta la asignación eficiente de espacio de almacenamiento para paquetes de tamaño variable.

Cada opción de administración de *buffers* ofrece ventajas al ruteador pero, como contraparte, también presenta algunos problemas que disminuyen su desempeño. Dichos problemas deben ser abordados con diseños arquitecturales que no incrementen onerosamente su complejidad de implementación, y sí en cambio, mejoren significativamente el desempeño. Por lo tanto, es importante seleccionar un modo de

administración cuya realización sea más sencilla y avocarse a resolver los inconvenientes sacando ventaja de la tecnología disponible e innovando en la arquitectura a fin de lograr la mejor relación costo/desempeño.

Considerando que la administración con *buffers* en las entradas resalta por su sencillez en la implementación del dispositivo de almacenamiento y por su habilidad para tratar con patrones de paquetes de longitud variable, hemos decidido utilizar esta alternativa.

En el proceso de diseño de *buffers* eficientes para *switches* de alto desempeño, varios trabajos [13], [14] han propuesto alternativas que capturan las ventajas de las colas FIFO en la entrada y evitan sus inconvenientes. Una de estas alternativas consiste en colocar múltiples *buffers* FIFO en los puertos de entrada, uno por cada puerto de salida. Cada *buffer* FIFO tiene una línea separada para cada uno de los puertos de salida. Así, el espacio de almacenamiento en cada canal de entrada es estáticamente dividido entre las colas asignadas a los puertos de salida. De esta manera se resuelve el problema de canales de salida ociosos, gracias al desacoplamiento de los puertos de entrada y de salida que se obtiene con esta división de espacio. Los problemas que presenta este diseño son los siguientes: (1) reducción del espacio para un mensaje que arriba a un puerto, debido a la división del *buffer* de entrada; (2) desperdicio de espacio debido a que si los mensajes saturan una partición no podrán utilizar otras particiones, aún cuando estén desocupadas; (3) incremento en la complejidad del *crossbar* debido a que su número de entradas es igual al número de puertos de salida multiplicado por el número de puertos de entrada, haciendo más difícil su control.

Con el fin de simplificar el *crossbar*, existe una propuesta que consiste en implementar todas las particiones de cada puerto de entrada como un solo *buffer* con su espacio dividido en colas separadas, de modo que sólo exista una entrada al *crossbar* por cada puerto de salida [13]. Esto reduce el número de paquetes que pueden leerse de las

colas asociadas con el puerto de entrada, y no reduce la taza a la cual los *buffers* pueden recibir los paquetes, ya que un solo puerto de entrada suministra a todas las colas de su *buffer*. No obstante, la simplificación del *crossbar* no resuelve el problema de manejo ineficiente del espacio de almacenamiento.

Para utilizar eficientemente el *buffer*, en el modelo anterior se asigna el espacio dinámicamente, en lugar de hacerlo estáticamente. Con asignación dinámica, de acuerdo con la demanda, cualquier mensaje que llega a un puerto de entrada puede usar el espacio que requiera, siempre y cuando exista espacio libre disponible dentro del *buffer*. Esto permite aprovechar aquellos espacios ociosos que se generarían con la asignación estática, cuando algunas particiones tienen menos demanda.

1.3.2 Implementación de un Buffer con Asignación Dinámica (*BAC*)

Las ventajas y desventajas que presentan cada uno de los dos modelos anteriores, asignación estática y asignación dinámica, hacen difícil una selección *a priori* de alguno de ellos para su implementación. El modelo de asignación estática implica un diseño menos complejo que el de asignación dinámica, pero no aprovecha eficientemente el espacio de almacenamiento como lo hace el modelo de asignación dinámica. Por lo que, para elegir el mejor modelo tomamos como referencia las pruebas que Tamir y Frazier [13] realizan sobre cuatro opciones con *buffers* en las entradas. Las pruebas se basan en la medición del comportamiento de la latencia contra el caudal de red y demuestran que, de las cuatro opciones que ellos proponen, la de asignación dinámica tiene un mejor desempeño. Sus resultados muestran mejoras en el desempeño que van desde un 5% hasta un 100%, aproximadamente, dependiendo del tamaño de *buffer* y de la frecuencia con la que se reciben los mensajes.

Por otro lado, para comprender mejor el funcionamiento de los modelos de asignación estática y dinámica, nosotros realizamos una simulación de eventos discretos, programada

en los lenguajes *Java* y *ModSim* [86]. Con esta simulación hicimos un análisis comparativo entre asignación estática y dinámica desde el punto de vista del tiempo de saturación del *buffer*. Los resultados obtenidos también mostraron que en la mayoría de los casos la asignación dinámica resulta ser mejor que la estática y en algunos casos son equivalentes. Nuestros resultados de esa simulación muestran mejoras en la latencia de saturación que van desde 4% hasta un 132% con respecto a la asignación estática. Por consiguiente, apoyándose en el criterio de asignación dinámica decidimos desarrollar un *buffer*, al cual hemos denominado *BAC* (*Bufer AutoCompactante*) porque mantiene compactado su espacio ocupado.

Delgado-Frías y otros investigadores han utilizado la técnica de *autocompactación* en el diseño de *buffers* para *switches* [24], [81]. Tales diseños, de la misma forma que nuestro *BAC*, ofrecen alto desempeño porque implementan eficientemente la opción de administración de *buffers* con varias colas asignadas dinámicamente, propuesta en [13]. Los trabajos presentados en [24] y [81] han sido implementados con técnicas de VLSI que permiten optimizar el diseño para alcanzar mayores índices de desempeño. Debido a sus altos costos de diseño e implementación, estas técnicas están orientadas a circuitos integrados de aplicación específica, conocidos como ASIC (*Application Specific Integrated Circuit*), donde los volúmenes de producción son altos. Aún, cuando la flexibilidad de diseño de los ASICs se ha mejorado con técnicas orientadas a celdas (*standard cell*), éstos no son viables comercialmente en aplicaciones de bajo volumen. En estas aplicaciones conviene utilizar dispositivos de lógica configurable, tales como los FPGAs (*Field Programmable Gate Arrays*). La reprogramabilidad de los FPGAs permite una fase de desarrollo más flexible y reduce los riesgos y costos.

Nuestro diseño del *BAC* está descrito el lenguaje VHDL, lo cual facilita su transportabilidad a cualquier tecnología de integración. El desempeño del *BAC* dependerá

de la tecnología utilizada en su implantación, ya sea que su diseño se transporte a un ASIC o se descargue directamente en un FPGA.

Para diseñar el *BAC*, primero propusimos un modelo abstracto basado en cinco operaciones básicas (lecturas, escrituras y combinaciones de éstas en paralelo), y se caracterizaron sus principales propiedades. Enseguida, concretamos el modelo abstracto en un modelo de operaciones más simples llamadas primitivas y se identificaron todos los casos posibles de comportamiento para cada localidad del *BAC*. Posteriormente, sintetizamos el modelo de operaciones primitivas en una tabla que describe el comportamiento completo del *BAC*. Finalmente, con la ayuda de dicha tabla, se implementa el modelo en lenguaje VHDL y se realiza la simulación lógica.

Existen otros trabajos que refieren diseños de memorias autocompactantes descritos en VHDL [62], [78], [79], [80]. Uno de los más recientes se ha desarrollado para la captura de eventos MIDI, donde Peña-Guerrero contempla la ejecución de operaciones de escritura o lectura en forma exclusiva [82]. En ella se identifican sólo 16 casos posibles de comportamiento para cada celda, derivados de estas operaciones simples.

En el caso de nuestro *BAC*, éste lo concebimos para aplicaciones de comunicación en sistemas paralelos, por lo que requiere de un ancho de banda mayor. El *BAC* puede realizar operaciones de escritura o lectura en forma inclusiva. Es decir, además de las operaciones simples de escritura o lectura, permite la ejecución de una escritura y una lectura en forma paralela con direcciones de escritura menores y mayores que las de lectura. Asimismo, el *BAC* permite una escritura y una lectura simultáneas en una misma localidad. Además de los 16 casos derivados de las operaciones simples, el *BAC* contempla otros 22 casos: 10 que se derivan de las operaciones donde la dirección de escritura es menor que la de lectura, 11 derivados de las operaciones con dirección de escritura mayor que la de lectura y un caso para cuando la escritura y la lectura se realizan en la misma localidad.

Es importante destacar que nuestro *BAC* facilita la construcción de *switches* y ruteadores en el nivel de capa física del modelo estándar de comunicaciones OSI (*Open System Interconnection*); por ejemplo, en el diseño se incorpora una señal que indica cuando el *BAC* se satura. Sin embargo, pensamos que dicho *BAC* se encuentra todavía en un nivel más elemental, porque nuestro diseño sólo permite la lectura y escritura de datos, no importando en que contexto se encuentren, o a cuál protocolo OSI pertenecen. Corresponde a los algoritmos del *switch* o ruteador –los cuales quedan fuera del alcance de la presente tesis– la implementación de los protocolos de las capas OSI necesarias para que estos dispositivos operen en un nivel de abstracción acorde con las necesidades de la red de interconexión.

En la Figura 1.1 se muestra la ubicación que tendría nuestro *BAC* en el contexto de un *switch* genérico (no diseñado en esta tesis). El switch contempla cuatro canales de entrada y salida. Nótese que se requiere un *BAC* por cada canal de entrada. Un procesador debe controlar las operaciones de escritura/lectura del *BAC* y, con la ayuda de un *crossbar*, ejecutar los algoritmos de enrutamiento y de control de flujo correspondientes.

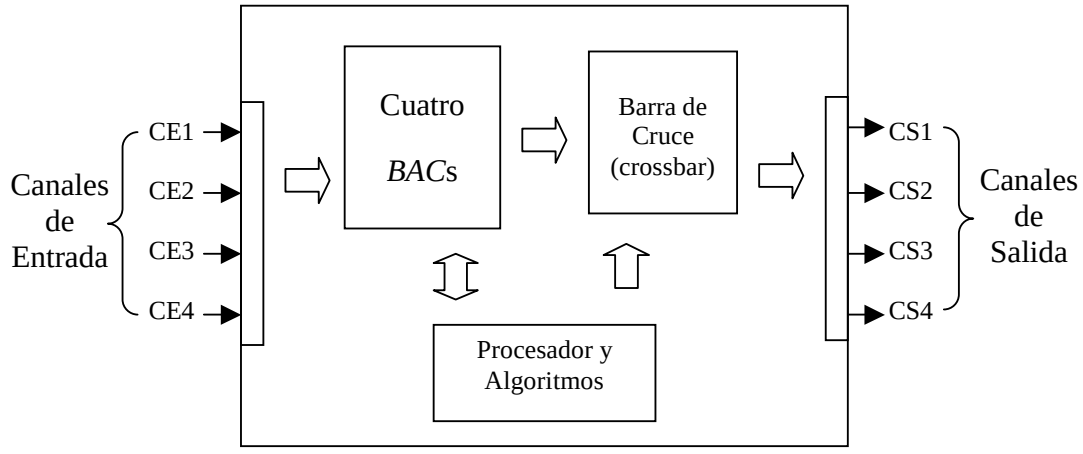


Figura1.1. Diagrama de contexto del *BAC* en un *switch* de cuatro canales de Entrada/Salida

El *BAC* está diseñado totalmente en *hardware*, con un tipo de control distribuido paralelo que asigna una celda de control a cada localidad del *buffer*. Con esta característica se resaltan las propiedades de desempeño del modelo obtenido. Además, la capacidad del *BAC* puede crecer en ancho y largo, sin variar la complejidad de sus celdas de control. Las operaciones de escritura y lectura pueden ejecutarse de manera concurrente durante un solo ciclo de reloj.

1.4 Resultados Obtenidos

Como resultado de esta tesis se obtuvo el diseño específico completo de un modelo de *buffer autocompactante (BAC)* con espacios de almacenamiento independientes asignados dinámicamente y con propiedades de concurrencia en la ejecución de sus operaciones. La simultaneidad de las operaciones de escritura y lectura en un solo ciclo de reloj, equivale a una duración de medio ciclo de reloj por cada operación simple, suponiendo una ejecución secuencial. El tamaño del *buffer* es expandible: puede crecer en número de localidades y cantidad de bits por localidad, manteniendo constantes la complejidad del circuito de control y su desempeño [80]. Su velocidad de operación depende de la tecnología en la que se implante. El diseño del *BAC* fue especificado en VHDL para facilitar la implantación en cualquier tecnología y las actualizaciones de diseño futuras. Las características de alto desempeño que ofrece se derivan del tipo de modelo arquitectural concebido. Tales características pueden resaltarse, aún más, mediante una implantación con tecnología de alta velocidad.

Actualmente, los dispositivos de memoria RAM comerciales con los que pueden construirse *buffers* para *switches* y ruteadores, alcanzan frecuencias de operación cercanas a 700 Mhz. Estas velocidades son menores a las que operan los procesadores recientes (4 GHz, aproximadamente). Sin embargo, pueden superarse mediante la implantación del

BAC en FPGAs con tecnologías de integración que utilizan procesos de 0.18 y 0.13 micras, así como de 90 nanómetros.

La mejora en el desempeño del *BAC* no sólo se debe a que sus ciclos de operaciones simples y concurrentes de lectura y escritura utilizan un período de reloj, hecho que por sí mismo es suficiente si se le compara con los tres o más ciclos de reloj que utiliza una memoria convencional. El verdadero potencial del *BAC* radica en la manera de administrar su espacio de almacenamiento, la cual se implementa completamente en *hardware* para tener mejores tiempos de respuesta. El modelo de administración del *BAC* ahorra varios ciclos de reloj necesarios en la administración por *software* de una memoria convencional, por lo que la aportación más importante del *BAC* está en su diseño arquitectural.

Durante el desarrollo del *BAC* realizamos actividades de verificación y pruebas de funcionamiento, al nivel de simulación de circuitos, que permitieron corroborar las expectativas de nuestro diseño. Con la ayuda del paquete de *software* para diseño digital *Design Environment Foundation*, distribuido por *Xilinx* [83], [84], describimos en VHDL el funcionamiento de los módulos que conforman al *BAC*. Se probó primero el módulo de control para una localidad y, mediante diagramas de tiempo, se verificaron los 38 casos identificados. Enseguida, agregamos los módulos de almacenamiento y direccionamiento, e hicimos pruebas con 8 y 16 localidades.

Diseñamos ejemplos de funcionamiento bajo diversas condiciones en que puede encontrarse el *BAC*. En ellos se aplican operaciones fundamentales simples de escritura o lectura y combinaciones de éstas en forma simultánea. Tales ejemplos contemplan condiciones necesarias y suficientes para crear los 38 casos mencionados. Por lo que, se prueban experimentalmente para demostrar el correcto funcionamiento del *BAC*. En las pruebas se aplican operaciones simples y combinadas, incluyendo las inserción y extracción simultánea de un dato en una misma localidad. Asimismo, aplicamos operaciones de

escritura hasta alcanzar la saturación del *BAC* para observar la aparición de la señal que permitirá inhibir la escritura debido a su saturación.

1.5 Estructura de la Tesis

El contenido de este trabajo está organizado como sigue. El capítulo 2 contiene conceptos fundamentales acerca de la red de interconexión en sistemas paralelos. Aquí se describe la problemática del diseño de redes de interconexión para sistemas paralelos de alto desempeño, haciendo énfasis en las técnicas de administración de *buffers* de mensajes para *switches* y ruteadores. El capítulo 3 contiene el modelado del *BAC*, al nivel de sus 5 operaciones básicas. En él se muestra la estructura general y funcional de nuestro *buffer* mediante un modelo abstracto. El capítulo 4, contiene la descripción del *BAC* a un nivel de operaciones primitivas. Su contenido describe la estructura del *BAC*, incluyendo la parte de control y las acciones que realiza durante la ejecución de las operaciones básicas. El capítulo 5 contiene la implementación del *BAC* en VHDL. En este capítulo se muestran las pruebas realizadas y los resultados obtenidos a nivel de simulación lógica. En el capítulo 6 se presentan las conclusiones de la tesis y los trabajos futuros. En el Apéndice A se presenta una breve descripción de la notación utilizada para describir el modelo del *BAC*. Finalmente, los Apéndices B y C contienen el código VHDL del diseño del *BAC*.

Capítulo 2

Comunicación en Sistemas Paralelos

2.1 Introducción

Las arquitecturas de comunicación para computadoras paralelas de alto desempeño tienen dos facetas: una es la definición de abstracciones críticas, especialmente las fronteras *hardware*/*software* y usuario/sistema en las que se definen las operaciones básicas de comunicación y sincronización y la otra es la estructura organizacional que realiza estas abstracciones para ofrecer alto desempeño [15]. El diseño de dicha estructura organizacional, conocida como *Red de Interconexión*, es crucial en los sistemas paralelos de alto desempeño porque debe reducir los efectos de "cuello de botella", originados por la latencia de comunicación.

Este capítulo contiene definiciones y conceptos generales de redes de interconexión para sistemas paralelos. Asimismo, contiene aspectos básicos que deben considerarse durante el diseño de redes y su relación con el desempeño desde el punto de vista de latencia y ancho de banda. El objetivo es mostrar la importancia de los *switches* y rutedores como componentes clave en el diseño de redes para sistemas paralelos, y el papel

preponderante de sus *buffers* internos, como la parte más crítica para alcanzar un desempeño alto.

En la sección 2.4 se plantea el problema de la administración de *buffers* y se hace un análisis comparativo de las opciones de administración más eficientes. Con base en dicho análisis, justificamos la elección de una opción de administración para usarla como referencia en la propuesta de nuestra solución.

2.2 Red de Interconexión

La red de interconexión de una máquina paralela se encarga de transferir información desde cualquier nodo fuente hasta cualquier nodo destino deseado, soportando las transacciones de red necesarias para realizar el modelo de programación utilizado. Debe permitir un número grande de transferencias, con el objeto de incrementar la concurrencia, y éstas deben realizarse con la menor latencia posible. Además, su costo debe ser relativamente bajo con respecto al resto de la máquina. La Figura 2.1 muestra la red de interconexión de una máquina paralela genérica. Cada nodo contiene un EP, un EM y un EC.

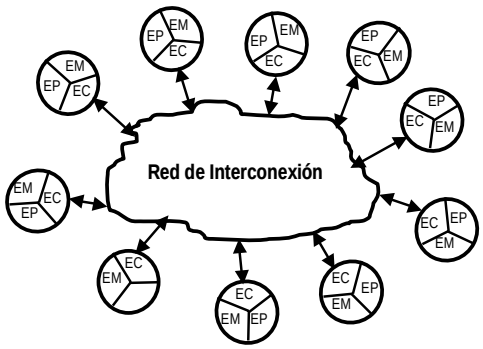


Figura. 2.1 Red de Interconexión de una Máquina Paralela Genérica. (EP: elemento de procesamiento, EM: elemento de memoria, EC: elemento de comunicación)

2.2.1 Taxonomía y Características

De la clasificación de redes [16], mencionada en el capítulo 1: medio compartido, directas, indirectas e híbridas; las ***redes de medio compartido*** tienen en cada nodo, un EC que está constituido por un circuito transmisor, un manejador y un receptor, para realizar el paso de direcciones y datos. Este tipo de red es pasiva: no genera mensajes por sí misma.

Un tema importante, en estas redes, es la estrategia de arbitraje para resolver conflictos de acceso al medio compartido. Su ancho de banda limitado lo hace poco atractivo para sistemas paralelos. Las dos principales clases de redes de medio compartido son: redes de área local, para construir sistemas multicomputador; y buses comunes, para construir sistemas multiprocesador.

Otra clase es la ***red directa***, la cual se modela mediante una gráfica G(N, C), donde los N vértices de la gráfica representan al conjunto de nodos de procesamiento y las líneas C representan al conjunto de canales de comunicación. Los nodos de las redes directas son computadoras, y los ECs son ruteadores asociados a los nodos, encargados de manejar la comunicación de mensajes entre dichos nodos. Cada ruteador tiene conexiones directas a los ruteadores contiguos. Dos nodos contiguos se conectan por medio de un par de canales unidireccionales en direcciones opuestas. Aunque cada procesador local puede realizar la función de enrutamiento, es necesario utilizar ruteadores para buscar el alto desempeño, por medio de la concurrencia de operaciones de cálculo y comunicación. Conforme se incrementa el número de nodos en el sistema, dentro de cierto límite, también se incrementan el ancho de banda de comunicación total, el ancho de banda de memoria, y la capacidad de procesamiento. Este tipo de red permite construir máquinas paralelas de gran escala.

Tenemos también la clase de ***redes indirectas***, cuyos ECs son adaptadores de red que se conectan a un *switch* de red. La red de interconexión está compuesta por un arreglo de *switches*. Algunos puertos de cada *switch* pueden estar conectados a procesadores o estar sin conexión, mientras que los demás puertos son conectados a otros *switches* para realizar la conectividad entre procesadores. La forma de interconectar los *switches* define varias topologías de red [38], que van desde las topologías regulares, usadas en multiprocesadores de memoria compartida, hasta las irregulares, actualmente usadas en redes de estaciones de

trabajo (NOWs) [46], [47]. Las topologías regulares tienen patrones de conexión regulares entre los *switches*. Las topologías irregulares no siguen ningún patrón predefinido. Por otra parte, las ***redes híbridas*** combinan mecanismos de redes de medio compartido con mecanismos de redes directas o indirectas.

2.2.2 Aspectos de Diseño

Los principales aspectos de diseño que caracterizan a las redes de interconexión son: topología, algoritmo de enrutamiento, mecanismo de control de flujo y estrategia de conmutación [34], [15], [58], [59]. El desempeño de las redes está estrechamente relacionado con la forma en que interactúan estos cuatro aspectos.

La ***topología*** es la estructura que define la manera de interconectar los nodos por medio de canales y *switches* o ruteadores. La mayoría de las máquinas paralelas emplean redes con topología ortogonal. Una topología de red es ortogonal cuando los nodos pueden ser arreglados en un espacio ortogonal de una o más dimensiones. Las redes directas tienen un EP conectado en cada ruteador, mientras que las indirectas tienen EPs conectados solamente a un subconjunto específico de *switches*. Muchas máquinas emplean estrategias mixtas con los dos tipos de nodos, de modo que los nodos servidores generan y remueven tráfico, mientras que los *switches* solamente mueven el tráfico a través de ellos.

El ***algoritmo de enrutamiento*** determina las rutas de los mensajes que fluyen a través de la red. En cada nodo intermedio, el algoritmo de enrutamiento selecciona, entre varios candidatos, el próximo canal a utilizar. Si todos los canales candidato están ocupados, el paquete es bloqueado y no avanza. El algoritmo de enrutamiento restringe el conjunto de trayectorias posibles al conjunto más pequeño de trayectorias permitidas. Existen muchos algoritmos de enrutamiento con diferentes características de seguridad y niveles de desempeño [41], [42], [43], [48], [57], [66]. Las rutas de interés en una máquina paralela son aquellas que van de un nodo de procesamiento a otro.

El mecanismo de ***control de flujo*** determina cuándo los mensajes, o porciones de ellos, se mueven a lo largo de la ruta. En particular, el control de flujo es necesario cuando dos o más mensajes intentan usar el mismo recurso de red al mismo tiempo. Un flujo de tráfico puede colocarse en su sitio, desviarse hacia los *buffers*, desviarse hacia una ruta alterna, o simplemente descargarse. Cada una de estas opciones implica requerimientos específicos en el diseño del *switch* e influye en otros aspectos del subsistema de comunicación [49]. La unidad mínima de información transferible sobre un enlace, que puede aceptarse o rechazarse, se llama unidad de control de flujo, o *flit*. La unidad de información que puede transferirse a través de un canal físico en un solo paso o ciclo se llama *phit*. La unidad de control puede ser tan pequeña como un *phit* o tan grande como un paquete o un mensaje.

La ***estrategia de conmutación*** determina cómo enviar los mensajes a través de su ruta. Cuando un encabezado de mensaje o paquete llega a un nodo intermedio, la estrategia de conmutación determina cómo y cuándo debe ser configurado el *switch* o el ruteador, definiendo la asignación de recursos de red para transmitir el mensaje o paquete.

Las estrategias básicas de conmutación son: de circuito, de paquete, de corte virtual a través de (*virtual cut-through*), y de túnel (*wormhole*).

Cuando se utiliza *conmutación de circuito*, se reserva una trayectoria física desde la fuente hasta el destino antes de transmitir los datos. El enlace se inicia inyectando un *flit* de encabezado de enrutamiento hacia la red y se completa cuando alcanza el destino, con un reconocimiento hacia la fuente. El mensaje se transmite después de haber reservado el circuito. La trayectoria se mantiene ocupada durante toda la sesión de comunicación. Esto implica desperdiciar los períodos durante los cuales las partes están en silencio.

En *conmutación de paquete* el mensaje se divide en fracciones llamadas paquetes, las cuales se envían en secuencia. Cada paquete se almacena completamente, en cada nodo intermedio de su trayectoria, antes de enviarse al próximo nodo. Un *paquete* es una serie

auto-delimitada de símbolos lógicos que consta de tres partes: encabezado, cuerpo principal, y cola. En la Figura 2.2 se muestra el formato típico de un paquete. Este formato es a nivel de la capa de enlace de datos. A nivel de red no tiene cola.

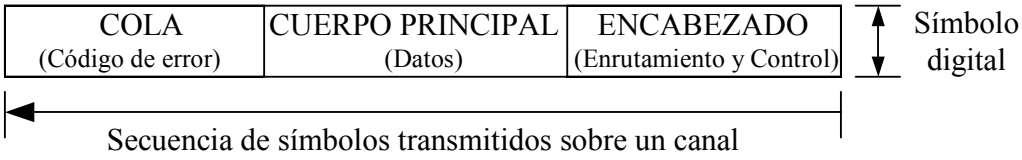


Figura 2.2 Formato de un paquete típico.

El paquete contiene información de enrutamiento y secuenciamiento, así como datos. La cola, típicamente contiene el código de chequeo de error. Los paquetes se enrutan individualmente desde el nodo fuente hasta el destino. En conmutación de paquetes, cada paquete transporta su propia información de dirección. Esto introduce un desperdicio, debido a la redundancia de información; no obstante, mediante la división de mensajes en paquetes, un mensaje puede usar varios enlaces simultáneamente en una trayectoria. De esta forma se comparten más eficientemente los enlaces de comunicación, se logra un mayor uso del canal y se obtiene un menor retardo de red [15]. La conmutación de paquetes permite una mejor utilización de los recursos de red, con respecto a la conmutación de circuito, debido a que los enlaces y los *buffers* de *switches* y ruteadores se ocupan solamente mientras un paquete pasa por ellos. La latencia que experimenta un paquete es proporcional a la distancia entre los nodos fuente y destino. La estrategia de conmutación de paquetes adquiere mayor ventaja cuando los mensajes son cortos y frecuentes. Los enlaces se comparte estadísticamente con muchos nodos, debido a que no se asignan a trayectorias completas de pares de nodos específicos fuente-destino.

Con la estrategia de conmutación de paquete existe un retardo innecesario, debido a que un paquete no se transmite antes de recibirse completamente. Sin embargo, cuando un paquete llega a un nodo intermedio y su canal de salida está libre, no necesita almacenarse completamente en el nodo antes de enviarlo a la salida. Así, puede evitarse un retardo de almacenamiento innecesario. El paquete puede transmitirse inmediatamente después de recibir el encabezado e identificar el canal de salida mediante la función de ruteo. Se requiere espacio de *buffer* solamente cuando se encuentra el canal ocupado. Esto es esencialmente una combinación de técnicas de conmutación de circuito y conmutación de paquete, la cual se conoce como *corte virtual a través de* [17]. Si los paquetes encuentran canales ocupados en todos los nodos intermedios, la salida es exactamente la misma como en la red de paquete conmutado. Por otra parte, si todos los canales intermedios están libres, la salida es similar al sistema de circuito conmutado. En la Figura 2.3 se muestra la secuencia de envío de paquetes con las técnicas almacena-y-envía y *cut-trhough*.

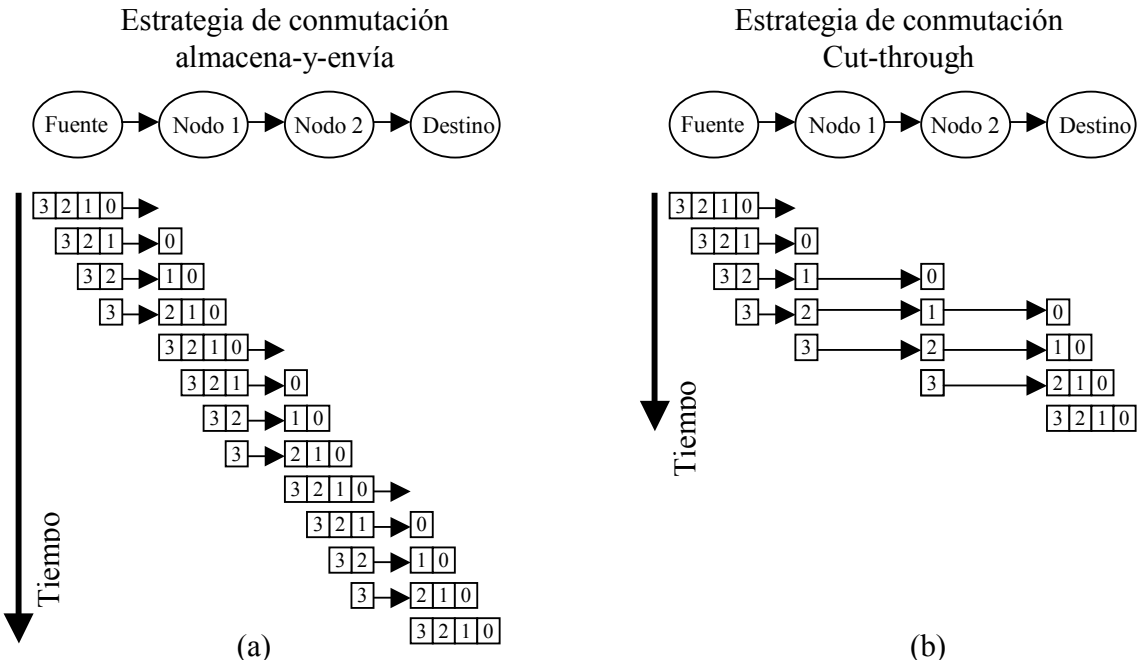


Figura 2.3 Estrategia almacena-y-envía contra cut-through para redes de paquete conmutado

Con la estrategia *de túnel*, los paquetes se envían a través de la red, de modo similar a una "línea de tubería" (*pipeline*) [18]. Los paquetes se dividen en pequeñas unidades que se envían en secuencia. Si la primera unidad no puede avanzar, se le pide al origen que deje de transmitir, y así el paquete podría quedar tendido a lo largo de dos o más ruteadores o *switches*. Una vez que se liberan los recursos, el paquete continúa. La ventaja de esta estrategia de conmutación es que reduce los requerimientos de almacenamiento y disminuye la latencia en cada nodo [55]. Sin embargo, su principal desventaja es que, cuando se incrementa el tráfico, la red se hace más susceptible a bloqueos [42].

2.2.3 Desempeño

La forma en que interactúan los cuatro aspectos de diseño de redes, de la sección anterior, determinan ampliamente su desempeño y funcionalidad. Dichos aspectos pueden ser vistos desde las perspectivas de latencia y ancho de banda.

Latencia

Si se observa el desempeño desde la perspectiva de la *latencia de comunicación*: tiempo para transferir *n* bytes de información desde su fuente (*S*) hasta su destino (*D*), se pueden identificar los siguientes componentes:

$$Latencia\ de\ Comunicación(n)_{S\text{-}D} = Latencia\ de\ Arranque + Latencia\ de\ Red \qquad (1)$$

La *Latencia de Arranque* es el tiempo que el procesador utiliza para iniciar la transferencia. En este tiempo, el procesador está ocupado con el evento de comunicación y no puede realizar otro trabajo útil o iniciar otra comunicación. Normalmente en este tiempo se realiza el entramado/desentramado, el copiado memoria/*buffer* y la validación de mensajes, entre otras operaciones. La latencia de arranque depende del diseño de *software* del sistema, dentro de los nodos y de la interface entre nodos y ruteadores. Su valor es fijo cuando el procesador simplemente llama al asistente de comunicación para iniciar, o puede ser lineal en *n*, si el procesador tiene que copiar el dato dentro del asistente.

La *Latencia de Red, LR*, es el tiempo que transcurre desde que la fuente introduce el encabezado de un mensaje a la red, hasta que la cola del mensaje llega a su destino. Idealmente, en los sistemas paralelos de alto desempeño, debería ser posible ocultar este componente de tiempo con otras operaciones del procesador [52].

Los componentes básicos de la Latencia de Red son los siguientes:

LR = Ocupación del Canal + Retardo de Enrutamiento + Retardo de Contienda *(2)*

La *Ocupación del Canal (OC),* es el tiempo que le toma al dato pasar a través del componente más lento sobre la trayectoria de comunicación. Por ejemplo, cada enlace por el que pasa un paquete, se mantiene ocupado por un tiempo *n*/B, donde B es el ancho de banda del enlace. Los datos ocupan varios recursos del elemento de comunicación, tales como: *buffers*, *crossbar*s, e interfaces de comunicación. El recurso más lento es el "cuello de botella" que determina la ocupación. La ocupación limita que tan frecuentemente pueden iniciarse las operaciones de comunicación. Una próxima transferencia de dato debe esperar hasta que un recurso crítico deje de estar ocupado, antes de poder usar el mismo recurso. Si existe almacenamiento ("buffereo") entre el procesador y el elemento de comunicación, el procesador puede realizar sus transferencias a una frecuencia mayor que 1/*OC*. Cuando el *buffer* está lleno, el procesador reduce sus transferencias a la razón fijada por la ocupación. Una transferencia nueva puede iniciar solamente cuando una anterior ha terminado. La ocupación de cada enlace está influida por el ancho del canal, la razón de señalización, y la cantidad de información de control, la cual depende de la topología y el algoritmo de enrutamiento.

El *Retardo de Enrutamiento* (RE), es el tiempo utilizado para mover el primer bit del mensaje, desde en nodo fuente hasta el destino. Cada paso a lo largo de la ruta incurre en un retardo parcial de enrutamiento que se acumula en el retardo global observado desde afuera de la red. El retardo de enrutamiento es una función del número de canales sobre la ruta,

llamado distancia de enrutamiento, *h*, y del retardo Δ incurrido en cada *switch*, como parte de la función de selección del puerto de salida correcto. La distancia de enrutamiento depende de la topología de red, el algoritmo de enrutamiento, y el par particular de nodos fuente y destino. El retardo Δ depende del tamaño del *switch*, de la arquitectura de sus *buffers* y su forma de administrarlos.

Debido a que muchas transferencias pueden ocurrir simultáneamente, la latencia de red presenta un factor adicional, llamado *Retardo de Contienda* ("*contention delay*"). Si dos de estas transferencias intentan usar el mismo recurso a la vez, una debe esperar. Esta contienda de recursos incrementa el tiempo de comunicación promedio. Desde el punto de vista del procesador, la contienda aparece como ocupación incrementada. Todos los retardos son fuertemente afectados por las estrategias de conmutación y enrutamiento.

Bajo una estrategia de conmutación de circuito, el retardo es proporcional a la distancia de enrutamiento, *h*. En este lapso de tiempo se establece el circuito, se configura cada *switch* de la ruta, y se informa a la fuente cuando se establece la ruta. Después del retardo, el dato se mueve a lo largo del circuito, en un tiempo *n/b* mas un pequeño retardo adicional, proporcional a *h*. Así, para un mensaje de *n*-bytes, a través de una distancia *h*, la latencia sin carga (en ausencia de mensajes), es: $T_{cc}(n, h) = n/b + h\Delta$. Las unidades de tiempo de $T_{cc}(n, h)$ son los ciclos de red, τ. En esta ecuación, los retardos de arranque y de enrutamiento son términos adicionales, independientes del tamaño del mensaje. Por lo tanto, conforme se incrementa la longitud del mensaje, la distancia de enrutamiento se convierte en una fracción insignificante de la latencia de comunicación sin carga.

En conmutación de paquete, un mensaje largo puede fragmentarse en varios paquetes pequeños, para que fluyan a través de la red en forma de pipeline, tal como se muestra en la Figura 2.3(a). En este caso, la latencia sin carga es $T_{cp}(n, h, n_p) = (n-n_p)/b + h(n_p/b + \Delta)$,

donde n_p es el tamaño de los fragmentos. El retardo de enrutamiento efectivo es proporcional al tamaño del paquete en lugar del tamaño del mensaje.

Muchas máquinas paralelas usan conmutación de paquetes con enrutamiento de "corte virtual a través de". En ellas, los *switches* realizan la decisión de enrutamiento después de interpretar solamente los primeros *phits* del encabezado. También, permiten que el que envía los paquetes "corte a través" del canal de entrada al canal de salida, como se indica en la Figura 2.3(b), tal que la transmisión de un paquete se realiza en modo "pipeline". Para enrutamiento de "corte virtual a través de", la latencia sin carga está dada por $T_{cvt}(n, h) = n/b + h\Delta$. Esta expresión tiene una forma similar al caso de conmutación de circuito, aunque el coeficiente de enrutamiento, Δ, puede diferir debido a que los mecanismos del proceso son diferentes. Obsérvese que con una técnica de "corte virtual a través de", un solo mensaje puede ocupar toda la ruta desde la fuente hasta el destino, muy similar a la conmutación de circuito. El encabezado del mensaje establece la ruta por la que se mueve hacia su destino, y la ruta se limpia conforme la cola se mueve a través de ella.

Con la estrategia de conmutación "de tunel" se almacenan solamente unos pocos *flits* en el *switch* y se deja la cola del mensaje a lo largo de la ruta. La porción bloqueada del mensaje es muy similar a un circuito que se mantiene abierto a través de una porción de la red.

Los *switches* tienen limitados sus *buffers* para paquetes, tal que bajo una contienda sostenida los *buffers* del *switch* pueden llenarse. ¿Qué sucede con los paquetes que llegan al *switch* cuando no hay espacio en el *buffer*? En redes de comunicación de datos tradicionales, los enlaces son largos y ocurre una pequeña retroalimentación entre las dos terminales del canal, tal que el enfoque típico es descartar el paquete. Así, bajo contienda, la red se hace altamente inestable, y se utilizan protocolos sofisticados (como *TCP/IP*, de

lento arranque) que adaptan la carga de comunicación requerida para que la red tenga una razón de pérdida baja.

En redes de computadoras paralelas, el encabezado de un paquete que llega a un *buffer* lleno, típicamente se bloquea en el sitio, en lugar de descartarse. Esto requiere un protocolo de control de flujo, al nivel de enlace, entre los puertos de entrada y salida. Bajo congestión sostenida, el tráfico se respalda desde el punto de la red donde ocurre la contienda. Eventualmente, las fuentes experimentan un rechazo de paquetes, que hace que el flujo de datos disminuya lentamente hasta convertirse en un "cuello de botella". Incrementos en la cantidad de almacenamiento de la red permiten una mayor contienda, al reducirse el rechazo de paquetes, pero aumentan el retardo de encolamiento dentro de la red cuando ocurre la contienda. Por lo que es importante considerar no solamente la cantidad de almacenamiento, sino también otros aspectos de diseño de la red, tales como, los enlaces y el ancho de banda, además de los ya vistos. Tales aspectos de diseño se combinan para determinar la latencia del mensaje.

Ancho de Banda

El ancho de banda de la red es crítico para el desempeño de un programa paralelo porque al incrementarse el ancho de banda se reducen la ocupación y la probabilidad de contienda. Además, en ancho de banda se vuelve crítico debido a que las fases de un programa pueden enviar un gran volúmen de datos a su alrededor, sin esperar que la transmisión de datos individuales sea completada.

Si el volúmen de comunicación total de un programa es M bytes y el ancho de banda de la red es B bytes por segundo, el tiempo de comunicación será de al menos M/B segundos. Por otro lado, si toda la comunicación es hacia o desde un solo nodo, esta estimación está lejos de ser correcta; el tiempo de comunicación podría ser determinado por el ancho de banda a través de ese solo nodo. Por tal motivo, es importante analizar las

opciones de diseño del EC en el nodo para utilizar aquellas que tengan mejor incidencia en el ancho de banda.

La envolvente del paquete incrementa la ocupación debido a que la fuente agrega símbolos de encabezado y cola, y el destino los retira. Si b es el ancho de banda bruto de un canal, n el tamaño de la carga útil, y n_E el tamaño de la envolvente, entonces el ancho efectivo del canal se ve disminuido por un factor $n/(n+n_E)$, para paquetes de tamaño fijo, y la ocupación del canal es $(n+n_E)/b$.

El ancho de banda local efectivo del enlace se reduce por la densidad de los paquetes y se expresa, a partir del ancho de banda bruto, como $b(n/(n+n_E))$. Además, si el *switch* bloquea el paquete durante Δ ciclos mientras hace su decisión de enrutamiento, entonces el ancho de banda local efectivo es $b(n/(n + n_E + w\Delta))$, donde $w\Delta$ es la oportunidad para transmitir datos que se pierden mientras que el enlace está bloqueado. Así, problemas de diseño de la red tales como el formato del paquete y el algoritmo de enrutamiento influirán en el ancho de banda visto por incluso un solo nodo. Si múltiples nodos se están comunicando a la vez y se eleva la contienda, el ancho de banda local percibido disminuirá y la latencia se elevará. La selección de la topología de red y el algoritmo de enrutamiento afectan la probabilidad de contienda dentro de la red.

Actualmente, el diseño de una red de interconexión para sistemas paralelos se reduce al diseño del EC, que consta fundamentalmente de un *switch*, para redes indirectas, o un ruteador, para redes directas.

2.3 *Switches* y Ruteadores

La estructura general del *ruteador* es esencialmente un *switch*, y su principal diferencia está en el canal interno bidireccional, que se agrega al ruteador, para generar mensajes hacia la red y extraerlos de ella. En la Figura 2.4 se muestra la arquitectura de un nodo

genérico típico de las redes directas o basadas en ruteador [37], [43]. Cada ruteador está asociado a un EP mediante un canal interno bidireccional. Por lo que, todos los ruteadores en la red pueden ser nodos fuente o destino.

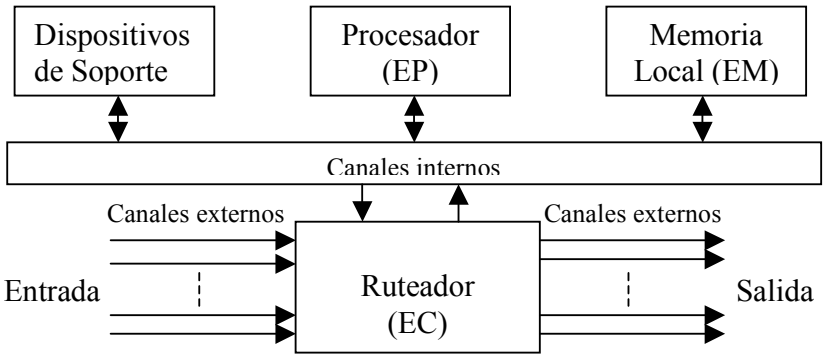


Figura 2.4. Arquitectura de un nodo genérico.

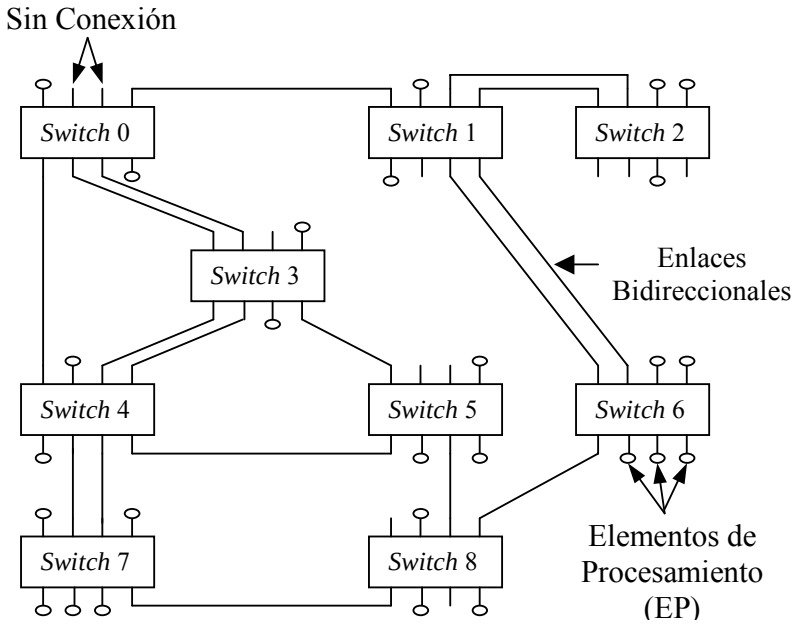


Figura 2.5 Ejemplo de una red basada en *switch* con topología irregular.

La Figura 2.5 muestra un ejemplo de red indirecta o basada en *switch*. Cada *switch* puede estar conectado a cero, uno, o más EPs. Es posible que algunos *switches* no estén

asociados a ningún EP, por lo que, no corresponden a nodos fuente o destino. En ambos casos el problema fundamental se reduce al diseño del *switch*.

2.3.1 Tamaño o Grado

Típicamente, el número de puertos de entrada del *switch* es igual al número de puertos de salida, y se conoce como *grado, n,* del *switch*. Cada puerto de salida contiene un transmisor para el manejo del enlace. Generalmente, cada puerto de entrada contiene un sincronizador que alinea los datos entrantes con el dominio del reloj local del *switch*. Como se mencionó en el capítulo anterior, los *switches* prevén cierta capacidad de almacenamiento temporal, la cual puede estar compartida por todo el *switch* o puede estar dividida y asignada a cada puerto de entrada o salida. La complejidad de la lógica de control necesaria para administrar los *buffers* depende del tipo de enrutamiento y control de flujo [67], [69], [71].

En algunos casos, el grado del *switch* puede usarse como una métrica del costo de red. El costo de algunas partes del *switch* - transmisores, receptores, y latches de puertos - es lineal con *n*. Sin embargo, el costo de la conexión interna: *crossbar* y *buffers*, puede incrementarse de manera no lineal, con n^2. En *switches* VLSI recientes, la principal restricción es el número de pines, el cual es esencialmente igual al número total de puertos de entrada y puertos de salida por el ancho del canal. Los diseños de *switches* con canales angostos de alta frecuencia, satisfacen la restricción del número de pines pero elevan la complejidad de la codificación de las señales de control dentro del flujo serial de bits. Por lo que, los *switches* pequeños de *n* x *n* son una alternativa clave en el diseño de redes de interconexión para sistemas paralelos. La arquitectura de tales *switches*, particularmente sus *buffers* internos, es crítica para alcanzar una comunicación de baja latencia y alto caudal bajo una implantación de bajo costo efectivo [13], [19], [65].

Los sistemas multiprocesadores con un número grande de EPs (ej., mayor de 64) usan redes de interconexión indirectas compuestas de *switches* pequeños *n* x *n* (típicamente, de 2 ≤ *n* ≤ 10) para comunicación [10], [20]. Similarmente, la comunicación a través de enlaces dedicados punto-a-punto en multicomputadoras [21], [22] se realiza sobre ruteadores con un número pequeño de puertos [10], [23] que funcionan como pequeños *switches n* x *n*, con *n* - 1 puertos conectados a otros nodos, y un puerto bidireccional conectado al procesador local. Por lo tanto, el diseño de pequeños *switches n* x *n* de alto desempeño es de importancia crítica para el éxito de los sistemas multiprocesador y multicomputador [24], [81], [45].

2.3.2 Arquitectura

Chien [25] desarrolló un modelo abstracto para la arquitectura de un *ruteador genérico*. Este modelo se concentra en la arquitectura interna del ruteador, cuyo desempeño depende principalmente de la opción de administración de sus *buffers* y de la algorítmica con que se implementa ésta. El desempeño externo de las operaciones de enlace entre ruteadores es muy sensitivo a la tecnología de implementación. Las funciones básicas del ruteador se obtienen de la arquitectura mostrada en la Figura 2.6.

El *Crossbar* es responsable de conectar los *buffers* de entrada del ruteador a los *buffers* de salida. Los ruteadores de alta velocidad utilizaran redes de *crossbar* con conectividad completa, mientras que las implementaciones de baja velocidad pueden utilizar redes que no proveen completa conectividad entre los *buffers* de entrada y los de salida [16], [51].

El *Controlador de Enlace* (LC) coordina la transferencia de las unidades de control de flujo. En el lado de la recepción debe considerarse suficiente espacio de *buffer* para tomar en cuenta los retardos en la propagación de los datos y señales de control de flujo. Si hay

canales virtuales, el controlador también es responsable de decodificar el canal destino del *phit* recibido [64].

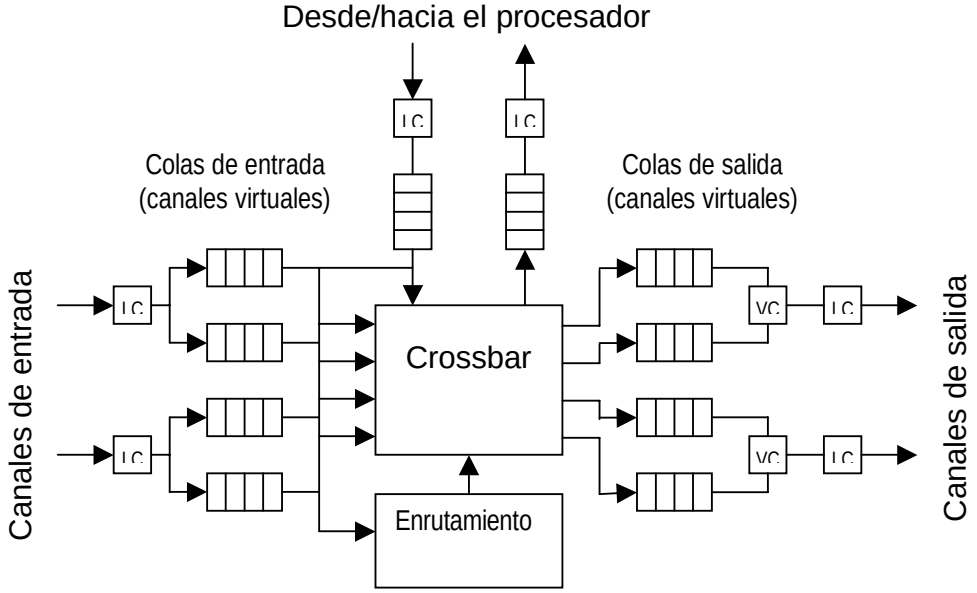


Figura 2.6. Arquitectura del Ruteador Genérico.

El *Controlador de canal virtual* (VC), es responsable de multiplexar los contenidos de los canales virtuales sobre los canales físicos.

El módulo de *Enrutamiento y Arbitraje* implementa la función que determina la ruta del mensaje y arbitra los canales del *crossbar*. Las políticas de arbitraje rápido son cruciales para mantener una latencia de control de flujo baja a través del *switch*.

Los *Buffers* son de tipo FIFO y se encargan de almacenar los mensajes en tránsito. En el modelo anterior, un *buffer* está asociado con los canales físicos de entrada y los canales físicos de salida. El tamaño del *buffer* es un número entero de unidades de control de flujo. En diseños alternativos, los *buffers* pueden ser asociados solamente con entradas (*bufereo* de entrada) o salidas (*bufereo* de salida). En conmutación VCT hay suficiente espacio

disponible para un paquete de mensajes completo. Para un tamaño de *buffer* fijo, la inserción y extracción del *buffer* no está usualmente en la trayectoria crítica del ruteador.

La *Interfaz del procesador* se encarga de interconectar el canal físico con el procesador, en lugar de un ruteador adyacente. Consiste de uno o más canales de inyección desde el procesador y uno o más canales de extracción hacia el procesador.

En la Figura 2.7 se muestra un modelo genérico para un *switch* de grado n = 4. Este modelo es una abstracción del ruteador genérico de la Figura 2.6. y es utilizado para analizar las funciones básicas. El modelo no considera el uso de canales virtuales, por lo que no muestra los controladores respectivos. Tampoco, considera los controladores de enlace en los canales de entrada y salida. Solamente muestra la parte fundamental de nuestro interés, que es el *switch*, compuesto básicamente por: *buffers* en las entradas y salidas, un *crossbar*, y las unidades de enrutamiento y de control de flujo.

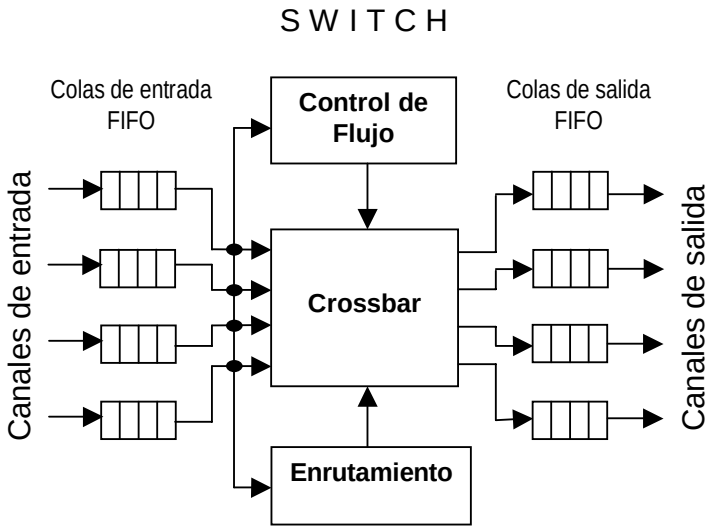


Figura 2.7. Arquitectura de un Switch Genérico.

2.4 Problemas en la Administración de Buffers

El desempeño de los *switches* depende significativamente de la organización de sus *buffers* de almacenamiento. Ruteadores y *switches* tradicionales tienden a tener *buffers* externos: SRAM y DRAM grandes. En *switches* VLSI, el almacenamiento es interno y se encuentra integrado, en el mismo silicio, con la ruta de datos y la sección de control.

El problema de diseño de los *buffers* no sólo radica en la búsqueda del uso eficiente del espacio de almacenamiento. El diseño de los *buffers* tiene como principal objetivo prever una mínima afectación en la utilización de otros componentes de la red. En este sentido, es prioritario buscar un diseño arquitectural que ayude a resolver los conflictos que ocasionan bloqueos inecesarios de canales. Al mismo tiempo que, ayude a resolver el problema de almacenamiento y control de flujo de mensajes, en el menor número posible de ciclos de reloj del sistema [77], [79].

En esta sección presentamos un análisis de las formas de administración de *buffers* para *switches* y ruteadores y hacemos énfasis en las arquitecturas que utilizan *buffers* en las entradas, por ser las que ofrecen mayores ventajas.

2.4.1 Opciones de Administración

Son tres las opciones básicas de administración de *buffers* en *switches* y ruteadores. Estas opciones, que ya se han mencionado en el primer capítulo, utilizan un grupo de *buffers* compartidos centralizados, o *buffers* en las entradas, o *buffers* en las salidas.

Administración de Buffers Compartidos Centralizados

En un grupo de *buffers* compartidos centralizados, cada puerto de entrada deposita datos en una memoria central, y cada puerto de salida los lee. Con esta opción se usa más eficientemente el espacio *buffer* y se evita el bloqueo de cabeza-de-línea, debido a que los puertos de entrada escriben en el grupo sin importar el puerto de salida, suponiendo que el espacio está disponible. Sin embargo, existen dificultades importantes en la implementación de esta opción. Para alcanzar un alto desempeño, debe disponerse de puertos de comunicación con múltiple ancho de banda para acceder a los *buffers* centrales simultáneamente. En el peor de los casos, el ancho de banda de la interconexión, entre el conjunto de *buffers* y los puertos, debe ser igual a la suma de los anchos de banda de todos los puertos. Además, para un *switch* n x n, los *buffers* deben tener 2 x n puertos con el fin

de soportar *n* lecturas simultáneas con *n* escrituras simultáneas. La memoria multipuerto no es recomendable porque su implementación es cara, en área, y conlleva un desempeño bajo debido a tiempos de acceso grandes..

Una segunda alternativa de diseño de *buffers* centrales compartidos consiste en alcanzar el ancho de banda requerido sin usar memoria multipuertos, y haciendo la memoria del *buffer*, y su bus, lo suficientemente anchos [23]. Sin embargo, la necesidad de "ensamblar" bytes en palabras anchas antes de que la transmisión incremente la latencia de comunicación (especialmente en redes cargadas fuertemente) provoca desperdicio de espacio *buffer* y hace que se pierda algo del ancho de banda disponible cuando se transmitan bloque más pequeños que el ancho del bus. Con paquetes de longitud variable, es difícil implementar *hardware* de control que asigne memoria rápidamente (en 1 o 2 ciclos) y minimice la fragmentación interna o externa.

Otra alternativa, propuesta en [26], consiste en tratar a todo el espacio del *buffer* central compartido como una memoria única, paginada. Esta alternativa permite que todos los canales compartan la demanda de un único espacio paginado. Para cada canal, se crea un *buffer* virtual dedicado que puede variar su tamaño de acuerdo con la demanda y con la disponibilidad de espacio. Se puede asignar a cada canal un espacio tan pequeño como una página ó tan grande como la totalidad de páginas disponibles. Los inconvenientes en los tiempos de respuesta pueden disminuirse manejando cada *buffer* virtual como un arreglo circular de páginas. Esta alternativa es muy útil en *switches* o ruteadores donde es más importante el uso eficiente del espacio de almacenamiento que el desempeño. Es decir, en sistemas que utilizan redes convencionales. Su implementación presenta problemas similares a las alternativas de diseño anteriores.

Además de las dificultades de implementación, los *buffers* centrales compartidos también pueden tener problemas de desempeño. Con compartición completa, un solo

puerto de salida congestionado puede acaparar la mayoría del espacio de almacenamiento e impedir la comunicación a través del conmutador [27], [28]. La necesidad de limitar el espacio máximo de almacenamiento a cada puerto previene el uso del *buffer* completo por una sola trayectoria, siempre y cuando se incremente el desempeño, pero incrementa la complejidad de la implementación [27], [28]. Estas consideraciones hacen más atractivas las opciones de diseño de *switches* con *buffers* independientes asociados a los puertos de entrada o de salida.

Administración de Buffers en las Salidas

Con el uso de *buffers* en los puertos de entrada o salida se desacoplan los *switches* en ambas terminales del enlace y tienden a proveer una mejora significante en el desempeño. Conforme el tamaño y densidad del chip se incrementen, se dispone de mayor capacidad de *buffer*, y el diseñador de la red tiene más opciones de diseño. Se ha mostrado que con el uso de *buffers* FIFO, en los puertos de salida, la longitud de la cola principal es más corta, que con *buffers* en los puertos de entrada [29]. Esto implica que un *switch* con *buffers* de entrada requiere *buffers* más largos para alcanzar la misma probabilidad de sobreflujo de *buffer*. La causa se debe a que, con *buffers* en los puertos de entrada, los paquetes destinados a puertos de salida desocupados pueden encolarse después de paquetes cuyo puerto de salida está ocupado, por lo que no se pueden transmitir. El problema con la implementación de *buffers* de puertos de salida es que, con el fin de que sean capaces de manejar llegadas simultáneas de paquetes, los *buffers* deben tener tantos puertos de escritura como puertos de entrada del *switch*. La implementación de los *buffers* con múltiples puertos de escritura incrementa su tamaño y reduce su desempeño, Además, si ocurre más de una escritura a la vez, hay un problema de asignación eficiente de los recursos de *buffer* para paquetes de tamaño variable.

Un diseño con *buffers* en puertos de salida debe proveer una forma de aceptar múltiples paquetes, en cada entrada, como candidatos para avanzar hacia los puertos de salida. Una opción natural para lograrlo consiste en proveer un espacio-*buffer* independiente, para cada puerto de salida, tal que los paquetes se ordenen por destino desde que se reciben. Con un flujo sostenido de tráfico sobre las entradas, las salidas pueden ser manejadas, en principio, al 100%. La desventaja de dicha opción de diseño es que requiere almacenamiento de *buffer* e interconexión interna adicionales. Además, se requiere una etapa de ordenamiento y multiplexores más anchos, lo cual incrementa el tiempo de ciclo del *switch* o el retardo de enrutamiento. Cada puerto de salida tiene el ancho de banda suficiente para recibir un paquete desde cada puerto de entrada en un ciclo. Esta propiedad se puede obtener colocando, en cada salida, un *buffer* FIFO con una velocidad interna de operación de *n* veces la de los puertos de entrada.

Administración de Buffers en las Entradas

Para la opción que utiliza *buffers* independientes en las entradas, Tamir y Frazier [13] proponen cuatro tipos de arquitecturas de *switch*, basados en la forma en que se manejan las colas y se almacenas los datos. En la sección siguiente se presentan las principales características de estos cuatro tipos de arquitectura.

2.4.2 Arquitecturas con Buffers en las Entradas

El primer tipo utiliza un *buffer* FIFO por cada canal de entrada, tal como se ilustra en la Figura 2.8. En esta arquitectura, solamente un paquete a la vez llega a cada puerto de entrada, por lo que el *buffer* solo requiere de un puerto de escritura. Si el *buffer* es manejado como una cola FIFO, es muy fácil tratar con paquetes de longitud variable y evitar problemas de asignación de memoria [10], [11], [61], [63]. Cada *buffer* debe estar habilitado para aceptar un *phit* en cada ciclo de operación y entregar un *phit* a la salida. Es necesario que el ancho de banda interno del *switch* sea acorde con el flujo de datos

simultáneos que llega a los canales de entrada. La operación del *switch* consiste en monitorear el encabezado de cada entrada FIFO, calcular el puerto de salida requerido, y programar los paquetes para moverlos coordinadamente a través del *crossbar*. La lógica de enrutamiento está asociada con cada puerto de entrada para determinar la salida requerida. Con enrutamiento *cut-through*, la lógica de decisión hace una selección independiente por cada paquete. Así, la lógica de enrutamiento es esencialmente una máquina de estados finitos, la cual envía todos los *flits* de un paquete al mismo canal de salida antes de hacer una nueva decisión de enrutamiento del paquete vecino [30].

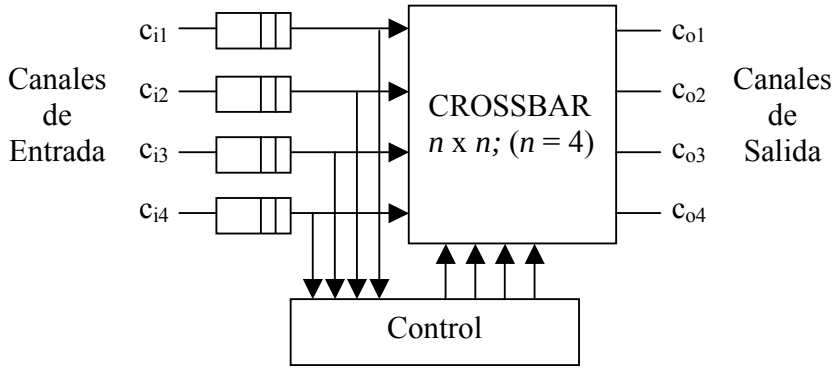


Figura 2.8. Estructura básica de un switch con *buffers* independientes en los canales de entrada. Este switch requiere un *crossbar* de *n*x*n*.

Es posible hacer una estimación simple del efecto de bloqueo de cabeza-de-línea, sobre la utilización del canal. Si se tienen dos puertos de entrada y, al azar, se selecciona una salida para cada uno, el primero tiene éxito y el segundo tiene una probabilidad del 50% de seleccionar un puerto no utilizado. El número esperado de paquetes por ciclo, a través del *switch*, es 1.5. La utilización esperada de cada salida es de 75%. Generalizando esto, si E(*n, k*) es el número esperado de puertos de salida cubiertos por *k* entradas aleatorias a un *switch* de *n*-puertos, entonces, $E(n, k + 1) = E(n, k) + (n - E(n, k))/n$. Calculando esta concurrencia con $k = n$, para varios tamaños de *switch*, revela que la

utilización esperada del canal de salida en un solo ciclo, para un *switch* cargado completamente, cae a cerca del 65% [31].

El impacto del bloqueo de cabeza-de-línea puede ser más significativo de lo que indica este análisis simple. Dentro de un *switch*, puede haber una explosión de tráfico para una salida, seguida de una explosión para otra, y así sucesivamente. Aunque el tráfico sea uniformemente distribuido, dada una ventana suficientemente grande, cada explosión resulta en un bloqueo sobre todas las entradas [32]. Una administración de *buffers* más flexible podría permitir el avance de paquetes hacia adelante de los que se encuentran bloqueados.

Como se dijo en el capítulo 1, el problema con *buffers* FIFO en los puertos de entrada es que los paquetes pueden ser bloqueados innecesariamente. Si el puerto de salida destino del paquete que se encuentra en el encabezado, está ocupado, entonces los demás paquetes del *buffer* se mantienen bloqueados, aún cuando sus puertos de salida destino estén ociosos. Para usar más eficientemente los puertos de salida, e incrementar el caudal del *switch*, los paquetes deben almacenarse separadamente, de acuerdo con el puerto de salida destino. Esto da origen al segundo tipo de arquitectura que propone Tamir y Frazier [13], el cual utiliza *buffers* FIFO separados para cada uno de los puertos de salida, en cada uno de los puerto de entrada [13], [15]. Este esquema requiere un *crossbar* de n^2 x n, ya que cada puerto de entrada se conecta con n líneas independientes a cada puerto de salida. Como puede verse en la Figura 2.9, cada *buffer* de entrada está dividido estáticamente entre n colas FIFO, necesarias para n puertos de salida.

Los inconvenientes del *crossbar* de n^2 x n son: el gasto que toma controlar n conmutadores de n x n; el desperdicio de espacio de almacenamiento, debido a su asignación estática [13]; y la complejidad en la implementación del control de flujo para que el *switch* transmita, sólo, si hay espacio en el *buffer* del *switch* destino. Con n

conmutadores de *n* x *n* se repite *n* veces el mismo *hardware* y cada puerto de entrada requiere *n buffers* separados y controladores de *buffers*. El espacio de *buffer*, disponible en cada puerto de entrada, está dividido estáticamente tal que, para un conmutador de *n* x *n*, solamente se dispone de 1/n del espacio del *buffer* de entrada como memoria potencial para cualquier paquete dado. Con *n buffers* separados en cada puerto de entrada, la información de cada *buffer* debe ser enviada al correspondiente puerto de salida. Esto es *n* veces la cantidad de información que se requiere para el control de flujo de un *buffer* FIFO.

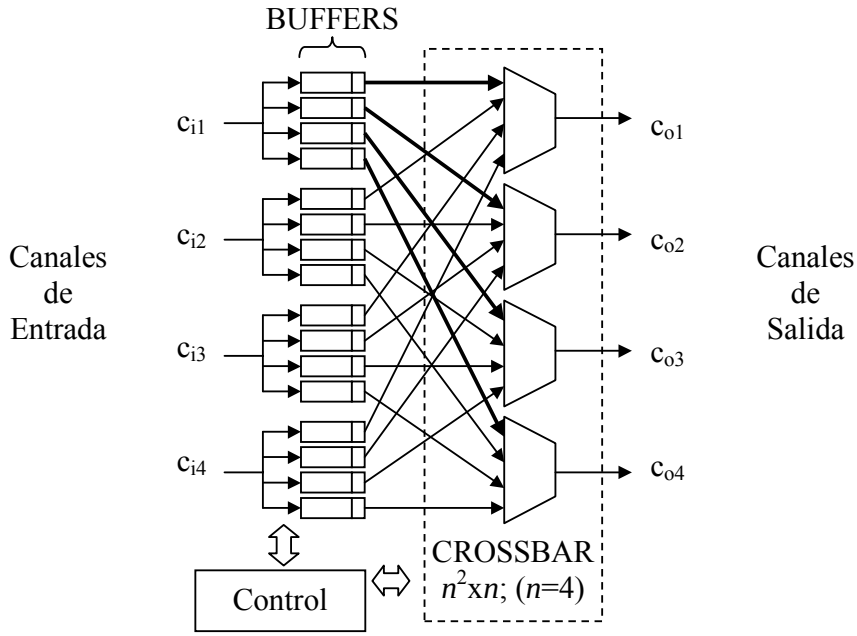


Figura 2.9. Estructura del *switch* (*n*=4) para evitar bloqueo de cabeza-de-línea. En cada puerto de entrada, se ordenan los paquetes por puerto de salida. Cuando alguna entrada tiene un paquete destinado a una salida, el controlador planifica la salida del paquete. Este tipo de *switch* requiere un *crossbar* n^2 x

Para simplificar el arreglo de *n buffers* separados, en cada puerto de entrada, Tamir y Frazier [13] proponen una tercer arquitectura que utiliza un solo *buffer* dividido en *n* colas separadas. Esto significa que el *buffer* tiene solamente un puerto de escritura y uno de

lectura, para las cuatro colas. El uso de un solo puerto de lectura no reduce el flujo de paquetes recibidos, debido a que las cuatro colas son suministadas, también, por un solo puerto de escritura. La ventaja está en que el *crossbar* se reduce a uno de n x n, evitando el desperdicio asociado al *crossbar* de n^2 x n. Sin embargo, los problemas causados por la asignación estática del espacio de almacenamiento, siguen siendo los mismos que ya se han mencionado.

Con asignación dinámica del espacio de almacenamiento, como se propone en la cuarta arquitectura de Tamir y Frazier [13], se puede incrementar el desempeño del *buffer*. La asignación dinámica almacena, en cada puerto de entrada, los paquetes destinados para cada puerto de salida en forma independiente. El *buffer* de cada puerto de entrada se divide en regiones -una región por cada puerto de salida- las cuales son de tamaño variable, de acuerdo con la demanda. La asignación de espacio se realiza a partir de una área vacía global de cada *buffer*, la cual es compartida por las regiones del *buffer*. A priori se espera que, en una red con un tráfico uniforme en los puertos de entrada, el desempeño del *buffer* de asignación dinámica sea similar al de asignación estática. Sin embargo, cuando ocurran explosiones de tráfico sobre un puerto de salida en particular, se espera que crezca la región de espacio asignado dinámicamente a dicho puerto y no se sature tan rápidamente como lo haría el *buffer* de asignación estática.

Tamir y Frazier [13] han reportado estudios comparativos de desempeño entre los cuatro tipos de arquitectura mencionados. Los resultados arrojados muestran que, en la mayoría de los casos, la asignación dinámica es mejor. Por otro lado, mediante pruebas preliminares que realizamos con simulación de eventos discretos para comprender mejor la asignación dinámica, también pudimos observar sus ventajas sobre la asignación estática. Por tal motivo, decidimos elegir como punto de referencia la opción de administración

dinámica de *buffers* para diseñar nuestra propuesta de solución, la cual presentamos a partir del capítulo siguiente con el nombre de *Buffer AutoCompactante* o *BAC*.

2.5 Conclusiones

En este capítulo vimos el contexto teórico general en el que se enmarca nuestra tesis. Hemos presentado las diversas opciones que existen actualmente para administrar los *buffers* de *switches* y ruteadores. Con base en un análisis comparativo, hemos visto que la opción con mejores características de desempeño es la que utiliza *buffers* en las entradas con múltiples colas asignadas dinámicamente. La implementación en *hardware* de un buffer con tales características es factible. Por tal motivo, a partir del capítulo siguiente desarrollaremos nuestro *buffer* denominado *BAC*. En el capítulo 3 veremos una descripción del *BAC*, a un nivel abstracto, en donde se aplican operaciones de escritura y lectura y se observa el comportamiento de las áreas del *buffer* definidas por las direcciones de escritura y de lectura. El modelo que desarrollamos en el capítulo mencionado no contempla detalles de implementación. Éstos se verán en los capítulos posteriores.

Ф

Capítulo 3

Modelo del Buffer AutoCompactante (*BAC*)

3.1 Introducción

El objetivo de éste capítulo consiste en diseñar el modelo sobre el cual se llevará a cabo el diseño lógico de nuestro *buffer*. El diseño se basa en la asignación dinámica del espacio de almacenamiento. Además, permite la ejecución simultánea de operaciones de escritura y lectura de datos en localidades distintas, así como en una misma localidad. Asimismo, maneja eficientemente el área de almacenamiento mediante un proceso de autocompactación de sus espacios.

El tratamiento del espacio de almacenamiento del *buffer*, consiste básicamente en el manejo de múltiples espacios, denominados *colas.* Análogamente a lo que ocurre en una memoria segmentada convencional, dichos espacios son independientes y pueden variar de tamaño [50], [70]. Regularmente, en sistemas de cómputo, las memorias segmentadas son diseños que combinan soluciones *hardware*/*software*, en donde la solución a este problema está dada por el diseño de la unidad de manejo de memoria del procesador, y es complementada en el nivel del sistema operativo [50], [68], [85], [70]. El desempeño de

este tipo de memorias se decrementa debido a dos factores primordiales: manejo de un control centralizado del espacio, y uso de *software* en la solución. Por lo anterior, proponemos un modelo de *buffer* con un control únicamente por *hardware*, debido a que el uso del software decrementaría su desempeño, parámetro importante en nuestro *buffer*.

Las colas son arreglos de datos, que deben permanecer contiguos para lograr un uso eficiente del espacio del *buffer*. Al mismo tiempo, la eficiencia del *buffer* debe ser manejada con características de alto desempeño. Por lo que nos hemos orientado hacia un diseño netamente hardware, donde los datos que se almacenan y se leen en el *buffer* son tratados como inserciones y extracciones, respectivamente [73], [78]. Las inserciones y extracciones provocan desplazamientos de datos sobre las celdas de almacenamiento, para mantener compactado el espacio del *buffer* al cual nos referiremos, a partir de ahora, como *Buffer AutoCompactante, BAC*.

En este capítulo presentamos un modelo abstracto tanto de la estructura del *BAC* como de las operaciones que puede realizar. La descripción del modelo no considera detalles de implementación y se centra en un modelo basado en teoría de conjuntos que permite definir sin ambigüedad tanto la estructura del *BAC* como sus funcionalidades. El capítulo contiene una sección que presenta una descripción general del *BAC*. Después presenta las estructuras básicas del *BAC* y la inicialización de éstas. Finalmente, describe las operaciones básicas y las operaciones paralelas del *BAC*. Ligado a este capítulo tenemos un Apéndice B en el que se da una breve explicación de la notación utilizada en la especificación del modelo del *BAC*.

3.2 Descripción General del *BAC*

El *BAC* se comporta como un arreglo de colas. Las colas están formadas por un conjunto de localidades de almacenamiento y son independientes. Cada cola es un arreglo

de datos FIFO, de tamaño variable, y está delimitado por un apuntador de escritura y otro de lectura. El tamaño de la cola varía de acuerdo con la demanda de escritura y lectura de datos.

Una de las ventajas del *BAC* radica en que las colas se mantienen siempre contiguas. El espacio libre del *BAC* está siempre disponible para permitir el incremento del tamaño de cualquiera de las colas. El *BAC* permite que una operación de lectura se realice al mismo tiempo que una escritura, aún en la misma localidad. Las colas están ubicadas dentro del *buffer* en orden creciente, con respecto a su dirección.

Los datos escritos en cada cola son tratados como inserciones. Una inserción, debida a la escritura de un dato, provoca el desplazamiento de los datos subsecuentes. Las operaciones de lectura son tratadas como extracciones, las cuales provocan desplazamientos en sentido inverso al de las operaciones de escritura. De esta manera se mantiene compactado el espacio ocupado y se evita la creación de espacios libres difíciles de manejar.

En la Figura 3.1 se muestra el ejemplo de un *BAC* con cuatro colas contiguas, de tamaños diferentes, y espacio libre. En la Figura 3.2.a se muestra el mismo *BAC* después de haber escrito un dato en la cola 2. La flecha de la figura indica que todos los datos que se encuentran en las localidades posteriores a la de escritura son desplazados hacia abajo, dejando así un hueco para insertar el nuevo dato a escribir. La Figura 3.2.b muestra el *BAC* después de haber realizado una escritura o inserción en la cola 2 y dos lecturas o extracciones en la cola 3. En esta figura, la flecha indica que todos los datos localizados después de la dirección de lectura son desplazados hacia arriba en una posición por cada operación de lectura, al mismo tiempo que se lee o extrae el dato durante la operación. Con este último corrimiento, se evita dejar el espacio que se crea durante la extracción, logrando así la compactación deseada.

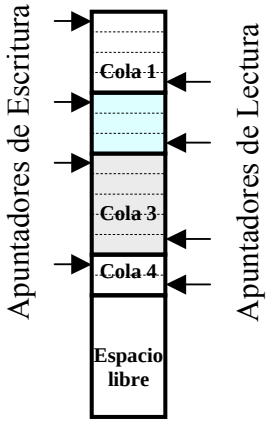


Figura 3.1 BAC con cuatro colas.

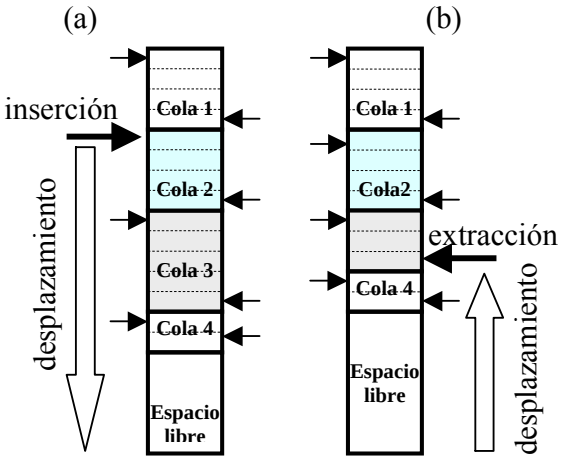


Figura 3.2 BAC con modificaciones.
Una escritura adicional en la cola 2.
Dos lecturas adicionales en la cola 3.

3.3 Estructura del *BAC*

El *BAC* está estructurado por dos módulos fundamentales: Un espacio de almacenamiento de datos *M* y un Registro de Direcciones *RD*. Cada módulo está constituido por un número *m* de localidades. Cada localidad del *BAC* contiene dos elementos: una *celda de almacenamiento* o *CM* y una *celda de registro de direcciones* o *CRD.* En la Figura 3.3 se muestra la estructura básica del *BAC* la cual está representada por los dos módulos fundamentales mencionados. También en ella se muestran sus buses de direcciones y datos.

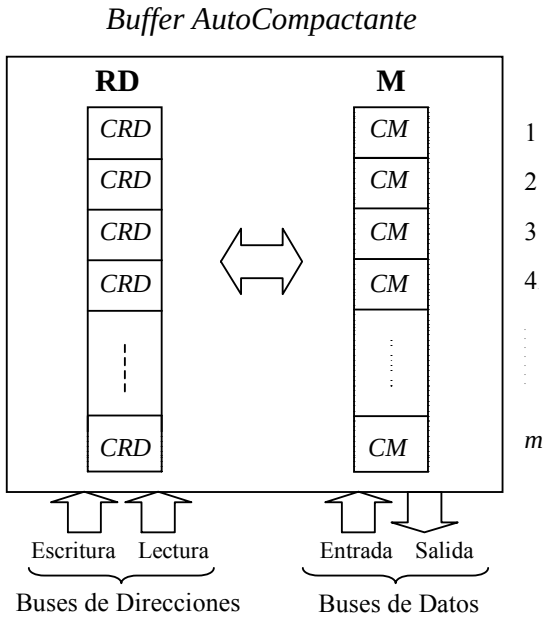


Figura 3.3 Estructura básica del *BAC*

3.3.1 Espacio de Almacenamiento

El espacio de almacenamiento es modelado por una función total *M* cuyo dominio es el conjunto de enteros de 1 hasta *m* (1..*m*) y su rango es el conjunto *MEM*, que representa el conjunto de valores que pueden almacenarse en cada celda de almacenamiento. Formalmente *M* queda especificada por la siguiente expresión:

$M \in 1..m \rightarrow MEM$

Informalmente *M* puede verse como un arreglo de dimensión *m* y tipo *MEM*; en este caso se suele decir que *M*(*i*), para cualquier valor de *i* entre 1 y *m*, representa una *celda de almacenamiento*.

Los valores almacenados en las celdas de almacenamiento pueden ser: datos provenientes de paquetes de información que circulan a través del *BAC*, modelados por el conjunto *DATA*, información sobre el estado de cada una de las colas del *BAC*, modelado por el conjunto *STATUS* o el valor *MFREE*, que sirve para denotar una celda de almacenamiento sin información. La unión de esos valores forma el conjunto *MEM*, que contiene el rango de *M*:

$MEM = DATA \cup STATUS \cup \{MFREE\}$

Los tres conjuntos constituyentes de *MEM* son conjuntos disjuntos, es decir la intersección de sus elementos es vacía. El estado de cada cola del *BAC* puede ser vacío, representado por el valor *VD*, o no vacío, representado por *NVD*; de esta manera *STATUS* se define como la unión de esos valores:

$STATUS = \{VD, NVD\}$

3.3.2 Registro de Direcciones

El espacio de almacenamiento es compartido entre las colas del *BAC*. Suponemos que *n* representa el número de colas manejadas; para cada una de esas colas, el *BAC* debe

identificar las celdas de almacenamiento que les son asociadas. La identificación se realiza a través del Registro de Direcciones que es especificado mediante una función total *RD*:

$RD \in 1..m \rightarrow DIR$

donde *DIR* es el conjunto de tipos de apuntadores:

$DIR = \{RR_c, WW_c, DD_c, FREE \mid c \in 1..n\}$

Para cualquier cola c, $1 \leq c \leq n$, los valores *RRc* y *WWc* denotan *apuntadores* a las celdas de almacenamiento donde se va a leer o escribir en la cola c, respectivamente; el valor *DDc* denota un apuntador a una celda de almacenamiento que contiene un dato de la cola c. Dado que por cada cola c, existe un único par de valores *RRc* y *WWc*, los índices i y j de *RD*, tales que $RD(i) = RRc$ y $RD(j) = WWc$, determinan las celdas de almacenamiento $M(i)$ y $M(j+1)$ que deben leerse o escribirse, respectivamente. El valor *FREE* denota un apuntador a una localidad de almacenamiento libre, es decir, para cualquier índice k, tal que $RD(k) = FREE$, la celda de almacenamiento $M(k)$ no contiene información de cualquiera de las colas del *BAC*.

Tradicionalmente en computación, un "apuntador" denota la dirección de una localidad en memoria donde se va a leer o escribir. En nuestro contexto, la semántica de un apuntador es ligeramente diferente. Nosotros interpretamos a un apuntador como un *valor* que identifica una operación de lectura o escritura en una cola determinada. Dado que por cada valor asociado a una operación de lectura o escritura en una cierta cola, se tiene asociado un índice único en *RD*, es ese índice quien funge como apuntador a la localidad que desea accederse.

Los elementos del arreglo, $RD(i)$, para $i \in 1..m$ son llamados *celdas del registro de direcciones*. Cada celda de almacenamiento $M(i)$ está asociada a la celda del registro de direcciones $RD(i)$.

3.3.3 Inicialización del *BAC*

Antes de entrar en operación normal, el *BAC* debe inicializar su registro de direcciones y su espacio de almacenamiento. En los párrafos siguientes mostramos los valores iniciales de las estructuras del *BAC*.

Las primeras n celdas del registro de direcciones son inicializadas de la manera siguiente:

$$\forall c \cdot (c \in 1..n \Rightarrow RD(c) = WW_c)$$

Esta inicialización indica que cualquier escritura en la cola c debe iniciar en la celda de datos $M(c+1)$. Las restantes celdas del registro de direcciones son inicializadas al valor *FREE*:

$$\forall j \cdot (j \in n+1..m \Rightarrow RD(j) = FREE)$$

El estado inicial de *RD* indica:

1. Las celdas de almacenamiento donde se escribirá el primer dato de cada cola y
2. Las celdas de almacenamiento disponibles para escribir datos.

Para cualquier cola c en estado inicial, ninguna celda del registro de direcciones tiene el valor *RRc*. Por lo que, en este estado es imposible realizar operaciones de lectura - las colas están vacías - . Los apuntadores de lectura se crean a partir de que las colas del *BAC* contienen datos.

Para ser coherente con la inicialización del registro de direcciones, el espacio de almacenamiento debe ser inicializado de modo que las n primeras celdas indiquen que cada una de las n colas están vacías. Esta condición queda formalmente definida por el siguiente predicado:

$$\forall i \cdot (i \in 1..n \Rightarrow M(i) = VD)$$

Después de la inicialización, las celdas de almacenamiento en el intervalo $n+1..m$ quedan disponibles para contener la información del las colas del *BAC*. Sin embargo, dado

que el registro de direcciones no contiene ningún apuntador de lectura, no necesitamos indicar explícitamente que las celdas en ese intervalo tienen el valor *MFREE*, y sólo requerimos especificar que las celdas en ese intervalo contienen cualquier valor de *DATA*. Por lo tanto, el siguiente predicado es verdadero después de la inicialización:

$$\forall i \cdot (i \in n+1..m \Rightarrow \exists d.(d \in DATA \wedge M(i) = d))$$

Ejemplo 3.3.1

Supóngase que se tienen cuatro colas en el *BAC* y que el espacio de almacenamiento consta de 9 celdas. Además considérese que *DATA* sólo consta de las letras del alfabeto. Los valores iniciales del espacio de almacenamiento y del registro de direcciones después de la inicialización son:

$M' = \{ (1,VD), (2,VD), (3,VD), (4,VD), (5,F), (6,T), (7,H), (8,G), (9,E) \}$

$RD' = \{ (1,WW_1), (2,WW_2), (3,WW_3), (4,WW_4), (5,FREE), (6,FREE), (7,FREE), (8,FREE), (9,FREE)\}$

Nótese que los valores de las celdas de almacenamiento en el intervalo 5..9 son uno de los posibles valores que son aceptables como valores iniciales. La especificación autoriza un no determinismo en esas celdas como valores iniciales. Sin embargo, dado que en ese mismo intervalo las celdas del registro de direcciones están marcadas con el valor *FREE* no existe el riesgo de confundir las celdas de almacenamiento en ese intervalo con información de alguna cola.

Fin del ejemplo

3.4 Operaciones Fundamentales del *BAC*

El *BAC* proporciona dos operaciones fundamentales para insertar y quitar información de cada una de sus colas. En los siguientes párrafos se presentan dichas operaciones, indicando para cada una de ellas, la información que reciben al invocarse, las condiciones

que deben satisfacer esas operaciones y los cambios en el espacio de almacenamiento y el registro de direcciones que realizan esas operaciones.

3.4.1 Operación de Escritura

Para realizar una operación de escritura, el *BAC* debe recibir dos datos:

1. el apuntador (o comando) de escritura en la cola *c* (*WWc*) y
2. el dato *D* a escribir.

La operación de escritura sólo puede realizarse si el *BAC* contiene localidades libres. Esta condición se verifica si existe una celda del registro de direcciones *RD*(*j*), para un valor de *j* entre 1 y *m*, que verifica la condición:

$RD(j) = MFREE$

Bajo esta condición, el *BAC* determina el índice *i*+1 de la celda de almacenamiento donde debe escribirse el dato *D*. El índice *i* se calcula mediante el registro de direcciones *RD*, y corresponde al índice de la celda del registro de direcciones cuyo contenido es igual a *WWc*. Es decir:

$RD(i) = WW_c$

Una vez determinado el índice de la celda de almacenamiento a modificar, se procede a la modificación del espacio de almacenamiento y del registro de direcciones. La modificación de ambas estructuras de datos se realiza en paralelo y depende del estado de la cola *c* antes de la operación, es decir si está vacía o no. Esta condición se verifica en la celda de almacenamiento *M*(*i*).

Escritura en cola sin información

Si la cola se encuentra vacía, es decir $M(i) = VD$, se realizan simultáneamente las siguientes tres operaciones sobre el espacio de almacenamiento:

1. la escritura de un valor *NVD* en la celda de almacenamiento *i*, indicando que la cola *c* tiene información,
2. la escritura del dato *D* en la celda de almacenamiento *i*+1 y
3. el corrimiento de cada celda *M*(*j*) a *M*(*j*+1) para cualquier *j*, tal que $i < j < m$

Después de la escritura, el nuevo valor de *M* está compuesto por:

1. las celdas de almacenamiento *M*(*j*), para todo *j*, tal que $j< i$: $1..i\text{-}1 \lhd M$
2. la celda de almacenamiento *i* que contiene un valor *NVD*: $\{(i, NVD)\}$
3. la celda de almacenamiento *i*+1 que contiene el nuevo dato *D*: $\{(i+1, D)\}$
4. el corrimiento de las celdas a partir de la celda *M*(*i*+1): $i+2..m \lhd (sub1 ; M)$ donde la función *sub1* está definida por $sub1(x) = x\text{-}1$ para todo *x*, tal que $x>0$.

Por lo tanto el nuevo valor de *M*, denotado por *M´*, después de la operación de escritura es:

$$M' = (1..i\text{-}1 \lhd M) \cup \{(i, NVD)\} \cup \{(i+1, D)\} \cup i+2..m \lhd (sub1; M)$$

El registro de direcciones se modifica acorde al estado vacío de la cola antes de la escritura. Esta creación conlleva dos operaciones simultáneas:

1. la creación de un apuntador de lectura para la cola c, *RRc*, en la celda *RD*(*i*+1) y
2. el desplazamiento de las celdas *RD*(*j*) a *RD*(*j*+1), para toda *j* entre *i*+1 y *m*-1.

Los valores de las celdas de *RD* entre 1 e *i* no cambian. De esta manera el valor de *RD* después de la operación de escritura está formado por:

1. las celdas del registro de direcciones *RD*(*j*), para todo *j*, tal que $j \leq i$: $1..i \lhd RD$
2. el apuntador de lectura a la cola *c* en la posición *i*: $\{(i , RR_c\}$
3. el corrimiento a partir de las celdas *RD*(*i*+1): $i+2..m \lhd (sub1 ; RD)$.

Por lo tanto el nuevo valor del registro de direcciones, denotado por *RD'*, después de la operación de escritura está dado por:

$RD' = (1..i \lhd RD) \cup \{(i+1 , RR_c)\} \cup i+2..m \lhd (sub1; RD)$

Ejemplo 3.4.1

Considerando los valores iniciales del espacio de almacenamiento y del registro de direcciones dados en el ejemplo 3.3.1, los valores de esas estructuras después de recibir una solicitud de escritura del valor *A* en la cola número tres (comando (WW_3,*A*)) son:

$M' = \{ (1,VD), (2,VD), (3,NVD), (4,A), (5,VD), (6,F), (7,T), (8,H), (9,G) \}$

$RD' = \{ (1,WW_1), (2,WW_2), (3,WW_3), (4, RR_3), (5,WW_4), (6,FREE), (7,FREE), (8,FREE), (9,FREE)\}$

Fin del ejemplo

Escritura en cola con información

En el caso que la cola no se encuentre vacía antes de la escritura, es decir $M(i)=NVD$, se realizan simultáneamente las siguientes tres operaciones en el espacio de direccionamiento:

1. la escritura de *D* en la celda de almacenamiento $i+1$ y
2. el corrimiento de cada celda $M(j)$ a $M(j+1)$ para cualquier *j*, tal que $i < j < m$

Por lo que, el nuevo valor de *M* está compuesto por:

1. las celdas de almacenamiento $M(j)$, para todo *j*, tal que $j \leq i$: $1..i \lhd M$
2. la celda de almacenamiento $i+1$ que contiene el nuevo dato *D*: $\{(i+1,D)\}$
3. el corrimiento de las celdas a partir de la celda $M(i+1)$: $i+2..m \lhd (sub1;M)$

En esta condición, el nuevo valor *M'* de *M* es:

$M' = (1..i \lhd M) \cup \{(i+1,D)\} \cup i+2..m \lhd (sub1;M)$

Dado que la cola *c* donde se escribe no está vacía, el registro de direcciones ya debe contener un apuntador de lectura *RRc* y sólo se requiere crear un apuntador *DDc* al nuevo dato introducido. Bajo estas condiciones, la actualización de *RD* conlleva dos operaciones simultáneas:

1. la escritura del apuntador DD_c en la celda *RD(i+1)* y
2. el desplazamiento de las celdas *RD*(*j*) a *RD*(*j*+1), para toda *j* entre *i*+1 y *m*-1.

Por lo tanto, el valor del registro de direcciones está formado por los siguientes valores:

1. las celdas del registro de direcciones *RD*(*j*), para todo *j*, tal que $j \leq i$: $1..i \lhd RD$
2. la celda *i*+1 que contiene el apuntador al nuevo dato escrito: $\{(i+1, DD_c)\}$
3. el corrimiento a partir de la celda *RD*(*i*+1): $i+2..m \lhd (sub1; RD)$

Por lo tanto, el nuevo valor *RD'* de *RD* después de la operación de escritura es:

$$RD' = 1..i \lhd RD \cup \{(i+1, DD_c)\} \cup i+2..m \lhd (sub1; RD)$$

Ejemplo 3.4.2

Continuando con el ejemplo 3.4.1, y tomando el estado del *BAC* después de la escritura en la cola 3:

M = { (1,*VD*), (2,*VD*), (3,*NVD*), (4,*A*), (5,*VD*), (6,*F*), (7,*T*), (8,*H*), (9,*G*) }

RD = { (1,WW_1), (2,WW_2), (3,WW_3), (4, RR_3), (5,WW_4), (6,*FREE*), (7,*FREE*), (8,*FREE*), (9,*FREE*)}

si se recibe otro comando de escritura del valor *B* en la cola tres (comando (WW_3,*B*)) cuyo estado no es vacío, los nuevos valores de las estructuras después de realizar la operación son:

M = { (1,*VD*), (2,*VD*), (3,*NVD*), (4,*B*) , (5,*A*), (6,*VD*), (7,*F*), (8,*T*), (9,*H*) }

RD = { (1,WW_1), (2,WW_2), (3,WW_3), (4,DD_3), (5, RR_3), (6,WW_4), (7,*FREE*), (8,*FREE*), (9,*FREE*)}

Fin del ejemplo

3.4.2 Operación de Lectura

Para realizar una operación de lectura, el *BAC* recibe como dato de entrada el valor del apuntador de lectura (o comando) a una cierta cola *c*, *RRc*. La operación de lectura sólo puede realizarse si existe al menos un dato en la cola *c*. Esta condición se verifica buscando un índice *i* del registro de direcciones que cumpla la condición:

$RD(i) = RR_c$

Si esta condición se cumple, el índice *i* también determina la celda de almacenamiento $M(i)$ que contiene el dato a leerse. La operación de lectura, dado que se comporta como una extracción de información, puede dejar con o sin información la cola leída. Esto conlleva a una modificación en paralelo de las estructuras del *BAC* y depende del estado que guarda la cola antes de la operación de lectura. Esta condición se verifica en la celda del registro de direcciones $RD(i\text{-}1)$.

Lectura en cola con un dato

Si se cumple la condición $RD(i\text{-}1) = WW_c$, esto implica que la cola *c* sólo contiene un dato, dado que $RD(i) = RRc$ e *i* apunta a ese dato. Bajo esta condición se realizan simultáneamente las siguientes operaciones en el espacio de almacenamiento:

1. la escritura del valor *VD* en la celda $i\text{-}1$, indicando que la cola *c* queda vacía.
2. el desplazamiento de las celdas $M(j+1)$ a $M(j)$ para toda *j* tal que $i \leq j < m$
3. la escritura del valor *MFREE* en la celda $M(m)$.

El valor de la celdas de almacenamiento en el intervalo $1..i\text{-}2$ no cambia. Por lo tanto el nuevo valor del espacio de almacenamiento después de la operación de lectura es:

$M' = (1..i-2 \lhd M) \cup \{(i\text{-}1, VD)\} \cup i..m\text{-}1 \lhd (add1 ; M) \cup \{(m, MFREE)\}$

El registro de direccionamiento se modifica para estar acorde con los cambios realizados en el espacio de almacenamiento. Esta modificación conlleva dos operaciones simultáneas:

1. el corrimiento de las celdas $RD(j+1)$ a $RD(j)$ para toda j tal que $i \leq j < m$, lo que provoca la desaparición del apuntador de lectura de la cola c y
2. la escritura del valor *FREE* en la celda $RD(m)$.

Los valores de las celdas del registro de direcciones entre los índices 1 e $i - 1$ no cambian. Por lo tanto, el nuevo valor del registro de direcciones después de la operación de lectura es:

$$RD' = (1..i-1 \lhd RD) \cup i..m-1 \lhd (add1 \; ; RD) \cup \{(m,FREE)\}$$

Ejemplo 3.4.3

Continuando con el ejemplo 3.4.1, el estado que guarda el *BAC* después de la escritura en la cola tres es:

$M = \{ (1,VD), (2,VD), (3,NVD), (4,A), (5,VD), (6,F), (7,T), (8,H), (9,G) \}$

$RD = \{ (1,WW_1), (2,WW_2), (3,WW_3), (4, RR_3), (5,WW_4), (6,FREE), (7,FREE), (8,FREE), (9,FREE)\}$

En este estado, la ejecución de una operación de lectura en la cola tres (comando RR_3) deja las estructuras del *BAC* en el siguiente estado:

$M' = \{ (1,VD), (2,VD), (3,VD), (4,VD), (5,F), (6,T), (7,H), (8,G), (9,MFREE)\}$

$RD' = \{ (1,WW_1), (2,WW_2), (3,WW_3), (4,WW_4), (5,FREE), (6,FREE), (7,FREE), (8,FREE), (9,FREE)\}$

Fin del ejemplo

Lectura en cola con más de un dato

Si se cumple la condición $RD(i\text{-}1) \neq WW_c$, esto implica que la cola c contiene más de un dato. Esto se deduce dado que $RD(i\text{-}1) = DD_c$ y $RD(i) = RR_c$, por lo que al menos i e i-1

apuntan a dos datos. Bajo esta condición se realizan simultáneamente las siguientes operaciones en el espacio de almacenamiento:

1. el desplazamiento de las celdas $M(j+1)$ a $M(j)$ para toda j tal que $i \leq j < m$, de modo que el estado de la cola se conserva en no vacío y
2. la escritura del valor *MFREE* en la celda $M(m)$ que indica la liberación de una celda de almacenamiento.

Las celdas de almacenamiento en el intervalo $1..i$ no cambian, por lo tanto el valor del espacio de almacenamiento después de la operación de lectura es:

$$M' = (1..i-1 \lhd M) \cup i..m\text{-}1 \lhd (add1 \; ; M) \cup \{(m,MFREE)\}$$

El registro de direcciones debe modificarse para indicar que se ha extraído un dato de la cola *c*, pero debe conservar el apuntador de lectura a la cola *c*, por lo tanto se deben realizar en paralelo las siguientes operaciones:

1. el desplazamiento de las celdas $RD(j+1)$ a $RD(j)$ para toda j tal que $i\text{-}1 \leq j < m$, lo que implica que el apuntador de lectura *RRc* sólo se desplace y
2. la escritura del valor *FREE* en la celad $RD(m)$.

Las celdas del registro de direcciones en el intervalo $1..i\text{-}2$ no cambian. Por lo tanto, el nuevo valor del registro de direcciones después de la operación de lectura es:

$$RD' = (1..i-2 \lhd RD) \cup i-1..m-1 \lhd (add1 \; ; RD) \cup \{(m,FREE)\}$$

Ejemplo 3.4.4

Continuando con el ejemplo 3.4.2, el estado que guarda el *BAC* después de la segunda escritura en la cola tres es:

$$M = \{ (1,VD), \; (2,VD), \; (3,NVD), (4,B) , \; (5,A), \; (6,VD), \; (7,F), \; (8,T), \; (9,H) \}$$

$$RD = \{ (1,WW_1), (2,WW_2), (3,WW_3), (4,DD_3), (5, RR_3), (6,WW_4), (7,FREE), (8,FREE), (9,FREE)\}$$

En este estado, si se recibe una solicitud de lectura en la cola tres (comando RR_3), el estado de las estructuras del *BAC* después de la operación es:

$M' = \{ (1,VD), (2,VD), (3,NVD), (4,B), (5,VD), (6,F), (7,T), (8,H), (9,\text{MFREE}) \}$

$RD' = \{ (1,WW_1), (2,WW_2), (3,WW_3), (4, RR_3), (5,WW_4), (6,FREE), (7,FREE), (8,FREE), (9,FREE)\}$

Fin del ejemplo

3.5 Operaciones Paralelas del *BAC*

El *BAC* dispone de dos buses de datos y dos de direcciones, por lo tanto le es posible realizar dos operaciones fundamentales simultáneamente. Sin embargo, por razones de implementación, el *BAC* no considera la concurrencia de dos operaciones del mismo tipo. En operaciones paralelas, el *BAC* sólo contempla la realización de una escritura y una lectura en cualquier cola, pero no dos escrituras o dos lecturas simultáneas. En función de los números de cola donde se realicen las operaciones, y para cualquier número de cola *c* y *d* en el intervalo 1..*n*, donde $c < d$, se tienen las siguientes posibilidades de recepción de comandos en los puertos del *BAC*:

1. escritura en la cola *c* y lectura en la cola *d*,
2. lectura en la cola *c* y escritura en la cola *d* y
3. escritura y lectura en una misma cola *c*.

Cada una de las combinaciones de recepción de comandos está asociada a una operación realizada por el *BAC*. Para referenciar esas operaciones denominamos operaciones “escritura/lectura” a las operaciones asociadas con la primera combinación, operaciones “lectura/escritura” a las asociadas con la segunda combinación y operaciones “escritura-lectura” (o “lectura-escritura”) a las asociadas con la tercera combinación. A las

tres clases de operaciones consideradas les denominaremos en general “operaciones paralelas” de ahora en adelante.

Los cambios en las estructuras del *BAC* provocados por la ejecución de las operaciones paralelas son iguales a los cambios realizados por la ejecución no determinista, secuencial y no interrumpida de las operaciones fundamentales que las componen. Sin embargo, aunque los cambios en las estructuras del *BAC* resultan los mismos, debe notarse que el número de corrimientos, tanto en el espacio de almacenamiento como en el registro de direcciones, se reduce en el caso de la ejecución paralela de las operaciones de escritura y lectura, dado que el desplazamiento de celdas originado por una operación de escritura se anula con el desplazamiento en sentido contrario provocado por una operación de lectura. Por otro lado, dado que en la operaciones paralelas siempre una operación de escritura se acompaña con una operación de lectura, una operación de escritura nunca puede abortar por falta de celdas de almacenamiento, dado que la operación de lectura siempre libera una. Los cambios en las estructuras del *BAC* efectuados por las operaciones paralelas dependen del estado que guardan las colas antes de realizarse las operaciones, de manera similar a los cambios provocados por las operaciones fundamentales. Por lo tanto, a partir de los dos casos considerados en la ejecución de las operaciones fundamentales de escritura y lectura, se tienen las siguientes posibilidades:

1. escritura en cola sin información y lectura en cola con un dato,
2. escritura en cola sin información y lectura en cola con más de un dato,
3. escritura en cola con información y lectura en cola con un dato y
4. escritura en cola con información y lectura en cola con más de un dato.

En los siguientes párrafos se detalla el funcionamiento de las operaciones paralelas considerando estos cuatro casos.

3.5.1 Operación de escritura/lectura

La operación de escritura/lectura, con comandos WW_c y RR_d, para valores de c y d en el intervalo $1..n$, tales que $c < d$, puede realizarse si la cola d tiene información. Suponemos que antes de realizarse la operación de escritura/lectura los índices i y j del registro de direcciones, con valores en el intervalo $1..m$, satisfacen la condición:

$$RD(i) = WW_c \wedge RD(j) = RR_d$$

y además que $i < j$ dado que $c < d$. Finalmente suponemos que D denota el dato a escribir ($D \in DATA$). En los párrafos siguientes analizamos los diferentes casos que pueden ocurrir en la ejecución de la operación de lectura/escritura bajo las condiciones dadas.

Escritura en cola sin información y lectura en cola con un dato

En este caso, el espacio de almacenamiento sólo se modifica en las celdas $M(k)$ para cualquier índice k tal que $i \leq k \leq j$. Como resultado de la ejecución de la operación de escritura, se marca el estado de la cola c a cola con información, se escribir el dato D y para cualquier valor de l tal que $i < l < j$-1, las celdas $M(l)$ se desplazan a $M(l+1)$. Por su parte, la operación de lectura sólo marca el estado de la cola d como cola vacía. Por lo tanto, el nuevo valor del espacio de almacenamiento después de la operación escritura/lectura es:

$$M' = (1..i\text{-}1 \lhd M) \cup \{(i, NVD)\} \cup \{(i+1, D)\} \cup (i+2..j\text{-}1 \lhd (sub1; M)) \cup \{(j, VD)\} \cup j+1..m \lhd M$$

Por otro lado, el registro de direcciones sólo se modifica en las celdas $RD(k)$ para cualquier índice k tal que $i < k \leq j$. Las modificaciones aportadas por la operación de escritura en el registro de direcciones son la inserción del apuntador de lectura a la cola c y el corrimiento para cualquier valor de l, tal que $i \leq l < j$, de las celdas $RD(l)$ a $RD(l+1)$. Nótese que el corrimiento de $RD(j\text{-}1)$ a $RD(j)$ borra el apuntador de lectura a la cola d. El nuevo valor del registro de direcciones después de la operación de escritura/lectura es:

$$RD' = (1..i \lhd RD) \cup \{(i+1, RRc)\} \cup (i+2..j \lhd (sub1; RD)) \cup j+1..m \lhd RD$$

Ejemplo 3.5.1

Considérese las estructuras del *BAC* del ejemplo 3.3.1 y supóngase el siguiente estado:

$M = \{(1,VD), (2,NVD), (3,A), (4,NVD), (5,C), (6, VD), (7,MFREE), (8,MFREE), (9, MFREE)\}$

$RD = \{(1,WW_1), (2,WW_2), (3,RR_2), (4,WW_3), (5,RR_3), (6, WW_4), (7,FREE), (8,FREE), (9,FREE)\}$

Supóngase que se recibe la operación de escritura/lectura con comandos (WW_1,D) y RR_3. Nótese que la cola 1 está vacía y la cola 3 tienen un dato. Por lo tanto, el estado de las estructuras después de la operación es:

$M' = \{(1,NVD), (2,D), (3,NVD), (4,A), (5,VD), (6, VD), (7,MFREE), (8,MFREE), (9, MFREE)\}$

$RD' = \{(1,WW_1), (2,RR_1), (3,WW_2), (4,RR_2), (5,WW_3), (6, WW_4), (7,FREE), (8,FREE), (9,FREE)\}$

Fin del ejemplo

Escritura en cola sin información y lectura en cola con más de un dato

En este caso las modificaciones del espacio de almacenamiento son similares a las modificaciones del caso anterior. La diferencia está en el rango de desplazamiento de las celdas y en el estado de la cola d al finalizar la operación. Así, en vez de desplazar las celdas de $M(l)$ a $M(l+1)$ para valores de l en el intervalo $i < l < j\text{-}1$, ahora se desplazan en el

intervalo $i < l < j$. Esto provoca el retiro del primer dato de la cola d, es decir, la celda $M(j)$ que contiene el dato a leerse en la cola d, es sobreescrita por el valor de la celda $M(j\text{-}1)$. Además nótese que el estado de la cola d no cambia con este corrimiento. Por lo tanto, el nuevo valor de M después de la operación de escritura/lectura es:

$$M' = (1..i\text{-}1 \lhd M) \cup \{(i, NVD)\} \cup \{(i+1, D)\} \cup (i+2..j \lhd (sub1; M)) \cup j+1..m \lhd M$$

En este caso también las modificaciones del registro de direcciones son similares a las modificaciones del caso anterior. La única diferencia está en el rango de los corrimientos de las celdas. En este caso, en vez de desplazar las celdas de $RD(l)$ a $RD(l+1)$ en el intervalo $i \leq l < j$, sólo se desplazan en el intervalo $i \leq l < j\text{-}1$, lo que provoca que las celdas de almacenamiento en el intervalo $j..m$ queden sin cambio y que la celda $M(j\text{-}1)$, que contiene un apuntador de dato a la cola d, sea sobrescrita con el valor de la celda $M(j\text{-}2)$, provocando la disminución de la cola d en un dato. Por lo tanto, el nuevo valor del registro de direcciones después de la operación de escritura/lectura es:

$$RD' = (1..i \lhd RD) \cup \{(i+1, RRc)\} \cup (i+2..j\text{-}1 \lhd (sub1; RD)) \cup j..m \lhd RD$$

Ejemplo 3.5.2

$M = \{(1,VD), (2,NVD), (3,A), (4,NVD), (5,C), (6, B), (7,VD), (8,MFREE), (9, MFREE)\}$

$RD = \{(1,WW_1), (2,WW_2), (3,RR_2), (4,WW_3), (5, DD_3), (6,RR_3), (7, WW_4), (8,FREE), (9,FREE)\}$

Supóngase que se recibe la operación de escritura/lectura con comandos (WW_1,D) y RR_3. Nótese que la cola 1 está vacía y la cola 3 tienen más de un dato. Por lo tanto, el estado de las estructuras después de la operación es:

$M' = \{(1,NVD), (2,D), (3,NVD), (4,A), (5,NVD), (6,C), (7, VD), (8,MFREE), (9, MFREE)\}$

$RD' = \{(1,WW_1), (2,RR_1), (3,WW_2), (4,RR_2), (5,WW_3), (6, RR_3), (7, WW_4), (8,FREE), (9,FREE)\}$

Fin del ejemplo

Escritura en cola con información y lectura en cola con un dato

En este caso las celdas el espacio de almacenamiento sólo se modifica en las celdas $M(k)$ para cualquier índice k tal que $i < k \leq j$. Dado que la cola c ya tiene información, sólo se le inserta el dato D y se hace el corrimiento de las celdas $M(l)$ a $M(l+1)$ para toda l en el intervalo $i < l < j$-1 como resultado de la operación de escritura. Esto provoca que el contenido de la celda $M(j$-1) que contiene el estado de cola con información, se sobrescriba con el valor de la celda $M(j$-2). Por otro lado, la cola d se marca como cola vacía en la celda $M(j)$ como resultado de la operación de lectura. Así, el nuevo valor del espacio de almacenamiento después de la operación de escritura/lectura, es el siguiente:

$$M' = (1..i \lhd M) \cup \{(i+1, D)\} \cup (i+2..j\text{-}1 \lhd (sub1; M)) \cup \{(j, VD)\} \cup j+1..m \lhd M$$

El registro de direcciones sólo se modifica en las celdas $RD(k)$ para cualquier índice k tal que $i < k \leq j$. Como resultado de la operación de escritura, solo se agrega un nuevo apuntador de dato a la cola c y se hace el corrimiento de las celdas $RD(l)$ a $RD(l+1)$ para toda l en el intervalo $i < l < j$. Nótese que en particular este corrimiento borra el contenido d la celda $RD(j)$ que contiene el apuntador de lectura a la cola d, dejando a dicha cola sin apuntador de lectura. De esta manera, el nuevo valor del registro de direcciones después de la operación de escritura/lectura es el siguiente:

$$RD' = (1..i \lhd RD) \cup \{(i+1, DDc)\} \cup (i+2..j \lhd (sub1; RD)) \cup j+1..m \lhd RD$$

Ejemplo 3.5.3

Considérese las estructuras del *BAC* del ejemplo 3.3.1 y supóngase el siguiente estado:

$M = \{(1,NVD), (2,E), (3,NVD), (4,A), (5,NVD), (6,C), (7, VD), (8,MFREE), (9, MFREE)\}$

$RD = \{(1,WW_1), (2,RR_1), (3,WW_2), (4,RR_2), (5,WW_3), (6,RR_3), (7, WW_4), (8,FREE), (9,FREE)\}$

Supóngase que se recibe la operación de escritura/lectura con comandos (WW_1,D) y RR_3. Nótese que las colas 1 y 3 tienen un dato. Por lo tanto, el estado de las estructuras después de la operación es:

M' = {(1,*NVD*), (2,*D*), (3,*E*), (4,*NVD*), (5,*A*), (6,*VD*), (7, *VD*), (8, *MFREE*), (9, *MFREE*)}

RD' = {(1,WW_1), (2, DD_1), (3,RR_1), (4,WW_2), (5,RR_2),(6,WW_3), (7, WW_4), (8,*FREE*), (9,*FREE*)}

Fin del ejemplo

Escritura en cola con información y lectura en cola con más de un dato

Las modificaciones del espacio de almacenamiento son similares al caso precedente. La diferencia está en el rango de los corrimientos y en el estado de la cola *d* que no cambia al finalizar la operación. De esta manera, en vez de desplazar las celdas $M(l)$ a $M(l+1)$ para valores de l en el intervalo $i < l < j-1$, ahora se desplazan en el intervalo $i < l <$ j. Este corrimiento sobrescribe la celda $M(j)$ donde se encuentra el dato a leerse de la cola *d*. De esta manera, el valor del espacio de almacenamiento después de la operación de escritura/lectura queda definido por la siguiente expresión:

$$M' = (1..i \lhd M) \cup \{(i+1, D)\} \cup (i+2..j \lhd (sub1; M)) \cup j+1..m \lhd M$$

El cambio del registro de direcciones también es similar al cambio realizado en el caso precedente. La sola diferencia está en el rango de desplazamiento; En este caso el desplazamiento de las celdas $RD(l)$ a $RD(l+1)$ está en el rango $i < l < j-1$, el cual no borra el contenido de la celda $RD(j)$ que contiene el apuntador de lectura a la cola *d*. Por lo tanto, el valor del registro de direcciones después de realizarse la operación de escritura/lectura es:

$$RD' = (1..i \lhd RD) \cup \{(i+1, DDc)\} \cup (i+2..j-1 \lhd (sub1; RD)) \cup j..m \lhd RD$$

Ejemplo 3.5.4

Considérese las estructuras del *BAC* del ejemplo 3.3.1 y supóngase el siguiente estado:

M = {(1,*NVD*), (2,*E*), (3,*NVD*), (4,*A*), (5,*NVD*), (6,*E*), (7,*C*), (8, *VD*), (9, *MFREE*)}

RD = {(1,WW_1), (2,RR_1), (3,WW_2), (4,RR_2),(5,WW_3), (6, DD_3), (7,RR_3), (8, WW_4), (9,*FREE*)}

Supóngase que se recibe la operación de escritura/lectura con comandos (WW_1,*D*) y RR_3. Nótese que la cola uno tiene un dato y la cola tres dos datos. Por lo tanto, el estado de las estructuras después de la operación es:

M' = {(1,*NVD*), (2,*D*), (3,*E*), (4,*NVD*), (5,*A*), (6,*NVD*), (7, *E*), (8,*VD*), (9, *MFREE*)}

RD' = {(1,WW_1), (2, DD_1), (3,RR_1), (4,WW_2), (5,RR_2),(6,WW_3), (7, RR_3), (8, WW_4), (9,*FREE*)}

Fin del ejemplo

3.5.2 Operación de lectura/escritura

La operación de lectura/escritura, con comandos RR_c y WW_d, para valores de c y d en el intervalo 1..n, tales que $c < d$, puede realizarse si la cola c tiene información. Suponemos que antes de realizarse la operación de lectura/escritura los índices i y j del registro de direcciones, con valores en el intervalo 1..m, satisfacen la condición:

$$RD(i) = RR_c \wedge RD(j) = WW_d$$

y además que $i < j$ dado que $c < d$. Finalmente suponemos que D denota el dato a escribir ($D \in DATA$). En los párrafos siguientes analizamos los diferentes casos que pueden ocurrir en la ejecución de la operación de lectura/escritura bajo las condiciones dadas.

Lectura en cola con un dato y escritura en cola sin información

En este caso, el espacio de almacenamiento sólo se modifica en las celdas $M(k)$ para cualquier índice k tal que $i\text{-}1 \leq k \leq j$. Como resultado de la ejecución de la operación de lectura, se marca el estado de la cola c a cola vacía y se recorren las celdas de $M(l\text{+}1)$ a $M(l)$ para cualquier valor de l tal que $i \leq l < j\text{-}1$. Nótese que el corrimiento borra el dato leído. Por su parte, los cambios efectuados por la operación de escritura ocasionan la inserción del dato D en la celda $M(j)$ y se marca en la celda $M(j\text{-}1)$ el estado de la cola d a cola con información. Por lo tanto, el nuevo valor del espacio de almacenamiento después de la operación de lectura/escritura es:

$$M' = (1..i-2 \lhd M) \cup \{(i\text{-}1, VD)\} \cup i..j\text{-}2 \lhd (add1 ; M) \cup \{(j\text{-}1, NVD)\} \cup \{(j, D)\} \cup j\text{+}1..m \lhd M$$

Por otro lado, el registro de direcciones sólo se modifica en las celdas $RD(k)$ para cualquier índice k tal que $i \leq k \leq j$. Las modificaciones aportadas por la operación de lectura consiste en el corrimiento de las celdas $RD(l\text{+}1)$ a $RD(l)$ para cualquier l en el intervalo $i \leq l < j$. Nótese que el corrimiento sobrescribe la celda $RD(i)$ que contiene el apuntador de lectura de la cola c, dejando ésta sin apuntador de lectura. Por su parte, el cambio provocado por la operación de escritura consiste en insertar un apuntador de lectura a la cola d. El nuevo valor del registro de direcciones después de la operación de lectura/escritura es:

$$RD' = (1..i-1 \lhd RD) \cup i..j\text{-}1 \lhd (add1 ; RD) \cup \{(j, RR_d)\} \cup j\text{+}1..m \lhd RD$$

Ejemplo 3.5.5

Considérese las estructuras del *BAC* del ejemplo 3.3.1 y supóngase el siguiente estado:

$M = \{(1,NVD), (2,E), (3,NVD), (4,A), (5,VD), (6,NVD), (7,B), (8,MFREE), (9,MFREE)\}$

$RD = \{(1,WW_1), (2,RR_1), (3,WW_2), (4,RR_2), (5,WW_3), (6,WW_4), (7,RR_4), (8,FREE), (9,FREE)\}$

Supóngase que se recibe la operación de lectura/escritura con comandos RR_1 y (WW_3,D) Nótese que la cola uno tiene un dato y la cola tres está vacía. Por lo tanto, el estado de las estructuras después de la operación es:

$M = \{(1,VD), (2,NVD), (3, A), (4,NVD), (5,D), (6,NVD), (7,B), (8, MFREE), (9, MFREE)\}$

$RD = \{(1,WW_1), (2,WW_2), (3,RR_2),(4,WW_3), (5, RR_3), (6, WW_4), (7, RR_4) , (8,FREE), (9, FREE)\}$

Fin del ejemplo

Lectura en cola con un dato y escritura en cola con información

En este caso los cambios en el espacio de almacenamiento son similares a los cambios del caso anterior. La diferencia está en el rango de desplazamiento de las celdas y en el estado de la cola *d* que ahora se mantiene en cola con información. Las celdas se desplazan de la posición $l+1$ a la posición l para toda l en el intervalo $i \leq l < j$. El nuevo valor del espacio de almacenamiento después de la operación de lectura/escritura en este caso está dado por la expresión siguiente:

$$M' = (1..i-2 \lhd M) \cup \{(i\text{-}1,VD)\} \cup i..j\text{-}1 \lhd (add1 ; M) \cup \{(j,D)\} \cup j+1..m \lhd M$$

El cambio del registro de direcciones es casi idéntico al caso precedente. La única diferencia está en que la celda $RD(j)$ ahora guarda una apuntador de dato a la cola *d*, en vez de un apuntador de lectura. El nuevo valor del registro de direcciones en este caso es:

$$RD' = (1..i-1 \lhd RD) \cup i..j\text{-}1 \lhd (add1 ; RD) \cup \{(j, DD_d)\} \cup j+1..m \lhd RD$$

Ejemplo 3.5.6

Considérese las estructuras del *BAC* del ejemplo 3.3.1 y supóngase el siguiente estado:

M = {(1,*NVD*), (2,*E*), (3,*NVD*), (4, *A*), (5,*NVD*), (6,*B*), (7,*NVD*), (8,*C*), (9, *MFREE*) }

RD = {(1,WW_1), (2,RR_1), (3,WW_2), (4,RR_2),(5,WW_3), (6 , RR_3), (7, WW_4), (8, RR_4) , (9,*FREE*)}

Supóngase que se recibe la operación de lectura/escritura con comandos RR_1 y (WW_3,*D*) Nótese que las colas uno y tres tienen un dato. Por lo tanto, el estado de las estructuras después de la operación es:

M = {(1,*VD*), (2,*NVD*), (3, *A*), (4,*NVD*), (5,*D*), (6,*B*), (7,*NVD*), (8, *C*), (9, *MFREE*)}

RD = {(1,WW_1), (2,WW_2), (3,RR_2), (4,WW_3), (5, DD_3), (6, RR_3), (7, WW_4), (8, RR_4) , (9,*FREE*)}

Fin del ejemplo

Lectura en cola con más de un dato y escritura en cola sin información

El espacio de almacenamiento en este caso se modifica en las celdas $M(k)$ para toda k en el intervalo $i \leq k \leq j$. Como resultado de la ejecución de la operación de lectura, las celdas $M(l+1)$ se desplazan a las celdas $M(l)$ para toda l en el intervalo) $i \leq l < j-1$. Nótese que el desplazamiento no altera el estado de la cola c el cual continúa a no estar vacía y sólo desaparece el dato leído. Por su parte, los cambios efectuados por la operación de escritura ocasionan la inserción del dato D en la celda $M(j)$ y se marca en la celda $M(j-1)$ el estado de la cola d a cola con información. Por lo tanto, el nuevo valor del espacio de almacenamiento después de la operación de lectura/escritura es:

$M' = (1..i-1 \lhd M) \cup i..j-2 \lhd (add1 ; M) \cup \{(j-1,NVD)\} \cup \{(j,D)\} \cup j+1..m \lhd M$

Por otro lado, el registro de direcciones sólo se modifica en las celdas $RD(k)$ donde k está en el intervalo $i-1 \leq k \leq j$. Las modificaciones aportadas por la operación de lectura son el corrimiento de las celdas $RD(l+1)$ a $RD(l)$ para cualquier l en el intervalo $i-1 \leq l < j-1$. Nótese que el corrimiento evita que la celda $RD(i)$ sea sobrescrita y por lo tanto se mantiene

el estado de la cola *c* en cola con información. Por su parte, el cambio provocado por la operación de escritura consiste en insertar un apuntador de lectura a la cola *d*. El nuevo valor del registro de direcciones después de la operación de lectura/escritura es:

$$RD' = (1..i-2 \lhd RD) \cup i-1..j-1 \lhd (add1 ; RD) \cup \{(j , RR_d)\} \cup j+1..m \lhd RD$$

Ejemplo 3.5.7

Considérese las estructuras del *BAC* del ejemplo 3.3.1 y supóngase el siguiente estado:

M = {(1,*NVD*), (2,*E*), (3,*F*), (4,*NVD*), (5, *A*), (6,*VD*), (7,*NVD*), (8,*B*), (9, *MFREE*)}

RD = {(1,WW_1), (2,DD_1), (3,RR_1), (4,WW_2), (5,RR_2), (6,WW_3), (7, WW_4), (8, RR_4), (9,*FREE*) }

Supóngase que se recibe la operación de lectura/escritura con comandos RR_1 y (WW_3,*D*) Nótese que la cola uno tiene dos datos y la cola tres está vacía. Por lo tanto, el estado de las estructuras después de la operación es:

M = {(1,*NVD*), (2,*E*), (3,*NVD*), (4, *A*), (5,*NVD*), (6,*D*), (7,*NVD*), (8,*B*), (9, *MFREE*)}

RD = {(1,WW_1), (2, RR_1), (3,WW_2), (4,RR_2), (5,WW_3), (6, RR_3), (7, WW_4), (8, RR_4) , (9, *FREE*)}

Fin del ejemplo

Lectura en cola con más de un dato y escritura en cola con información

Los cambios en el espacio de almacenamiento en este caso son similares al caso anterior. La diferencia está en los corrimientos y en el estado de la cola *d* que permanece sin cambio. El corrimiento se hace de las celdas $M(l+1)$ a $M(l)$ para toda l en el rango $i \leq l < j$. El corrimiento evita que el valor de la celda $M(j)$ se pierda, dejando así el estado de la cola *d* en cola con información. Por consiguiente, el estado del espacio de direccionamiento después de la operación de lectura/escritura es:

$$M' = (1..i-1 \lhd M) \cup i..j-1 \lhd (add1 ; M) \cup \{(j,D)\} \cup j+1..m \lhd M$$

Los cambios en el registro de direcciones son casi idénticos a los del caso anterior. La única diferencia está en que la celda $RD(j)$ ahora guarda un apuntador de dato a la cola d en vez de un apuntador de lectura. Por lo tanto, el valor del registro de direcciones después de la operación de lectura/escritura es:

$$RD' = (1..i-2 \lhd RD) \cup i-1..j-1 \lhd (add1 ; RD) \cup \{(j, DD_d)\} \cup j+1..m \lhd RD$$

Ejemplo 3.5.8

Considérese las estructuras del *BAC* del ejemplo 3.3.1 y supóngase el siguiente estado:

$M = \{(1,NVD), (2,E), (3,F), (4,NVD), (5,A), (6,NVD), (7,G), (8,NVD), (9,B)\}$

$RD = \{(1,WW_1), (2,DD_1), (3,RR_1), (4,WW_2), (5,RR_2), (6,WW_3), (7,RR_3), (8,WW_4), (9,RR_4)\}$

Supóngase que se recibe la operación de lectura/escritura con comandos RR_1 y (WW_3,D) Nótese que la cola uno tiene dos datos y la cola tres tiene uno. Por lo tanto, el estado de las estructuras después de la operación es:

$M = \{(1,NVD), (2,E), (3,NVD), (4,A), (5,NVD), (6,D), (7,G), (8,NVD), (9,B)\}$

$RD = \{(1,WW_1), (2,RR_1), (3,WW_2), (4,RR_2), (5,WW_3), (6,DD_3), (7,RR_3), (8,WW_4), (9,RR_4)\}$

Fin del ejemplo

3.5.3 Operación de escritura-lectura

La operación de escritura-lectura, con comandos WW_c y RR_c, para valores de c en el intervalo $1..n$, siempre puede realizarse cuando la cola c no está vacía. Suponemos que antes de realizarse la operación de escritura-lectura los índices i y j del registro de direcciones, con valores en el intervalo $1..m$, satisfacen la condición:

$$RD(i) = WW_c \wedge RD(j) = RR_c$$

y además que $i < j$ dado que siempre se retira el dato más viejo de la cola. Finalmente suponemos que D denota el dato a escribir ($D \in DATA$).

En las secciones precedentes, por cada operación paralela, se han analizado cuatro combinaciones que son resultado de las combinaciones de las operaciones fundamentales. En esta operación, dado que se lee y se escribe en una misma cola, siempre suponemos que la operación de escritura-lectura se efectúa en una cola con información. Por lo tanto los cuatro casos mencionados sólo se reducen a dos:

1. escritura y lectura en cola con más de un dato y
2. escritura y lectura en cola con un dato.

En los párrafos siguientes analizamos estos casos que pueden ocurrir en la ejecución de la operación de escritura-lectura bajo las condiciones dadas.

Escritura y lectura en cola con más de un dato

En este caso el cambio en las celdas de almacenamiento realizado por la ejecución de esta operación es idéntico al cambio que ocasiona la operación de escritura/lectura en el caso cuando se escribe en una cola con información y se lee de una cola con más de un elemento. Por lo tanto el valor del espacio de almacenamiento después de realizarse la operación de escritura-lectura es el siguiente:

$$M' = (1..i \lhd M) \cup \{(i+1, D)\} \cup (i+2..j \lhd (sub1; M)) \cup j+1..m \lhd M$$

Dado que los cambios en las celdas de almacenamiento fueron iguales a los cambios de la operación de escritura/lectura en el caso de escritura en cola con información y lectura en cola con más de un elemento, podríamos esperar que los cambios en el registro de

direcciones también fueran los mismos que los cambios hechos por esa operación. Sin embargo, dado que el corrimiento de las celdas borra un apuntador de datos y la escritura inserta otro apuntador de datos, el registro de direcciones permanece sin cambios después de realizarse esta operación, es decir:

$RD' = RD$

Ejemplo 3.5.9

Considérese las estructuras del *BAC* del ejemplo 3.3.1 y supóngase el siguiente estado:

$M = \{(1,NVD), (2,E), (3,F), (4,NVD), (5, A), (6,B), (7,VD), (8,NVD), (9,C)\}$

$RD = \{(1,WW_1), (2,DD_1), (3,RR_1), (4,WW_2), (5, DD_2), (6,RR_2), (7,WW_3), (8, WW_4), (9, RR_4)\}$

Supóngase que se recibe la operación de escritura-lectura con comandos (WW_2,D) y RR_2. Nótese que la cola dos tiene dos datos. Por lo tanto, el estado de las estructuras después de la operación es:

$M = \{(1,NVD), (2,E), (3,F), (4,NVD), (5, D), (6,A), (7,VD), (8,NVD), (9,C)\}$

$RD = \{(1,WW_1), (2,DD_1), (3,RR_1), (4,WW_2), (5, DD_2), (6,RR_2), (7,WW_3), (8, WW_4), (9, RR_4)\}$

Fin del ejemplo

Escritura y lectura en cola con un dato

En este caso el espacio de almacenamiento sólo se modifica en la celda $M(j)$ que es de donde se lee y a donde se escribe el dato. Todas las demás celdas no se alteran. Por lo tanto, el nuevo valor del espacio de almacenamiento es:

$M' = (1..j\text{-}1 \lhd M) \cup \{(j,D)\} \cup j+1..m \lhd M$

Por su parte, el registro de direcciones no cambia de valor después de esta operación, dado que la lectura hace desaparecer el apuntador de lectura y la escritura lo vuelve a crear. Por lo tanto se tiene:

$RD' = RD$

Ejemplo 3.5.10

Considérese las estructuras del *BAC* del ejemplo 3.3.1 y supóngase el siguiente estado:

$M = \{(1,NVD), (2,E), (3,F), (4,NVD), (5,B), (6,VD), (7,NVD), (8,C), (9,FREE)\}$

$RD = \{(1,WW_1), (2,DD_1), (3,RR_1), (4,WW_2), (5,RR_2), (6,WW_3), (7, WW_4), (8, RR_4), (9,MFREE)\}$

Supóngase que se recibe la operación de escritura-lectura con comandos (WW_2,D) y RR_2. Nótese que la cola dos tiene un dato. Por lo tanto, el estado de las estructuras después de la operación es:

$M = \{(1,NVD), (2,E), (3,F), (4,NVD), (5,D), (6,VD), (7,NVD), (8,C), (9,FREE)\}$

$RD = \{(1,WW_1), (2,DD_1), (3,RR_1), (4,WW_2), (5,RR_2), (6,WW_3), (7, WW_4), (8, RR_4), (9,MFREE)\}$

3.6 Conclusiones

En el presente capítulo hemos descrito el modelo de nuestro *buffer*. Primero, de una manera general, para lograr una rápida comprensión. Después, apoyándose en una notación con elementos de teoría de conjuntos, la descripción se hizo a un nivel abstracto, mostrando las características arquitecturales y funcionales del *BAC*. En la descripción se contempla un conjunto de 5 operaciones básicas, compuesto por dos fundamentales (escritura y lectura), y tres paralelas (escritura/lectura, lectura/escritura y escritura-lectura). Par facilitar la comprensión del modelo del *BAC*, introdujimos diversos ejemplos que muestran sus

aspectos interesantes de funcionalidad. Cada ejemplo se realiza bajo condiciones específicas diferentes de manera todos ellos se complementan entre sí para describir completamente la funcionalidad del *BAC*.

Capítulo 4

Diseño del Módulo de Control para el Buffer AutoCompactante (*BAC*)

4.1 Introducción

Para que el *BAC* se comporte de acuerdo con la descripción del modelo abstracto, es necesario un módulo de control que genere comandos a cada una de sus localidades, y éstas realicen las acciones que satisfagan el comportamiento. Este capítulo contiene el diseño del módulo de control del *BAC*, al cual nos referiremos de aquí en adelante como *Control Distribuido, CD*.

El capítulo inicia con una breve descripción estructural del *BAC* que incluye al módulo de control *CD* y su relación con el registro de direcciones *RD* y con el espacio de almacenamiento *M*. Posteriormente, se describen las acciones que puede realizar el módulo de control en cada una de las localidades del *BAC*. Finalmente, se describen los diversos casos en los que puede encontrarse cada localidad y las acciones que deben realizar para satisfacer el modelo abstracto.

4.2 Estructura General del Módulo de Control, *CD*

Las acciones individuales de cada localidad del *BAC* conllevan, en su conjunto, a la ejecución de las operaciones fundamentales y paralelas que puede realizar. Para ello, se toma en cuenta el tipo de operación recibida y la posición relativa de cada localidad a controlar dentro del *BAC*. A las acciones que realiza cada localidad las hemos denominado *primitivas*. Las *primitivas* son operaciones más simples, comandadas por el módulo *CD*.

En la Figura 4.1.a se muestra la estructura del *BAC*, similar a la de la Figura 3.3, del capítulo anterior, pero incluyendo al módulo de control. El módulo *CD* está compuesto por

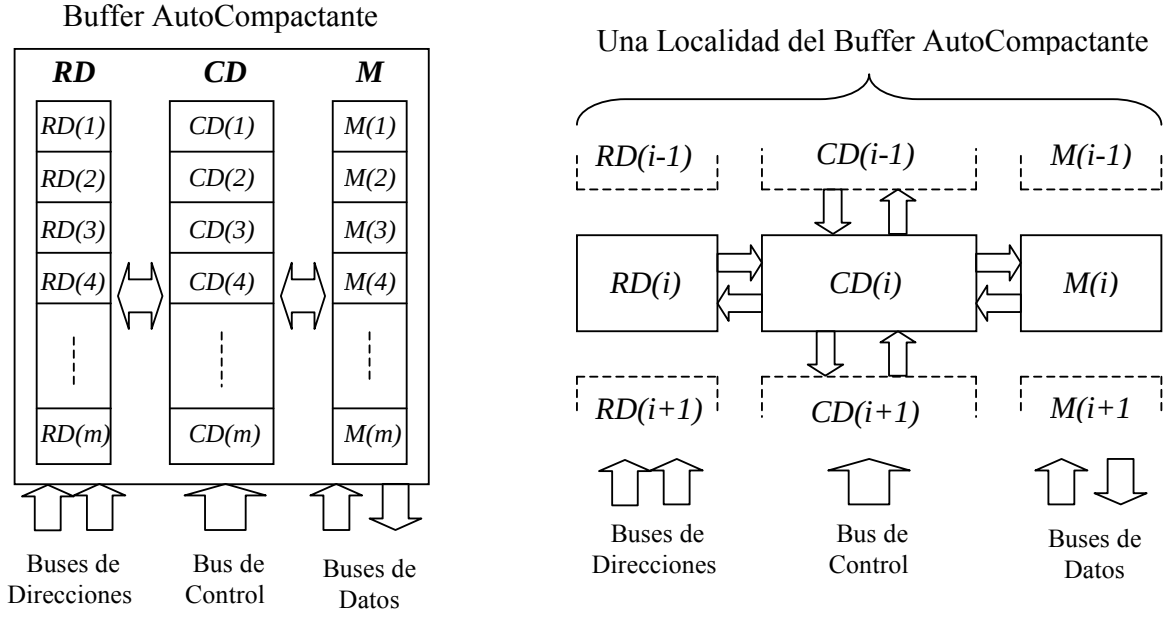


Figura 4.1 Estructura del BAC. (a) Estructura general. (b) Vista de una localidad.

m celdas; cada celda *CD*(*j*), para cualquier *j* entre 1 y *m*, controla tanto las celdas de almacenamiento *M*(*j*), como a las celdas del registro de direcciones *RD*(*j*). Además, cada celda *CD*(*i*), donde $1 < i < m$, envía y recibe señales de referencia hacia y desde las celdas

CD(*i*+1) y *CD*(*i*-1), como se muestra en la Figura 4.1.b, correspondiente a una localidad del *BAC*. Tales señales hacen que las celdas de almacenamiento *M*(*i*) y las del registro *RD*(*i*) realicen simultáneamente las operaciones *primitivas* que les corresponden.

4.3 Modelo Concreto al Nivel de Operaciones Primitivas

A partir del tipo de operación que se aplica al buffer (fundamental o paralela), cada celda de control toma decisiones de control que dependen de su posición relativa con respecto a las direcciones de escritura y/o lectura. El estado de la celda de control y de sus celdas adyacentes, determina la ejecución de una o varias operaciones primitivas que concretan la operación básica. Las operaciones primitivas son:

- *escritura o lectura de dato,*
- *lectura y escritura de dato,*
- *desplazamiento de dato hacia abajo o hacia arriba,*
- *escritura o lectura de estado de cola,*
- *creación o eliminación de apuntador de dato,*
- *creación o eliminación de apuntador de lectura,*
- *ninguna operación.*

En función de la posición de la celda de control y de la operación fundamental o paralela a ejecutarse, existen treinta y ocho casos en los que pueden encontrarse las diferentes celdas. En cada uno de esos casos, cada celda de control genera comandos para realizar una o varias operaciones primitivas.

Los casos son analizados agrupándolos en cinco tipos, derivados de cada una de las 5 operaciones básicas de nuestro modelo abstracto (escritura, lectura, escritura/lectura, lectura/escritura, y escritura-lectura).

La descripción de cada caso se inicia con su identificación general y continúa con la definición de las condiciones de entrada de cada celda de control correspondiente al caso en estudio. Posteriormente, se definen las salidas de cada celda de control asociada al caso en análisis.

En la descripción de casos, se considera a la celda de control como la entidad de referencia. Ver la Figura 4.1.b. Así, una celda de control *CD*(*i*) se comunica a través de comandos de entrada y salida con cuatro entidades: *CD*(*i*-1), *CD*(*i*+1), *RD*(*i*) y *M*(*i*). Además, las celdas de control reciben las siguientes señales externas: dirección de escritura, dirección de lectura, y comandos para indicar que tipo de operación se realizará.

El *bus de control*, mostrado en la Figura 4.1, contiene las cuatro señales siguientes: *CW*, *CR*, *CWR* y *CRW*. Estos comandos activan las operaciones fundamentales deseadas sobre el *BAC* y están implícitos en la descripción de casos. *CW* sirve para activar las operaciones de escritura. *CR* se utiliza para activar las operaciones de lectura. *CWR* activa una operación paralela de tipo escritura/lectura y *CRW* activa una operación lectura/escritura. La operación de escritura-lectura descrita en el capítulo anterior es un caso especial de la operación de escritura/lectura, por lo que se activa con el comando *CWR*.

En este capítulo denominamos el subconjunto de apuntadores *DIR* - {*FREE* } como el conjunto de *direcciones* y lo denotamos por *ADR*. De la definición del conjunto *DIR* del capítulo precedente resulta que:

$$ADR = \{RR_c, WW_c, DD_c \mid c \in 1..n\}$$

A partir de este capítulo, los elementos de *ADR* los denominamos "direcciones". Con el fin de poder comparar direcciones, definimos la relación de orden "menor que" entre los elementos de *ADR*. De esta manera, para dos direcciones diferentes *a* y *b*, *a* es menor que *b* ($a \prec b$), si y solamente si existen dos celdas del registro de direcciones con índices *i* y *j*, donde $i < j$, conteniendo respectivamente las direcciones *a* y *b*. Formalmente esta relación está definida por:

$$\forall a,b \cdot (a \in ADR \land b \in ADR \land a \neq b \Rightarrow$$

$$a \prec b \Leftrightarrow \exists i,j \cdot (i \in 1..m \land j \in 1..m \land RD(i) = a \land RD(j) = b \land i < j))$$

Tomando en cuenta el funcionamiento del *BAC* descrito en el capítulo anterior, podemos verificar que para cualquier cola *c* con más de un dato tenemos:

$$WW_c \prec DD_c \prec RR_c$$

Además, para dos colas cualquiera *c* y *d*, donde $c < d$, tenemos

$$WW_c \prec WW_d$$

Un *apuntador activo* es una celda del registro de direcciones *RD*, con índice *i*, tal que su valor coincide con el valor aplicado en el bus de direcciones del *BAC*. Por ejemplo, para una operación de escritura sobre una cola *c* existe un apuntador activo $RD(i)$, tal que $RD(i)=WW_c$. Los apuntadores activos pueden ser de escritura o lectura. A los apuntadores que no son activos los denominaremos *apuntadores pasivos*. Las direcciones de localidades vacías, correspondientes al espacio libre del *BAC*, son un caso especial de dirección y toman el valor cero.

En la siguiente descripción de casos utilizamos el término *precedencia* para indicar el tipo de apuntador que antecede a otro. Supóngase una operación de escritura sobre una cola *c* y las celdas *RD*(*i*) y *RD*(*i*+1), tal que, *RD*(*i*)=WW_c. Podemos decir que la celda *RD*(*i*+1) tiene una precedencia de apuntador de escritura activo.

4.3.1 Casos Derivados de una Operación de Escritura en una Cola *c*

Los comandos externos necesarios para que el *BAC* realice una operación de escritura, sobre una cola *c* dada, son la dirección de escritura WW_c, y un comando de escritura *CW*. Durante la operación de escritura sobre la cola *c*, tal que *RD*(*i*) = WW_c, el *BAC* está definido por las cuatro regiones siguientes:

1. Localidades con un índice *j*, tal que *j*< *i*,
2. Localidad con índice *i* sobre la cual se hará la escritura,
3. Localidades con un índice *j* > *i*, que no pertenezcan al espacio libre
4. Localidades con un índice *j* > *i*, pertenecientes al espacio libre.

La Figura 4.2 muestra las cuatro regiones del *BAC* para una operación de escritura. En ella se toma como referencia al registro de direcciones *RD*. La parte alta de la figura contiene las direcciones bajas.

Casos 1 y 2

Estos casos se presentan en la localidad con índice *i,* que es apuntada por una dirección de escritura WW_c, sobre una cola *c*. Lo interesante de estos dos casos está en determinar si cambia el estado de la cola *c*.

Las condiciones de entrada son:

1. *RD*(*i*)=WW_c,

El estado de la cola *c* a escribir es *VD* (caso 1) o *NVD* (caso 2)

Las acciones de control de la celda *CD*(*i*) son:

1. Escritura de estado *NVD* de la cola *c*, en la celda *M*(*i*).

   En el caso 1 cambia de *VD* a *NVD*. En el caso 2 se mantiene en estado *NVD*.

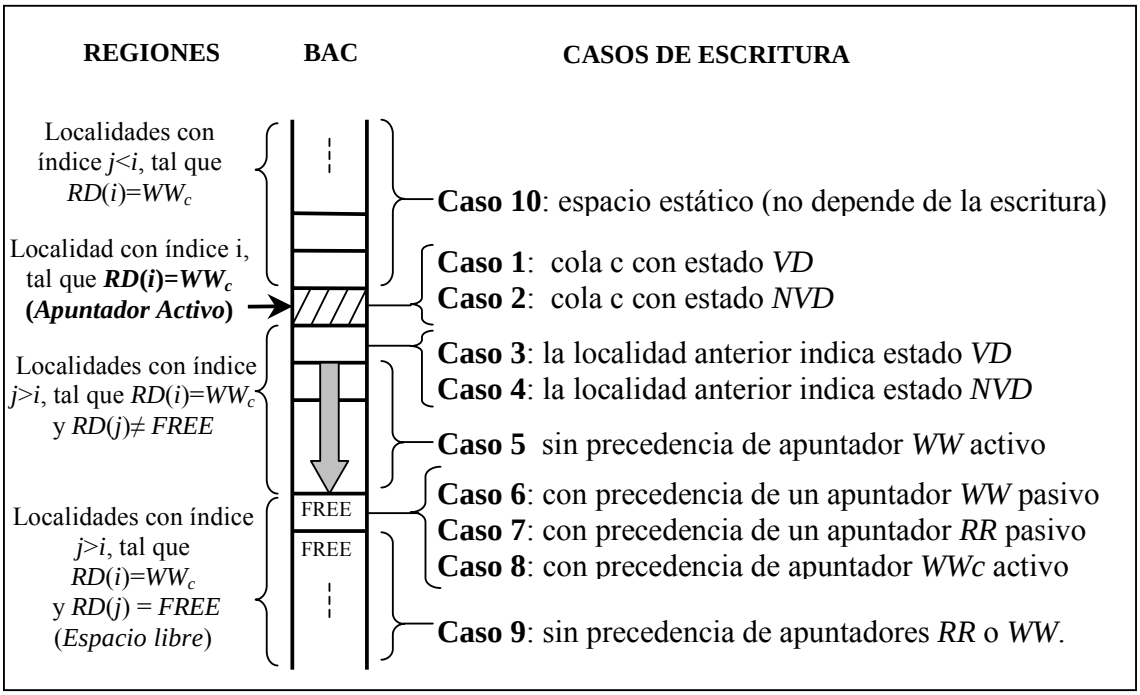


Figura 4.2 Casos que se derivan de una operación de escritura en una cola *c* .

Casos 3 y 4

Estos casos se presentan en la localidad con índice *i*+1, tal que $RD(i) = WW_c$. Lo que interesa aquí es saber si la cola *c* está vacía. Cuando la cola *c* está vacía, se crea en *RD*(*i*+1) un apuntador RR_c. En caso contrario, se crea un apuntador DD_c. En ambos casos se escribe el dato *D* en la celda *M*(*i*+1).

Las condiciones de entrada son:

1. $RD(i+1) \succ WW_c$,

2. $RD(i) = WW_c$,

3. $M(i) = VD$ (caso 3), o $M(i) = NVD$ (caso 4)

Las acciones de control de la celda $CD(i+1)$ son:

1. Crear en $RD(i+1)$ un apuntador *RRc* (caso 3), o un apuntador *DDc* (caso 4),

2. Escribir el dato *D* en la celda $M(i+1)$.

Caso 5

El caso 5 se presenta en las localidades con índice $j>i+1$, tal que $RD(i)=WWc$ y $RD(j)\neq FREE$. Las localidades con índice *j* que cumplen con la condición anterior desplazan su contenido hacia las localidades con índice $j+1$ y escriben el contenido de las localidades con índice $j-1$.

Las condiciones de entrada para cada celda $CD(j)$ son:

1. $RD(j) \succ WW_c$

2. $RD(j-1) \neq WW_c$ (no es apuntador de escritura activo)

Las acciones de control de cada $CD(j)$ son:

1. Desplazar direcciones hacia abajo. El valor de $RD(j)$ toma el valor de $RD(j-1)$.

2. Desplazar datos hacia abajo. El valor de la celda $M(j-1)$ se escribe en $M(j)$.

Casos 6 y 7

Estos casos aparecen en la localidad del espacio libre con índice *j*, tal que la celda $RD(j) = FREE$ está precedida por un apuntador pasivo *WW* (caso 6), o *RR* (caso 7). Es decir $RD(j-1)$ es un apuntador de escritura o lectura de una cola diferente a la cola *c* donde se hace la escritura. En estos casos, la respuesta es una operación de desplazamiento hacia abajo.

Las condiciones de entrada son:

1. $RD(j) = FREE$
2. $RD(j\text{-}1) = RR_d$. (para el caso 6)

   $RD(j\text{-}1) = WW_d$. (para el caso 7

donde *d* denota una cola diferente a la de escritura.

Las acciones de control de la celda $CD(j)$ son:

1. Desplazamiento hacia abajo sobre la celda $RD(j)$,
2. Desplazamiento hacia abajo sobre la celda $M(j)$.

Caso 8

Este caso aparece en la localidad del espacio libre con índice *j*, tal que la celda $RD(j) = FREE$ está precedida por un apuntador activo. Es decir, $RD(j\text{-}1)$ es un apuntador a una cola vacía donde se realiza la escritura, WW_c. La operación de escritura implica la creación de un apuntador de lectura RR_c, en la celda $RD(j)$. Simultáneamente, en la celda $M(j)$ se escribe el dato *D*, tomado del bus de datos del *BAC*.

Las condiciones de entrada son:

1. $RD(j) = FREE$,
2. $RD(j\text{-}1) = WW_c$

Las acciones de control de la celda $CD(j)$ son:

1. Crear un apuntador de lectura, RR_c, en la celda $RD(j)$
2. Escribir en la celda de almacenamiento $M(j)$ el dato *D* presente en el bus de datos del *BAC*.

Caso 9

Si una localidad del espacio libre con índice j es precedida por algún apuntador WW_n o RR_n, donde n es el número de cola del *BAC*. Entonces la localidades con índices j+1 hasta m pertenecen a este caso. Dichas localidades permanecen sin cambio después de la escritura.

Las condiciones de entrada son:

1. $RD(j) = FREE$,
2. $RD(j\text{-}1) \neq WW_n$ y $RD(j\text{-}1) \neq RR_n$.

Las acciones de salida de cada celda $CD(j)$ son:

1. Ninguna operación

Caso 10

Las localidades con índice $j<i$, tal que $RD(i)=WW_c$, pertenecen a este caso. Las operaciones de escritura afectan solamente a localidades con índice mayor o igual que el índice i. Por lo tanto, estas localidades no son afectadas por la escritura.

Las condiciones de entrada son:

1. $RD(j) \succ WW_c$
2. $RD(j) \neq FREE$

Las acciones de control de las celdas de este caso son:

1. Ninguna operación

4.3.2 Casos Derivados de una Operación de Lectura en una Cola *c*.

Los comandos que requiere el *BAC* para realizar una operación de lectura, sobre una cola *c*, son la dirección de lectura RR_c, y un comando *CR*. Ante la operación de lectura en una localidad *i* de la cola *c*, el *BAC* muestra las regiones siguientes:

1. Localidades con índice $j<i$, tal que $RD(i)=RR_c$,
2. Localidad con índice *i*, tal que, $RD(i)=RR_c$,
3. Localidades con índice $j>i$, tal que $RD(i)=RR_c$ y $RD(j)\neq FREE$,
4. Localidades con índice $j>i$, tal que $RD(i) = RR_c$ y $RD(j) = FREE$

La Figura 4.3 muestra las cuatro regiones del *BAC* para una operación de lectura. Tal como en la Figura 4.2, en ella se toma como referencia al registro de direcciones *RD*.

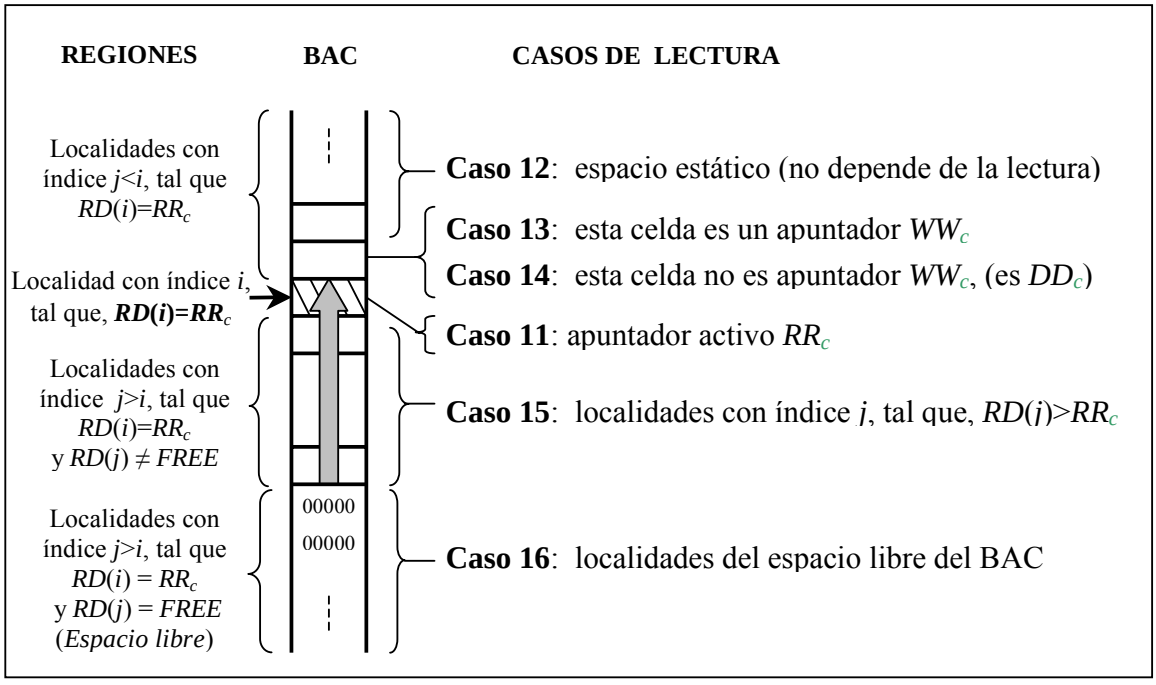


Figura 4.3 Casos que se derivan de una operación de lectura en una cola *c*

Caso 11

Este caso se observa en la localidad i, tal que $RD(i) = RR_c$. Es decir, se trata de la localidad donde se realiza la operación de lectura. Con la operación de lectura, queda una localidad libre en el *BAC* que es ocupada por lo que contiene la localidad siguiente, con índice i+1.

Las condiciones de entrada son:

1. $RD(i) = RR_c$

Las acciones de control de la celda $CD(i)$ son:

1. Habilitar la lectura de dato
2. Desplazar hacia arriba los contenidos de $M(i+1)$ y $RD(i+1)$

Caso 12

El caso 12 se presenta en las localidades con índice $j < i$-1, tal que $RD(i) = RR_c$. Estas localidades no son afectadas por la operación de lectura. Por consiguiente no cambian.

Las condiciones de entrada son:

1. $RD(j) \succ RR_c$
2. $RD(j+1) \neq RR_c$,

Las acciones de control en las celdas $CD(j)$ son:

1. Ninguna operación

Casos 13 y 14

Estos casos se observan en la localidad con índice i-1, tal que $RD(i) = RR_c$. Es decir, el caso 13 ocurre cuando la cola c tiene un solo dato. El caso 14 ocurre cuando en la cola c

hay más de un dato. Lo interesante de estos casos se observa en el estado de la cola guardado en $M(i)$.

Las condiciones de entrada son:

1. $RD(i\text{-}1) = WW_c$. (para el caso 13).

   Significa que la cola c tiene sólo un dato.

2. $RD(i\text{-}1) \neq WW_c$. (para el caso 14).

   Significa que la cola c tiene mas de un dato.

Acciones de control de la celda $CD(i\text{-}1)$:

1. Escribir en $M(i\text{-}1)$ el estado *VD* de la cola c. Respuesta para el caso 13.

   $RD(i\text{-}1)$ no cambia. Solo cambia $M(i\text{-}1)$, de estado *NVD* a *VD*.

2. Escribir en $RD(i\text{-}1)$ el contenido de $RD(i)$. Respuesta para el caso 14.

   Significa que el apuntador de escritura RR_c se desplaza hacia arriba.

Caso 15

Las localidades que se encuentran en el caso 15 son aquellas que se desplazan hacia arriba para llenar el espacio que deja la operación de lectura. Estas son las que tienen índice $j>i$, tal que $RD(i)=RR_c$ y $RD(j)\neq FREE$.

Las condiciones de entrada son:

1. $RD(j) \succ RR_c$

2. Las acciones de control de $CD(j)$ son:

1. Desplazar hacia arriba los contenidos de las celdas $RD(j)$ y $M(j)$

Caso 16

El caso 16 se encuentra en las localidades del espacio libre del *BAC*. Estas localidades no se modifican con la operación de lectura.

Las condiciones de entrada son:

1. $RD(j) = FREE$

Las acciones de control son:

1. Ninguna operación

4.3.3 Casos Derivados de una Operación de Escritura/Lectura.

Las operaciones de escritura/lectura se activan con la señal de control *CWR*. La escritura y lectura están comandadas por WW_c y RR_d, respectivamente. La escritura se realiza en una localidad con índice i de la cola c y la lectura en una localidad con índice j de la cola d, tal que $c<d$, por lo que $i<j$. Así, el *BAC* puede visualizarse mediante las 6 regiones siguientes:

1. Localidades con índice $k<i$,
2. Localidad con índice i, tal que, $RD(i) = WW_c$
3. Localidades con índice k, tal que, $i<k<j$,
4. Localidad con índice j, tal que, $RD(j) = RR_d$
5. Localidades con índice $k>j$, tal que $RD(k) \neq FREE$
6. Localidades con índice $k>j$, tal que $RD(k) = FREE$

En la Figura 4.4 se muestran las seis regiones del *BAC*. En ella se observa que los casos 17, 18, 19, 20, 21 y 22 son similares a los casos 10, 1, 2, 3, 4 y 5, respectivamente. Esta similitud aparece en las primeras 3 regiones del *BAC*, las cuales se comportan igual

que las primeras 3 regiones de *BAC* durante una operación de escritura (véase la Figura 4.2). Sin embargo, la similitud sólo está en las acciones de control y no en las condiciones de entrada. Por simplicidad, la siguiente descripción de los casos 17 a 22 hará referencia a los casos 10, 1, 2, 3, 4 y 5.

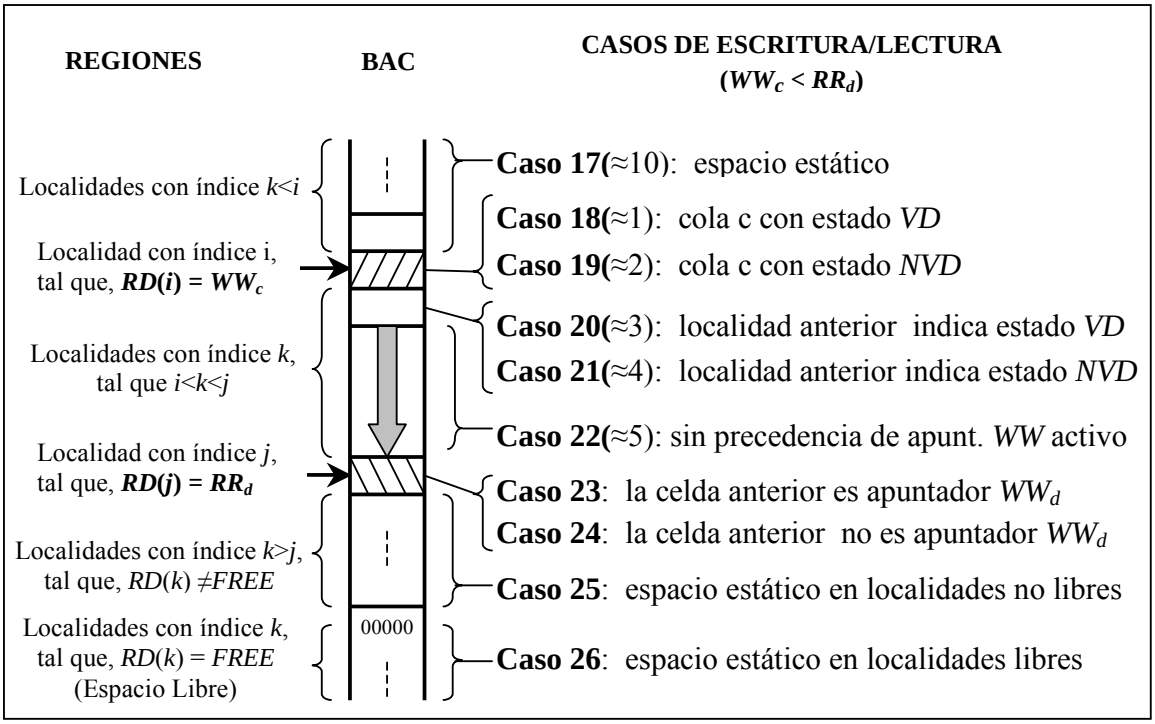


Figura 4.4 Casos derivados de una escritura/lectura con escritura en cola *c* y lectura en cola *d*, donde *c*<*d*. Por lo que, $WW_c < RR_d$.
(el símbolo "≈" denota un caso similar)

Casos 17 a 21

Condiciones de entrada:

1. Son las mismas que para los casos 10, 1, 2, 3, y 4, excepto que la activación es con *CWR* en lugar de *CW*.

Acciones de control :

1. Son las mismas que para los casos 10, 1, 2, 3 y 4, respectivamente.

Caso 22

Dadas las localidades i y j, tal que, $RD(i)=WW_c$ y $RD(j)=RR_d$. El caso 22 se presenta en las localidades con índice k, tal que, $i+1<k<j$. Como se muestra en la Figura 4.4, dichas localidades son desplazadas hacia abajo.

Las condiciones de entrada para cada celda $CD(k)$ son:

1. $RR_d \succ RD(k)$
2. $RD(k) \succ WW_c$
3. $RD(i+1) \neq WW_c$ (no es apuntador activo de escritura)

Las acciones de control son similares a las del caso 5:

1. Desplazamiento de dirección hacia abajo. El valor de $RD(k)$ toma el valor de $RD(k\text{-}1)$.
2. Desplazamiento de dato hacia abajo. El contenido de la celda $M(k\text{-}1)$ se escribe en $M(k)$.

Casos 23 y 24

En la localidad con índice j de la cola d, tal que $RD(j) = RR_d$, se presentan los casos 23 y 24. En el caso 23, la localidad anterior (con índice j-1), es un apuntador WW_d, (cola d con un solo dato). Por lo que, el valor RR_d de la celda $RD(j)$ desaparece y toma el valor WW_d de $RD(j\text{-}1)$, (desplazamiento hacia abajo). Además, como la cola d queda vacía, la celda $M(j)$ cambia a un valor VD.

En el caso 24, la cola d contiene más de un dato. Por lo tanto, el apuntador RR_d no desaparece y se mantiene en la celda $RD(j)$. Sólo se modifica la celda de almacenamiento $M(j)$, adquiriendo el valor de $M(j\text{-}1)$, (desplazamiento hacia abajo).

Condiciones de entrada:

1. $RD(j) = RR_d$,
2. $RD(j\text{-}1) = WW_d$, (para el caso 23).

   $RD(j\text{-}1) \neq WW_d$, (para el caso 24).

Acciones de control de la celda $CD(j)$ para el caso 23:

1. Desplazar dirección hacia abajo. La celda $RD(j)$ toma el valor de $RD(j\text{-}1)$,
2. Escribir estado *VD*, en la celda $M(j)$.

Acciones de control de la celda $CD(j)$ para el caso 24:

1. Desplazar dato hacia abajo. La celda $CD(j)$ adquiere el valor de $CD(j\text{-}1)$.

Casos 25 y 26

Estos dos casos se presentan en las localidades con índice $k>j$, tal que, $RD(j) = RR_d$. Tales localidades reciben el efecto resultante de un desplazamiento hacia abajo, provocado por la escritura, mas el desplazamiento hacia arriba, debido a la lectura. El resultado es que ambos desplazamientos se anulan. Por lo tanto, el control de estas localidades no realiza ninguna operación.

Las condiciones de entrada para el caso 25 son:

1. $RD(k) \succ RR_d$,
2. $RD(k) \neq FREE$.

Las condiciones de entrada para el caso 26 son:

1. $RD(k) = FREE$.

La acción de control en ambos casos es:

1. Ninguna operación.

4.3.4 Casos Derivados de una Operación de Lectura/Escritura.

Las operaciones de lectura/escritura se activan con la señal de control *CRW*. La lectura y escritura están comandadas por RR_c y WW_d, respectivamente. La lectura se realiza en una localidad con índice *i* de la cola *c* y la escritura en una localidad con índice *j* de la cola *d*, tal que $c<d$, por lo que $i<j$. Así, el *BAC* nuestra las 6 regiones siguientes:

1. Localidades con índice $k<i$,
2. Localidad con índice *i*, tal que, $RD(i) = RR_c$
3. Localidades con índice *k*, tal que, $i<k<j$,
4. Localidad con índice *j*, tal que, $RD(j) = WW_d$
5. Localidades con índice $k>j$, tal que $RD(k) \neq FREE$
6. Localidades con índice $k>j$, tal que $RD(k) = FREE$

En la Figura 4.5 se muestran las seis regiones del *BAC* durante una operación de lectura/escritura. En ella observamos que los casos 27, 28, 29, 30, y 31 son similares a los casos 12, 13, 14, 11, y 15, respectivamente. Esta similitud aparece en las dos primeras regiones del *BAC* y parte de la tercera. La similitud sólo está en las acciones de control y no en las condiciones de entrada. A continuación se describen los casos 27 a 31 haciendo referencia a sus casos similares.

Casos 27 a 30

Condiciones de entrada:

1. Son las mismas que las de los casos 12, 13, 14 y 11, respectivamente, excepto que la activación es con *CRW* en lugar de *CR*.

Acciones de control:

1. Son las mismas que para los casos 12, 13, 14 y 11, respectivamente.

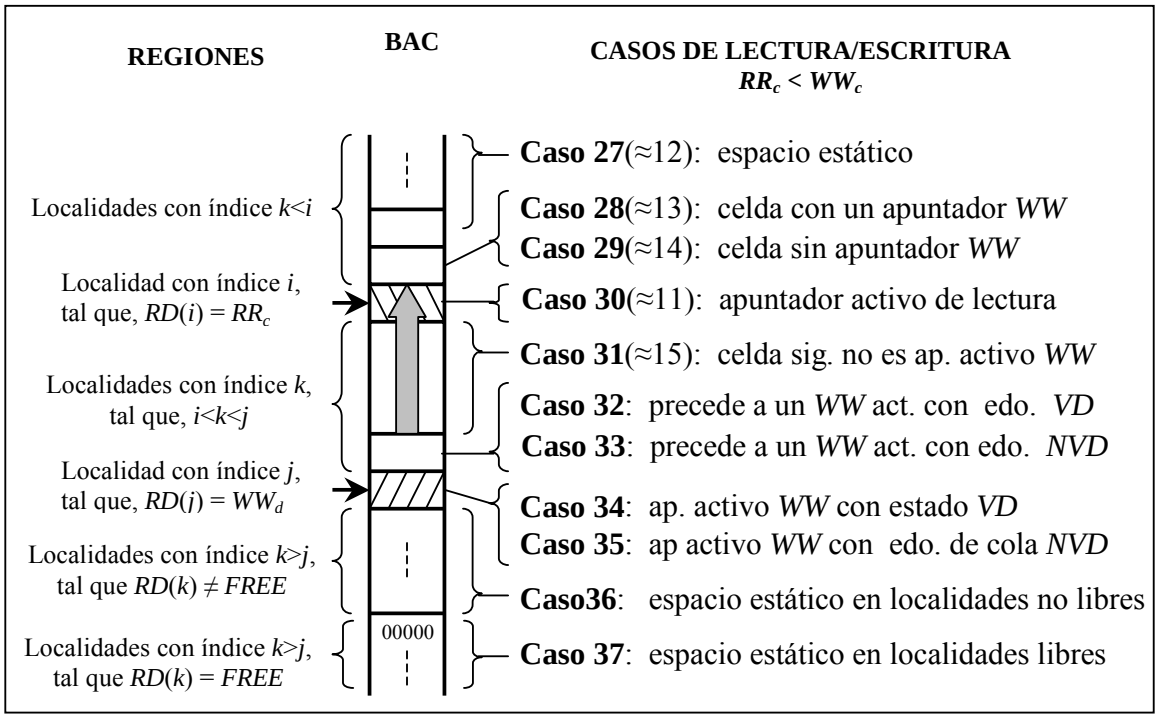


Figura 4.5 Casos derivados de una lectura/escritura con lectura en cola c y escritura en cola d, donde $c<d$. Por lo que, $RR_c < WW_d$. (el símbolo "≈" denota un caso similar)

Caso 31

Dadas las localidades i y j, tal que, $RD(i)=RR_c$ y $RD(j)=WW_d$. El caso 31 se presenta en las localidades con índice k, tal que, $i<k<j$-1. Como se muestra en la Figura 4.5, dichas localidades son desplazadas hacia arriba.

Las condiciones de entrada para cada celda $CD(k)$ son:

1. $WWd \succ RD(k)$
2. $RD(k) \succ RR_c$
3. $RD(j\text{-}1) \neq WW_d$ (no es apuntador activo de escritura)

Las acciones de control son similares a las del caso 15:

1. Desplazamiento de dirección hacia arriba. El valor de $RD(k)$ toma el valor de $RD(k+1)$.
2. Desplazamiento de dato hacia arriba. El contenido de la celda $M(k+1)$ se escribe en $M(k)$.

Casos 32 y 33

Estos casos se presentan en la localidad con índice j-1, tal que, $RD(j)=WW_d$ y $M(j)=VD$ (caso 32) o $M(j)=NVD$ (caso 33). Esta localidad es afectada por el desplazamiento hacia arriba que provoca la lectura. Por lo que, $RD(j\text{-}1)$ adquiere el valor de $RD(j)=WW_d$. Adicionalmente, en el caso 32, se escribe el valor *NVD* en $M(j\text{-}1)$. En el caso 33, se desplaza el valor de $M(j)$ hacia $M(j\text{-}1)$.

Las condiciones de entrada son:

1. $WW_d \succ RD(k)$,
2. $RD(k) \succ RR_c$,
3. $RD(k\text{-}1) = WW_d$,
4. $M(k\text{-}1) = VD$ (caso 32).
   $M(k\text{-}1) = NVD$ (caso 33).

Las acciones de control de la celda $CD(k\text{-}1)$ son:

1. Desplazar dirección hacia arriba. $RD(j\text{-}1)$ adquiere el valor de $RD(j)$.
2. Escribir estado *NVD* en la celda $M(j\text{-}1)$, (caso 32).
   Desplazar estado hacia arriba. $M(j\text{-}1)$ adquiere el valor de $M(j)$, (caso 33).

Casos 34 y 35

Estos casos se observan en la localidad con índice j, tal que, $RD(j) = WW_d$**.** Debido a la lectura que forma parte de esta operación paralela, el apuntador WW_d se desplaza hacia arriba y deja un espacio libre en el cual debe escribirse el dato D, sobre la celda $M(j)$.

Condiciones de entrada:

1. $RD(j) = WW_d$,
2. $M(j) = VD$ (caso 34).

   $M(j) = NVD$ (caso 35).

Acciones de control de la celda $CD(j)$:

1. Escribir dato D en la celda $M(j)$,
2. Crear un apuntador de lectura, RR_d, en la celda $RD(j)$, (caso 34),
3. Crear un apuntador de dato, DD_d, en la celda $RD(j)$, (caso 35).

Casos 36 y 37

Estos dos casos se presentan en las localidades con índice $k>j$, tal que, $RD(j) = WW_d$. De manera similar a los casos 25 y 26, estas localidades reciben el efecto resultante de un desplazamiento hacia arriba, provocado por la lectura, mas el desplazamiento hacia abajo, debido a la escritura. El resultado es que ambos desplazamientos se anulan. Por lo tanto, el control de estas localidades no realiza ninguna operación.

Las condiciones de entrada para el caso 36 son:

1. $RD(k) \succ WW_d$,
2. $RD(k) \neq FREE$.

La condición de entrada para el caso 37 es:

1. $RD(k) = FREE$.

La acción de control en ambos casos es:

1. Ninguna operación.

4.3.5 Casos Derivados de una Operación de Escritura-Lectura.

La operación de escritura-lectura sobre una cola *c*, solo puede realizarse si dicha cola no está vacía. Es decir, la cola *c* debe contener al menos un dato. Bajo esta condición, los comandos WW_c y RR_c mantienen una relación de valores tal que $WW_c \prec RR_c$. Esta relación es idéntica a la que existe en una operación de escritura/lectura, donde $WW_c \prec RR_d$ con $c = d$. Por lo tanto, la operación de lectura-escritura es activada también por el comando *CWR*.

Para analizar los casos derivados de una operación de escritura-lectura, solo bastará con modificar la Figura 4.4, de escritura/lectura, considerando que la escritura y lectura se realizan en la misma cola. Es decir, $c=d$. Con esta modificación se observan también 6 regiones del *BAC*, que son virtualmente las mismas que para escritura/lectura. Debido a que la escritura-lectura se activa con *CWR*, los casos que se presentan en esta operación son idénticos a algunos que aparecen en la de escritura/lectura (casos 17, 19, 21, 22, 24, 25 y 26), excepto por un nuevo caso. Ello se muestra en la Figura 4.6., con los casos derivados de una operación de escritura-lectura. En la figura se observa que el único caso diferente de los ya estudiados es el 38. Por tal razón, solo describiremos este caso. Los otros ya han sido descritos en secciones anteriores.

Caso 38

El caso 38 se presenta en la localidad con índice *j* de la cola *c*, tal que, $RD(j) = RR_c$ y $RD(j\text{-}1) = WW_c$. Es decir, los apuntadores activos WW_c y RR_c están en localidades contiguas. Este caso se caracteriza porque las operaciones de escritura y lectura se realizan en la misma cola y en la misma localidad. Cuando ocurre este caso, tal como se explicó en la sección 3.5.3 del capítulo anterior, el registro de direcciones *RD* no cambia. El único cambio ocurre en la celda *M*(*j*) con la lectura y escritura sobre la misma localidad.

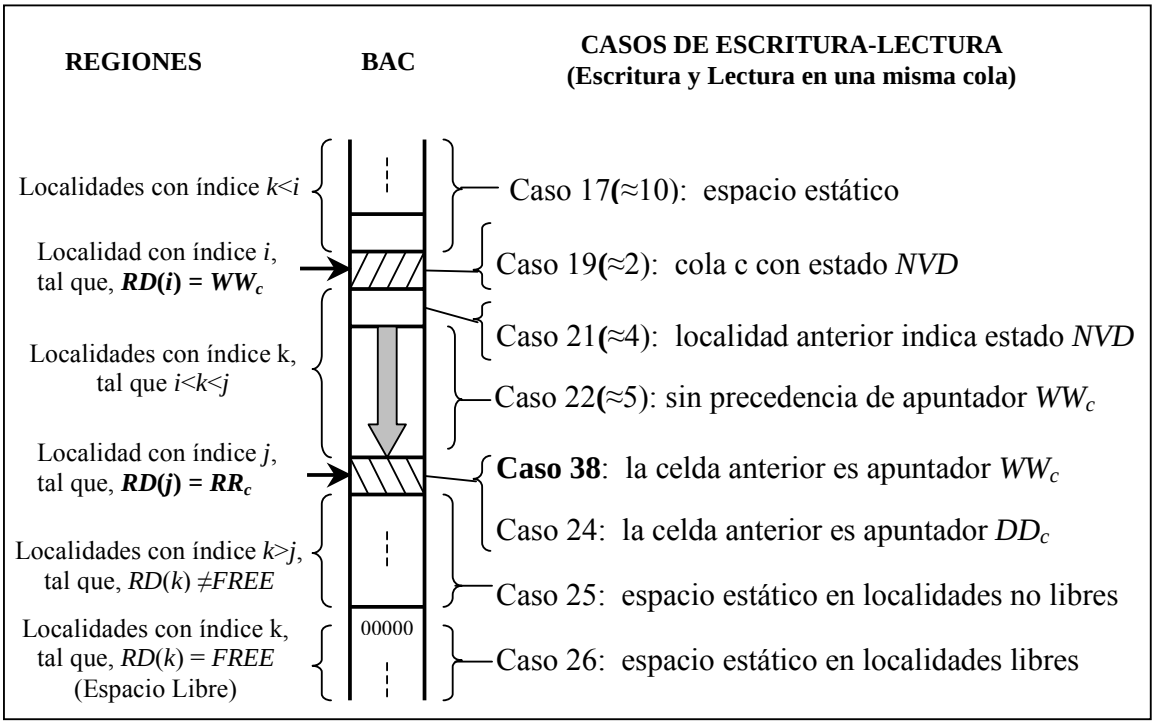


Figura 4.6 Casos derivados de una escritura-lectura. La escritura y la lectura se realizan sobre una misma cola c.

Condiciones de entrada:

1. $RD(j) = RR_c$,
2. $RD(j\text{-}1) = WW_c$.

Las acciones de control de la celda *CD*(*j*) son:

1. lectura del dato que está en *M*(*j*) y escritura del nuevo dato en *M*(*j*)

4.4 Conclusiones

Hasta ahora hemos hecho una descripción de las condiciones particulares en que puede encontrarse cada celda o grupo de celdas de control, de acuerdo con su posición y la operación fundamental que realiza el *BAC*. Dichas condiciones particulares generan acciones de control, las cuales se han descrito en función de las operaciones primitivas que cada localidad del *BAC* puede realizar. En el siguiente capítulo veremos cómo las acciones que generan las celdas del módulo de control, se traducen en un conjunto de señales que sincronizan la ejecución de las operaciones primitivas. Observaremos que la traducción de operaciones primitivas en señales de control depende de la arquitectura que se utilice en el diseño del circuito.

Capítulo 5

Implementación y Pruebas del Buffer AutoCompactante (*BAC*)

5.1 Introducción

En este capítulo describiremos los detalles del diseño lógico del *Buffer AutoCompactante*, nuestro *BAC*. Asimismo, presentaremos los resultados de pruebas de simulación lógica realizadas sobre sus partes más relevantes.

Visualizando a la arquitectura de *BAC* como un sistema digital típico, podremos distinguir sus dos partes fundamentales: el circuito de control, constituido por el módulo *CD*, y la trayectoria de datos, constituida por los módulos *RD* y *M*. Véase la Figura 4.1.a. Así, el módulo *CD* genera comandos de ejecución de operaciones primitivas, y los módulos *RD* y *M* realizan tales operaciones.

Es preciso decir que las celdas del módulo de control *CD*, son el corazón de nuestro multicitado *Buffer AutoCompactante*. Si describimos sus detalles de diseño y funcionamiento, estaremos describiendo el funcionamiento básico del *BAC*. Primero mostraremos esquemáticamente cada uno de los tres módulos que lo conforman y posteriormente daremos los detalles de su descripción con un lenguaje formal, conocido como VHDL (*VHSIC Hardware Description Language*). Utilizaremos para ello una

herramienta computacional, de la compañía *Xilinx*, que soporta las fases de desarrollo de circuitos digitales. Finalmente, mostraremos los resultados de la simulación lógica del *BAC* para probar su comportamiento funcional.

5.2 Celda de Control Distribuido

Retomando la Figura 4.1b, que muestra la estructura de una localidad del *BAC*, recordamos que cada localidad del *BAC* está constituida por: una celda del registro de direcciones *RD*, una celda del módulo de almacenamiento de datos *M*, y una celda del módulo de control *CD*. Si analizamos el modelo concreto al nivel de operaciones primitivas que fue desarrollado en el capítulo anterior, podemos inferir los tipos de circuitos que se requieren en cada celda para llevar a cabo dichas operaciones. Así, cada celda del registro *RD* es implementada mediante un multiplexor de direcciones *MRD* para desplazar su contenido hacia arriba o hacia abajo. Ese mismo circuito debe poder crear apuntadores de datos y direcciones cuando sea necesario [62], [74]. Las señales de entrada y salida utilizadas entre una celda *RD*(*i*) y otra *CD*(*i*) son:

HRD.- Habilita registro de direcciones

S0RD.- Selecciona desplazamiento de dirección, (0= hacia abajo, 1=hacia arriba)

CADD.- Crea un apuntador de dato

CARR.- Crea un apuntador de lectura

RD[7:0].- Valor del registro de dirección

Por otra parte, cada celda del módulo de almacenamiento *M* debe permitir la escritura y/o lectura de datos y del estado de una cola. También debe permitir el desplazamiento de los datos. Esta celda se implementa con: un multiplexor de datos *MM*, un registro auxiliar de datos *RXD*, un registro principal de almacenamiento de datos *RAD* y un dispositivo de tercer estado que habilita la salida de datos del *BAC*. El multiplexor *MM* es para las

operaciones de desplazamiento hacia arriba o hacia abajo. El registro *RXD* almacena temporalmente un dato D_2 a escribir, mientras que se realiza la posible lectura de un dato D_1 en esa misma localidad. Las señales utilizadas son:

S0M, S1M.- Desplazan dato en el *MM*, (0,0 = abajo; 0,1 = selecciona dato; 1,1 arriba)

HM.- Habilita la memoria *M*

ENVD.- Escribe un estado no-vacío.

EVD.- Escribe un estado vacío

HLM.- Habilita lectura de memoria

SS.- Estado de la cola (1= no-vacío, 0 = vacío)

Como se ha mencionado en capítulos anteriores, las celdas de control toman decisiones a partir de conocer las características de sus dos localidades vecinas, tales como, el estado de la cola y/o el tipo de apuntador asociado a dichas localidades. Para ello, las celdas de control envían o reciben las siguientes señales:

CAW.- La celda anterior es un apuntador *WW*.

CAR.- La celda anterior es un apuntador *RR*.

CAWA.- La celda anterior es un apuntador *WW* activo.

SSCA.- Estado de la celda anterior

IASS.- Indica a la celda anterior el estado de la cola

IAWA.- Indica a la celda anterior que la actual en apuntador de escritura activo.

IARA.- Indica a la celda anterior que la actual es apuntador de lectura activo

CPW.- La celda presente es un apuntador *WW*, (señal interna).

CSWA.- La celda siguiente es un apuntador *WW* activo.

CSRA.- La celda siguiente es un apuntador *RR* activo.

SSCS.- Estado de la celda siguiente.

ISWA.- Indica a la celda siguiente su condición de apuntador *WW* activo.

ISSS.- Le envía el estado de la cola a la celda siguiente.

WO.- Le indica a la siguiente celda su condición de apuntador de escritura.

RO.- Le indica a la siguiente celda su condición de apuntador de lectura.

En la Figura 5.1 se muestra, de manera esquemática, el conjunto de señales de cada celda de control. El sentido de la flecha de cada señal indica su atributo de entrada/salida.

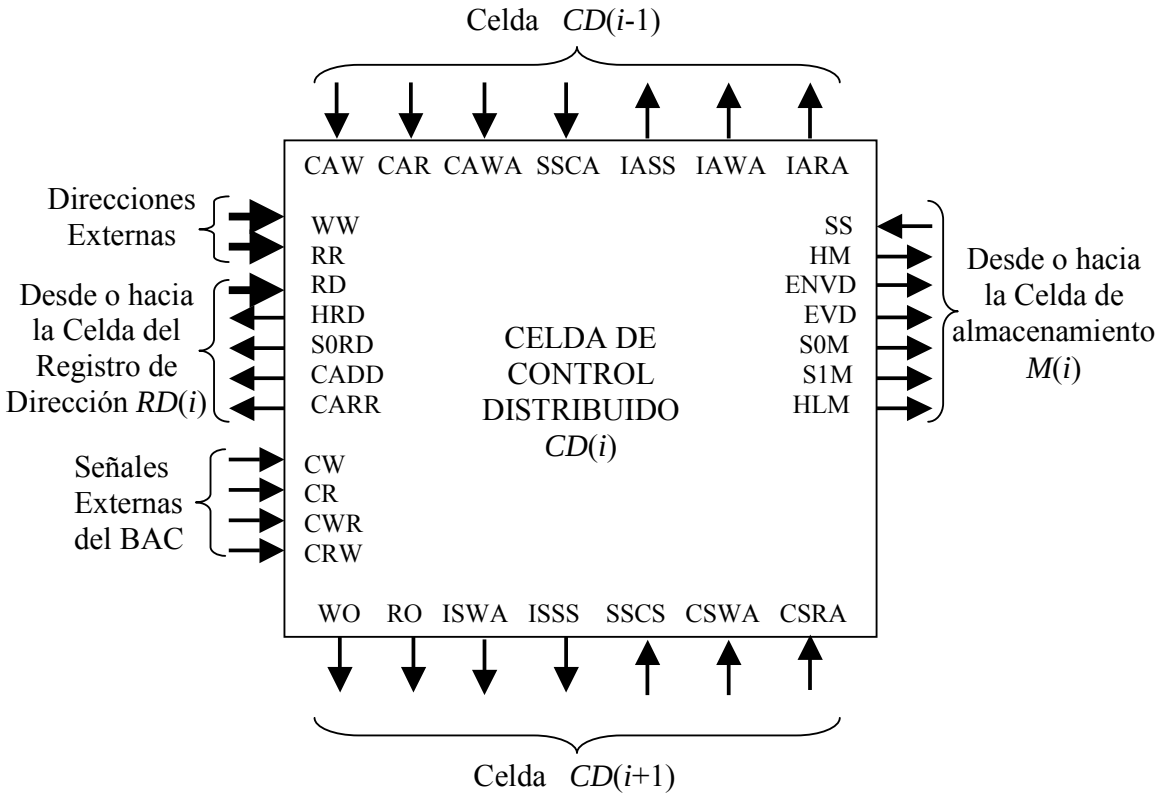


Figura 5.1 Diagrama esquemático de la celda de control que muestra sus señales de entrada y salida.

5.2.1 Comandos de Control

Tomando como referencia el modelo del capítulo anterior y la forma en que implementamos la trayectoria de datos, definimos ahora los comandos que una celda de

control debe enviar y recibir en cada caso. Para tal fin, presentamos dos tablas que sintetizan el modelo de operaciones primitivas. La Tabla 5.1 contiene los 16 casos que se derivan de las operaciones fundamentales de escritura y de lectura. La Tabla 5.2 nos muestra los 22 casos que se derivan de las operaciones paralelas.

		Tabla 5.1 Casos Derivados de Operaciones Simples															
		E s c r i t u r a										**L e c t u r a**					
Número de Caso		***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***16***
Condiciones de Entrada	**CW**	1	1	1	1	1	1	1	1	1	1						
	CR											1	1	1	1	1	1
	CWR																
	CRW																
	RD=FREE					0	1	1	1	1	0		0	0	0		1
	RD=WW	1	1														
	RD=RR											1					
	RD>WW			1	1	1											
	RD>RR															1	
	RD<WW										1						
	RD<RR												1	1	1		
	CAW						1			0							
	CAR							1		0							
	CPW													1	0		
	SS	0	1														
	CSWA																
	CSRA												0	1	1		
	CAWA			1	1	0			1								
	SSCS																
	SSCA			0	1												
Comandos de Salida	**HRD**			1	1	1	1	1	1			1				1	
	HM	1		1	1	1	1	1	1			1				1	
	ENVD	1															
	EVD													1			
	S0RD					0	0	0				1			1	1	
	CADD				1												
	CARR			1													
	S0M			1	1	0	0	0	0			1				1	
	S1M			0	0	0	0	0	0			1				1	
	HLM											1					
	IAWA																
	IARA											1					
	ISWA	1	1														
	IASS																
	ISSS	0	1														

Los números de columna, en las Tablas 5.1 y 5.2, indican los números de casos, acorde con la descripción hecha a detalle en el capítulo 4. Cada columna de la tabla nos indica, con un cero (‘0) o un uno (‘1’), el valor en que se encuentran las señales de entrada y salida de nuestra celda de control, etiquetadas como condiciones de entrada y comandos de salida. Los valores que no están definidos en la tabla son indistintos.

Tabla 5.2 Casos Derivados de Operaciones Paralelas

		Escritura/Lectura										Lectura/Escritura											*
		17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
CONDICIONES DE ENTRADA	**CW**																						
	CR																						
	CWR	1	1	1	1	1	1	1	1	1	1												1
	CRW											1	1	1	1	1	1	1	1	1	1	1	
	RD=FREE	0									1	0	0	0								1	
	RD=WW		1	1															1	1			
	RD=RR							1	1						1								1
	RD>WW				1	1	1														1		
	RD>RR									1						1	1	1					
	RD<WW	1														1	1	1					
	RD<RR				1	1	1					1	1	1									
	CAW							1	0														
	CAR																						
	CPW												1	0									
	SS		0	1															0	1			
	CSWA															0	1	1					
	CSRA											0	1	1									
	CAWA				1	1	0																1
	SSCS																0	1					
	SSCA				0	1																	
COMANDOS DE SALIDA	**HRD**				1	1	1	1				1	1	1	1	1	1	1	1	1			
	HM		0		1	1	1	1	1			1	1	1	1	1	1	1	1	1			1
	ENVD		0														1						
	EVD							1					0										
	S0RD						0	0						0	0	1	1	1					
	CADD					1														1			
	CARR				1							1	1	1	1				1				
	S0M				1	1	0		0			1	1	1	1	1		1	1	1			1
	S1M				0	0	0		0						0	1		1	0	0			0
	HLM							1	1						0								
	IAWA												1	1	1	1			1	1			1
	IARA														0								
	ISWA		0	0																			
	IASS																		0	1			
	ISSS		0	0																			

*** Escritura-Lectura**

DIRECCIONES VÁLIDAS DEL RD		
Dirección		Tipo de Dirección
(Hex)	(Bin)	
02	00010	Apuntador de Escritura WW_1
03	00011	$Dato_1$ DD_1
04	00100	Apuntador de Lectura RR_1
0A	01010	Apuntador de Escritura WW_2
0B	01011	$Dato_2$ DD_2
0C	01100	Apuntador de Lectura RR_2
12	10010	Apuntador de Escritura WW_3
13	10011	$Dato_3$ DD_3
14	10100	Apuntador de Lectura RR_3
1A	11010	Apuntador de Escritura WW_4
1B	11011	$Dato_4$ DD_4
1C	11100	Apuntador de Lectura RR_4
00	00000	Espacio vacío ϕ

Orden creciente

Tabla 5.3 Valores de los apuntadores del registro de direcciones *RD*

Los comandos de salida de cada celda de control determinan las acciones que realizarán sus correspondientes módulos *RD* y *M*, después de un pulso de reloj. La definición de tales acciones es complementada por las señales que recibe de las celdas de control adyacentes.

Para verificar las condiciones de entrada es necesario hacer comparaciones entre los valores de dirección de escritura *WW*, de dirección de lectura *RR* y los valores de dirección de cada celda. Por lo que, se ha definido un patrón de valores de direcciones válidas para cada cola del *BAC*. Las celdas de control toman estos valores, mostrados en la Tabla 5.3, como una referencia de espacio para tomar sus decisiones.

5.2.2 Implementación de las Tablas de Casos

Utilizamos las tablas de casos, 5.1 y 5.2, como base para describir en VHDL las celdas de control del *BAC*. Con el objeto de confirmar un correcto funcionamiento, describimos a la celda de control de dos maneras. En la primera se interpreta cada caso con dos vectores: uno de entrada y otro de salida. Si se cumplen las condiciones del primer vector (entrada), entonces se genera un segundo vector (salida), cuyos elementos se programan

para convertirse en comandos de salida. La programación de esta primera interpretación se muestra en el Apéndice B.

Veamos en la Tabla 5.1 el caso número uno, correspondiente a una localidad con índice *i* apuntada por una dirección de escritura *WW*. Este caso se presenta cuando se va a escribir sobre una cola vacía (con estado *VD*). En la columna etiquetada con el número uno, la señal *CW* (comando de escritura) tiene un valor de '1'. También, se tiene la condición de que el valor de dirección asociado a esta celda debe ser igual al valor que tiene el apuntador de escritura (*RD*=*WW*). Además, el estado de la cola, indicado por la señal SS que viene de la celda de almacenamiento, tiene un valor de '0'. Ante estas condiciones de entrada, la celda de control prepara a las celdas *RD* y *M* para realizar las operaciones correspondientes al caso uno, cuando aparezca un pulso de reloj. Por lo tanto, las señales de salida HM, ENVD e ISWA toman el valor '1', y la señal ISSS es '0'. Con HM se habilita a la celda de almacenamiento para escribir un estado NVD, mediante la señal ENVD. La salidas ISWA e ISSS le indican a la localidad siguiente (con índice i+1), su condición de apuntador de escritura activo con estado de cola vacía.

Podemos observar en la Figura 5.2, la codificación en VHDL correspondiente al caso uno descrito.

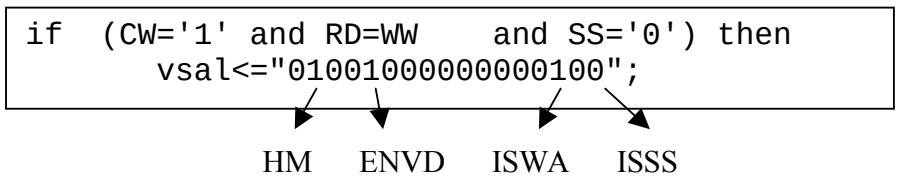


Figura 5.2. Código en VHDL que describe la función de la celda de control para el caso 1 del BAC.

Una segunda manera de implementar las tablas de casos, fue a través de una interpretación clásica de una tabla de verdad. Por lo que, cada una de las salidas se

describió como una suma de productos. Ambas interpretaciones fueron simuladas. La primera se usó para probar exclusivamente a una celda de control *CD*(*i*). Y, la segunda se integró a las descripciones de las celdas de dirección *RD*(*i*), y de almacenamiento *M*(*i*), para probar en forma conjunta varias localidades del *BAC*. En el Apéndice C se muestra el código de una localidad completa. En dicho código se describe de manera independiente cada uno de los comandos de salida, considerando a las dos tablas de casos, 5.1 y 5.2, como una sola tabla.

5.3 Pruebas de Simulación Lógica

Durante las primeras pruebas de nuestro diseño lógico, obtuvimos una gráfica de tiempos de las señales de entrada y salida de la celda de control. Esta gráfica comprende los 38 casos que presenta el *BAC* y la hemos dividido en dos partes. En la Figura 5.3 se muestran las señales con operaciones fundamentales simples. En ella se podrá observar su correspondencia con la Tabla 5.1, que sintetiza las operaciones primitivas necesarias para escrituras y lecturas simples. En la Figura 5.4 se muestran las estradas y salidas para los casos derivados de operaciones paralelas. En dicha figura podemos observar, también, la correspondencia que existe con la Tabla 5.2, que sintetiza los casos derivados de operaciones paralelas.

Por ejemplo, observemos el caso 20 de la Figura 5.4, correspondiente a una operación de escritura/lectura, la cual se activa con la señal *CWR*=’1’. Este caso se presenta en la localidad con índice *i*+1, tal que *RD*(*i*) = *WWc*, donde *WWc* es el comando de escritura sobre una cola *c* que está vacía. En la celda de almacenamiento *M*(*i*+1) se almacenará el dato a escribir. Y, en la celda *RD*(*i*+1) deberá crearse un apuntador de lectura *RRc*. Para tal fin, la localidad con índice *i*+1 deberá saber que la localidad con índice *i* corresponde a un apuntador activo de escritura, lo cual se indica con CAWA = ‘1’. Asimismo, debe saber

que la cola *c* está vacía, lo cual se indica con SSCA = ‘0’. Dadas estas condiciones de entrada, la celda de control *CD*(*i*+1) genera las señales HRD = ‘1’ y CARR = ‘1’, habilitando, así, a la celda *RD*(*i*+1) para crear un apuntador de lectura. Asimismo, la celda de control genera las señales HM = ‘1’, S0M = ‘1’ y S1M = ‘0’ para que en la celda de almacenamiento *M*(*i*+1) se escriba un dato.

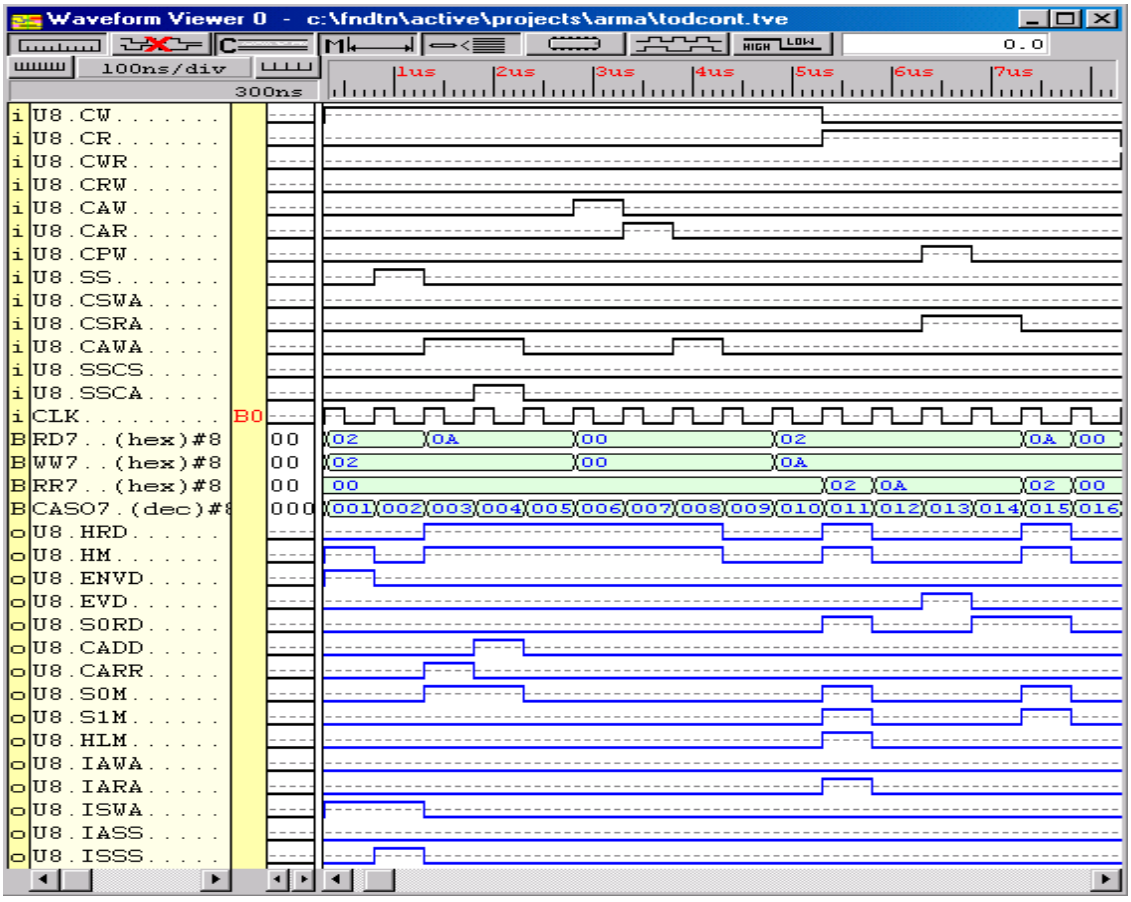


Figura 5.3. Diagrama de tiempos de las señales de entrada y salida de una celda de control, para los casos derivados de operaciones simples.

Siguiendo las formas de onda, de las Figuras 5.3 y 5.4, y analizándolas de una manera similar al ejemplo anterior, podemos verificar el funcionamiento correcto de la celdas de control en los 38 casos sintetizados en las tablas 5.1 y 5.2.

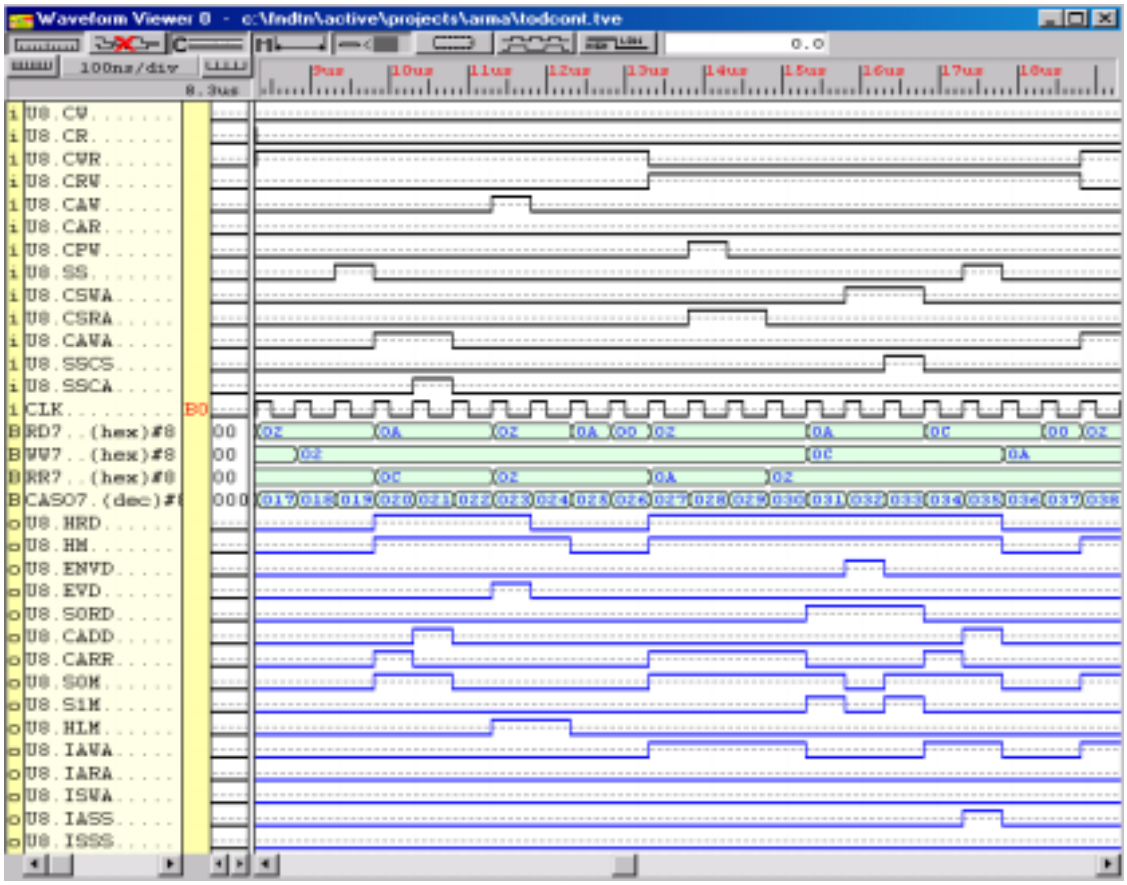


Figura 5.4. Diagrama de tiempos de las señales de entrada y salida de una celda de control, para los casos que se derivan de operaciones paralelas.

Ahora veremos las pruebas de simulación lógica, utilizando el código del Apéndice C, para una localidad completa. Como lo habíamos mencionado, la forma de codificar la celda de control en el Apéndice C es diferente a la que utilizamos para generar las Figuras 5.3 y 5.4. Sin embargo, fue útil para verificar el funcionamiento al nivel de localidad. Dicho código se integró dentro de una macro llamada LOCALBAC la cual se utilizó para configurar un *BAC* de ocho localidades y con éste, realizar las pruebas.

En la Figura 5.5 presentamos el diagrama de tiempos obtenido durante la inicialización del *BAC*. Para inicializar nuestro *Buffer AutoCompactante*, utilizamos un algoritmo

denominado PRUEBAC que pone en ‘ceros’ a todas las celdas del registro de direcciones *RD* y a las del espacio de almacenamiento de datos *M*. Posteriormente, inicializa las primeras cuatro localidades de acuerdo con la descripción de la sección 3.3.3 del tercer capítulo. Es decir, crea los apuntadores WW_1, WW_2, WW_3 y WW_4 en las primeras cuatro celdas del registro *RD*.

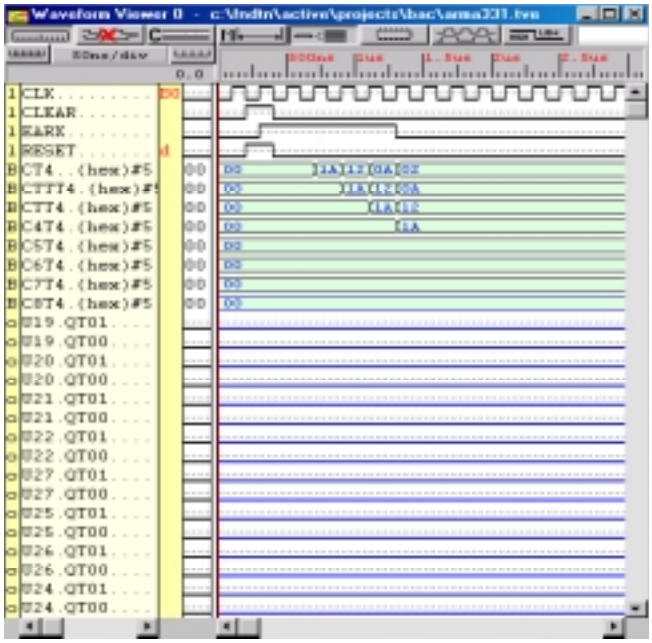


Figura 5.5 Inicialización del *BAC*

Con el objetivo de probar todas las condiciones necesarias que demuestren la correcta funcionalidad del *BAC*, en las pruebas se utilizaron los mismos ejemplos del Capítulo 3, ajustándolos a un *BAC* de ocho localidades. Con estos ejemplos probamos todas las combinaciones posibles de las operaciones fundamentales simples y paralelas sobre colas vacías y con información. De esta manera, creamos y probamos los 38 casos posibles en que se encuentran las localidades del *BAC* y que se derivan de sus cinco operaciones fundamentales.

Las primeras dos pruebas que presentamos son: una con operación de escritura simple, la cual se apega a las condiciones del ejemplo 3.4.1; y otra con operación de escritura/lectura, de acuerdo con las condiciones del ejemplo 3.5.3. Estas pruebas las hemos descrito con mayor detalle para facilitar la interpretación de sus resultados. De una manera similar, es posible interpretar los resultados de las pruebas restantes.

Siguiendo el ejemplo 3.4.1 que dimos en el capítulo 3, supongamos un *buffer* con ocho localidades después de haberse inicializado como en la Figura 5.5. Supóngase también, la aplicación de un comando de escritura en la cola tres con un valor de dato igual a 3, es decir, (WW_3, D) = (12H, "11").

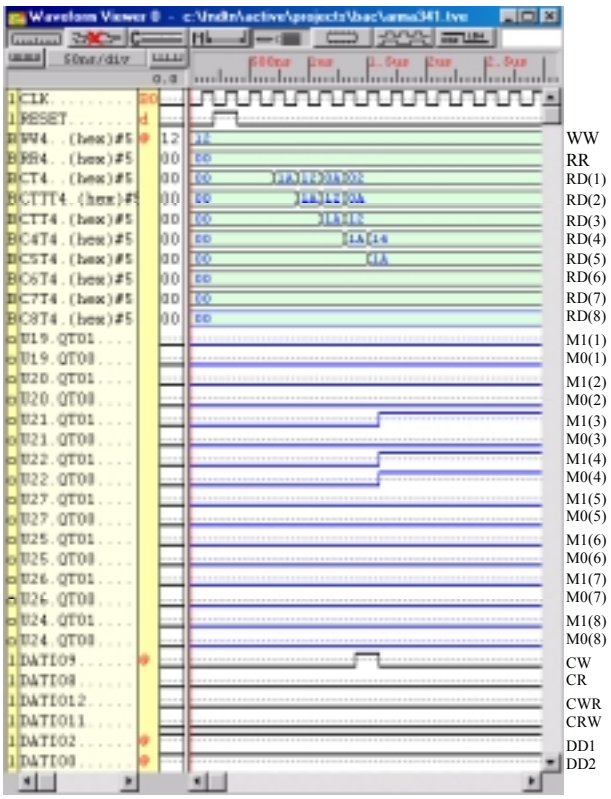


Figura 5.6 Prueba del BAC con una operación de escritura en una cola vacía.

De la Figura 5.6, podemos observar que en la celda cuatro del registro de direcciones (*RD*(4)) se ha insertado un valor 14(hex), correspondiente a un apuntador de lectura RR_3. Asimismo, podemos ver que la celda cuatro del registro de almacenamiento de datos (*M*1(4), *M*0(4)), representada por U22.QT01 y U22.QT00, tiene un valor de dato igual a 3 ("11"). También, podemos ver en el registro de direcciones que el valor 1AH, correspondiente al inicio de la cola 4, se desplazó hacia la celda 5. Así, la única cola con información es la cola 3, por lo que su estado, representado por el bit U21.QT01= M1(3)='1' es el de una cola que no está vacía. El nivel alto que se presenta en la señal DATI09 corresponde al comando de escritura *CW* y se activa para habilitar una operación de escritura.

La Figura 5.7 muestra los resultados obtenidos en el *BAC* con una operación paralela de escritura/lectura, con condiciones similares a las del ejemplo 3.5.3, dadas en el tercer

capítulo. La prueba consiste en aplicar una operación de escritura sobre la cola uno, que tiene un dato y, simultáneamente, realizar una operación de lectura sobre la cola tres, que también tiene un dato. Las condiciones de esta prueba son similares a las del ejemplo 3.5.3, dadas en el tercer capítulo de la tesis.

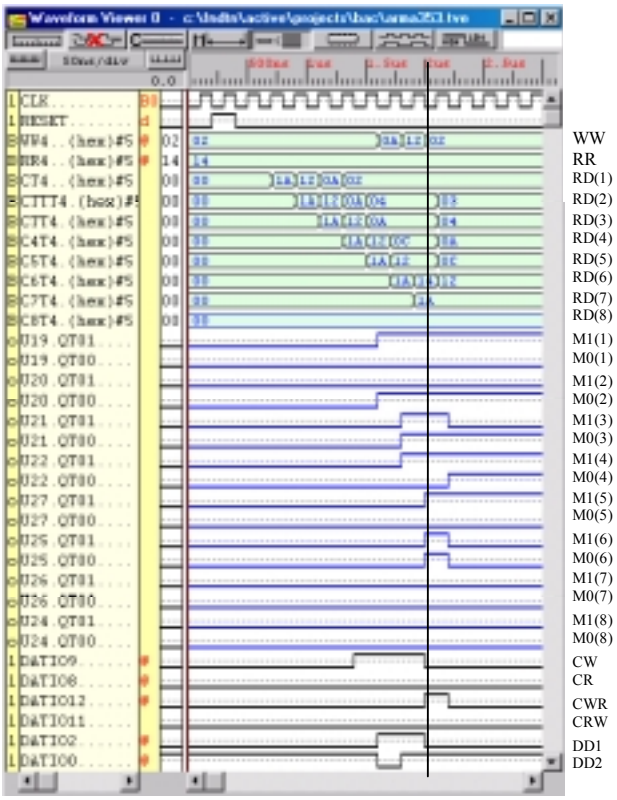


Figura 5.7. Prueba del BAC con una operación paralela que escribe en la cola 1 y lee en la cola tres. Al iniciar la operación, cada cola tiene un dato.

Observando la Figura 5.7., vemos que el comando *CW* se activa durante tres ciclos de reloj para inicializar las condiciones del ejemplo. Para crear las condiciones del ejemplo, se escribe un dato en cada una de las primeras tres colas. En la cola 1 (con apuntador WW_1= 02H), se escribe un dato de dos bits (D_1, D_2) con valor igual a '01'. En la cola 2 (con apuntador WW_2=0AH), se escribe un dato igual a '10'. Y, en la cola 3 (con apuntador WW_3=12H), se escribe un dato igual a '11'. Enseguida se desactiva *CW* y se activa el comando *CWR* para realizar la operación de escritura/lectura. En ese instante, los registros *M* y *RD* del *BAC* se encuentran como se muestra en la Figura 5.8. En dicha figura se indica el significado de la información que contiene cada elemento del arreglo *M*, así como del registro de direcciones *RD*.

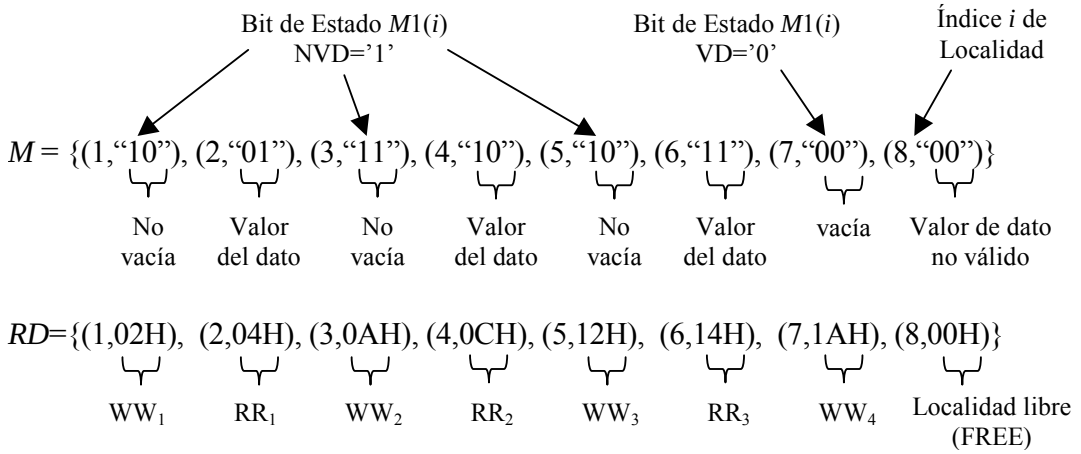


Figura 5.8. Estado de los registro *M* y *RD* del *BAC*, antes de la operación de escritura/lectura, de acuerdo con el ejemplo 3.5.3 del capítulo tres.

Después de la operación de escritura/lectura, los registros *M* y *RD* quedan como sigue:

M = {(1,"10"), (2,"01"), (3,"01"), (4,"11"), (5,"10"), (6,"00"), (7,"00"), (8,"00")}

RD = {(1,02H), (2,03H), (3,04H), (4,0AH), (5,0CH), (6,12H), (7,1AH), (8,00H)}

Lo cual podemos verificar en la Figura 5.7. Por lo que, en estas condiciones, el *BAC* se comporta correctamente. Es decir, la operación de escritura deja a la cola 1 con dos datos, y la operación de lectura deja vacía a la cola 3.

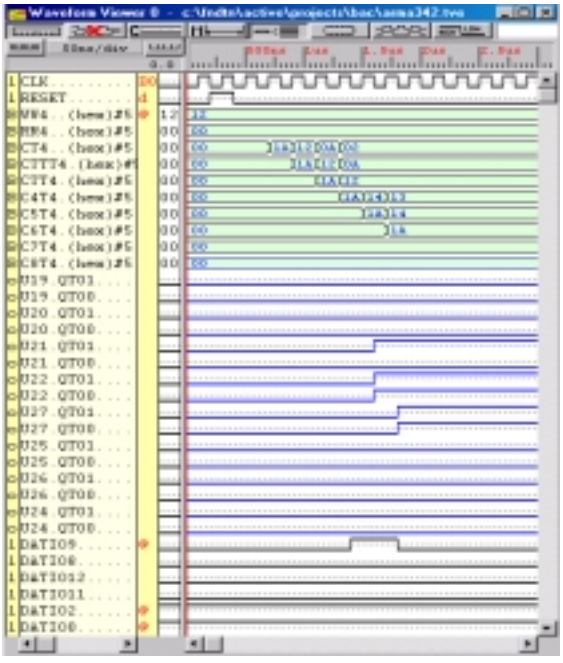


Figura 5.9. Escritura en cola tres con información

En la Figura 5.9 se muestran las formas de onda de las principales entradas y salidas del *BAC*, bajo condiciones de prueba similares a las del ejemplo 3.4.2. La prueba consiste en realizar una operación de escritura sobre una cola que

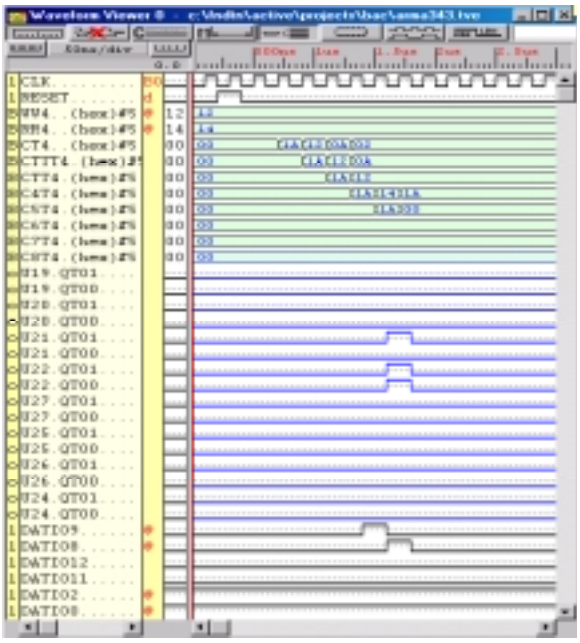


Figura 5.10 Lectura en cola con un dato

contiene información. La escritura se realiza sobre la cola 3, que contiene un dato.

La Figura 5.10 muestra la prueba del *BAC* con una operación de lectura sobre una cola con información. Se lee sobre la cola 3 que tiene un dato. Las condiciones de prueba son similares a las del ejemplo 3.4.3 del tercer capítulo. En esta prueba podemos observar que, después de la lectura del único dato existente sobre la cola tres, la cola queda vacía. Es decir, desaparece el apuntador RR_3 y el *BAC* queda compactado. Las cuatro colas quedan en estado de cola vacía.

Ahora, en la Figura 5.11, mostramos los resultados de una operación de lectura sobre una cola que contiene dos datos. Las condiciones de prueba son similares a las del ejemplo 3.4.4 que se muestra en el tercer capítulo. Después de escribir dos datos en la cola tres, se realiza una lectura en esa misma cola. Como resultado, desaparece en el registro de direcciones, el apuntador de dato DD_3, y solamente queda un dato apuntado por RR_3. La cola

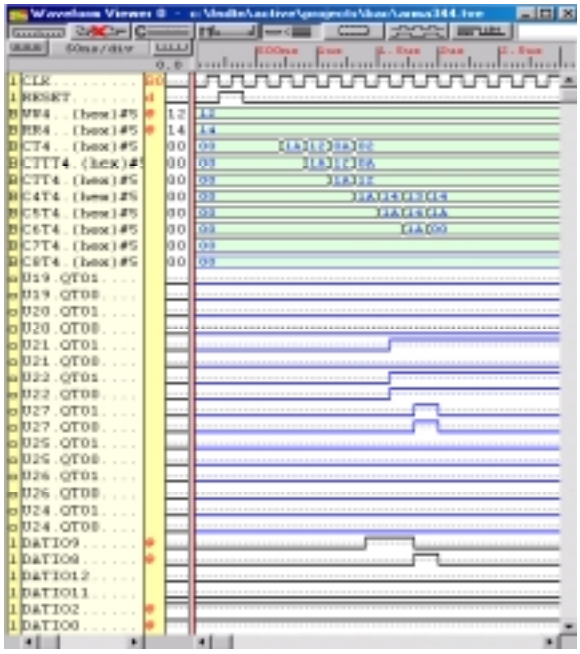


Figura 5.11 Lectura en cola con más de un dato

Figura 5.12. Escritura en cola sin información y lectura en cola con un dato.

corresponden al ejemplo 3.5.1 del capítulo tres.

En la Figura 5.13 tenemos el resultado de una operación paralela con una escritura en la cola uno, estando vacía, y una lectura sobre la cola tres, que tiene dos datos. Las condiciones de prueba corresponden a las utilizadas en el ejemplo 3.5.2 del capítulo 3.

Con la Figura 5.14 mostramos la prueba de una escritura/lectura, con escritura sobre la cola 1, que contiene un tres permanece en estado no vacío. Dicho estado se representa por el valor "10" en la celda número tres del registro *M*.

Las siguientes figuras nos muestran los resultados obtenidos de pruebas realizadas con operaciones paralelas del tipo escritura/lectura.

Los resultados de la Figura 5.12. muestran el comportamiento del *BAC* con una operación de escritura sobre la cola uno (estando vacía) y, simultáneamente, una lectura en la cola tres (teniendo un dato). Estas condiciones de prueba

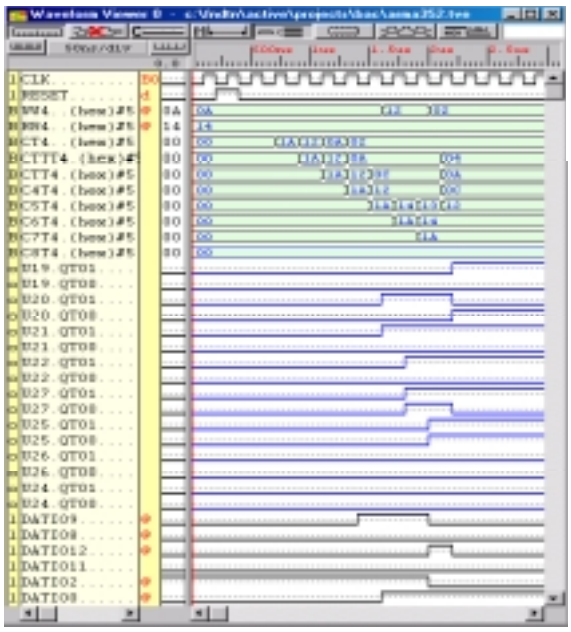


Figura 5.13 Escritura en cola sin información y lectura en cola con más de un dato.

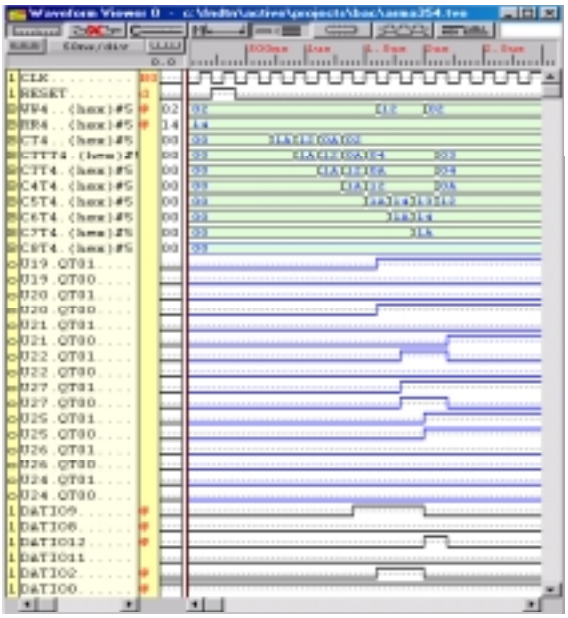


Figura 5.14 Escritura en cola con información y lectura en cola con más de un dato, y lectura sobre la cola 3 que contiene 2 datos. Las condiciones de esta prueba son similares a las del ejemplo 3.5.4 explicado en el capítulo tres. La única variante es que la cola dos, que no se usa en la prueba, se encuentra vacía.

En las siguientes cuatro figuras, 5.15. a 5.18, presentamos los resultados obtenidos durante las pruebas con operaciones de lectura/escritura. Y, en las Figuras 5.19. y 5.20 se muestran los resultados de las operaciones de escritura-lectura sobre una misma cola. Particularmente, la Figura 5.20 nos muestra el resultado de la escritura-lectura en una misma localidad. En todas ellas, sustituimos la información del registro de almacenamiento *M*, mostrada en las pruebas anteriores, por las salidas de datos de cada localidad del *BAC*. Esto es con el fin de observar la salida de los datos cuando son leídos. Tales salidas están diseñadas para mantenerse en alta impedancia, excepto cuando sacan un dato leído.

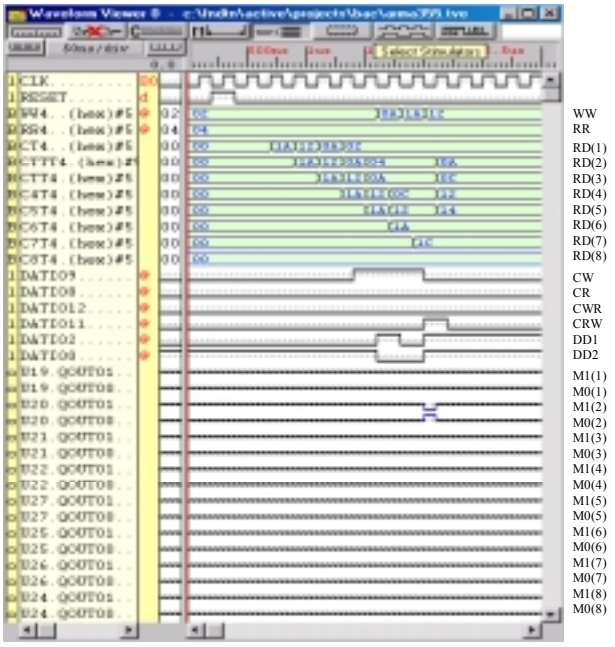


Figura 5.15. Prueba de lectura/escritura. Lectura en cola 1 con un dato y escritura en cola 3 vacía. Similar al ejemplo 3.5.5

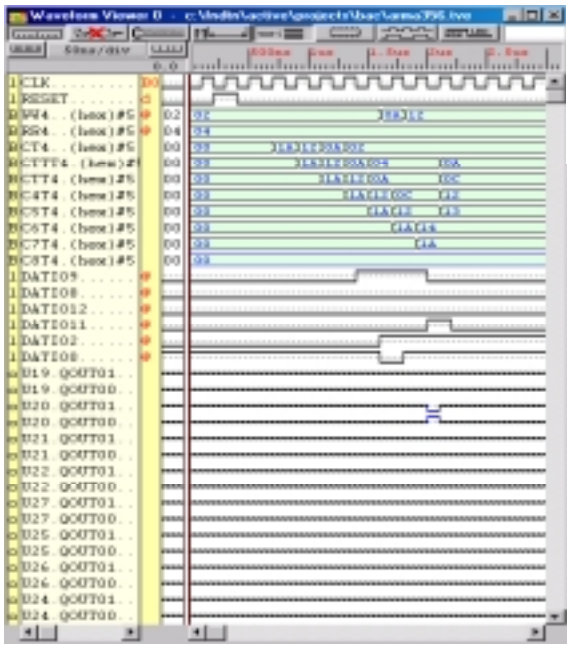


Figura 5.16. Prueba de lectura/escritura. Lectura en cola 1 con un dato y escritura en cola 3 con información. Similar al ej. 3.5.6.

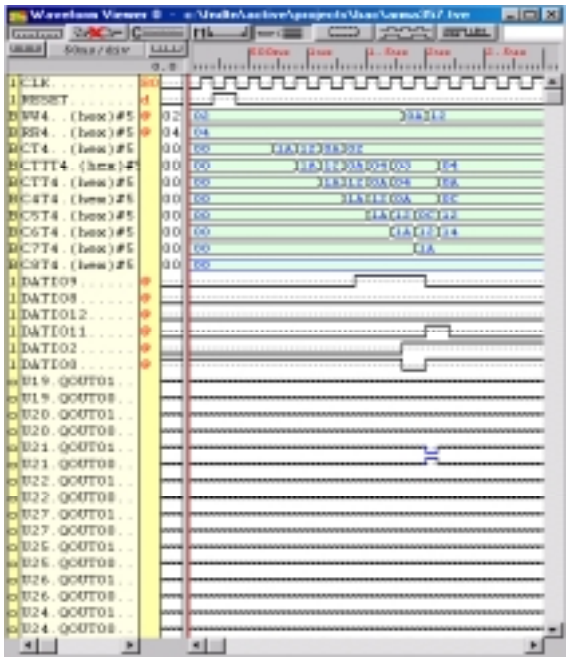


Figura 5.17. Prueba de lectura/escritura. Lectura en cola 1 con más de un dato y escritura en cola 3 vacía. Similar al ej. 3.5.7.

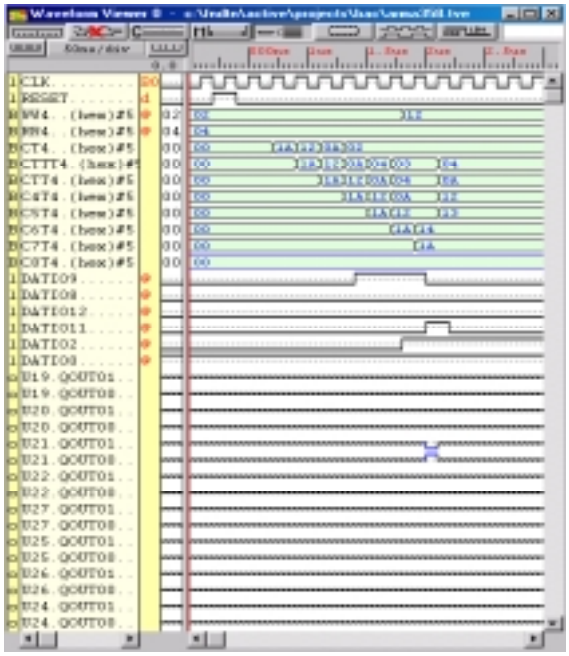


Figura 5.18. Prueba de lectura en cola 1 con más de un dato y escritura en cola 3 con información. Similar al ej. 3.5.8.

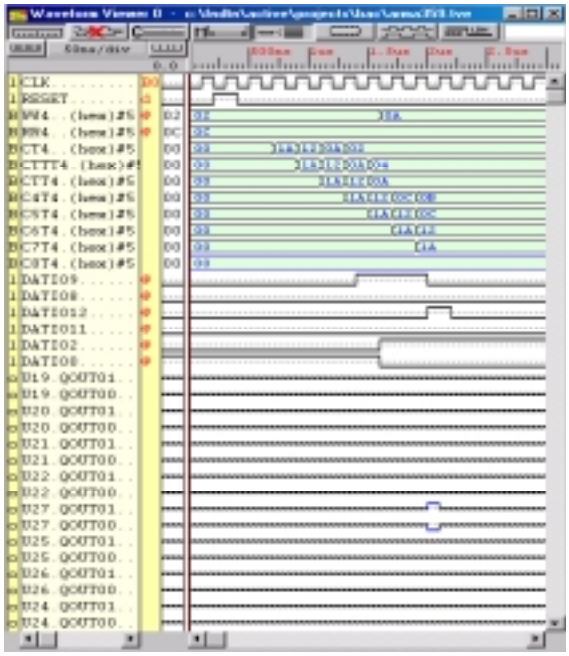


Figura 5.19. Prueba de escritura-lectura. Escritura y lectura en cola 2 con más de un dato. Similar al ejemplo 3.5.9.

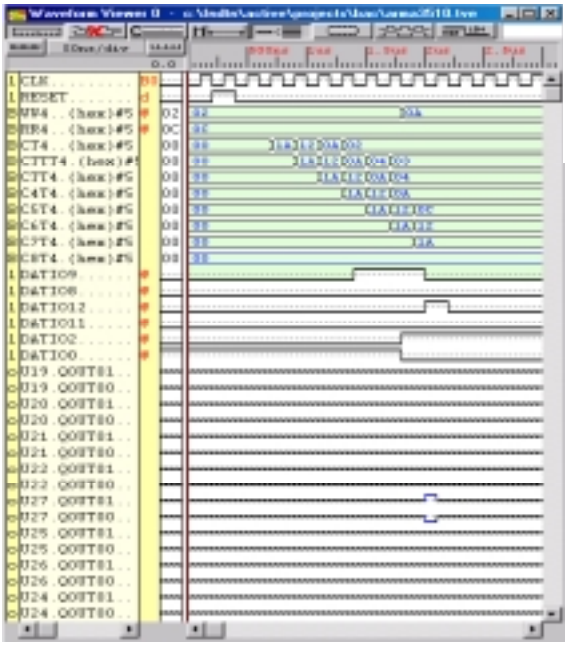


Figura 5.20. Prueba de escritura-lectura en cola 2 con un solo dato (misma localidad). Similar al ejemplo 3.5.10

el controlador de flujo del *switch* para indicarle al emisor de datos que se detenga. De esta manera se impide la pérdida de datos debida a la incapacidad del *BAC* para almacenar los datos cuando se encuentra saturado. La prueba de esta señal, mostrada en la Figura 5.21, consiste en aplicar operaciones de escritura sobre la cola 3 del *BAC*, hasta que STOPW cambie a '1'. Dicha señal aparece cuando sólo queda una localidad libre y desaparece después de una operación de lectura.

Durante las pruebas, las salidas de datos de cada localidad son independientes, pero están diseñadas para conectarse en paralelo al bus de datos de salida de nuestro *BAC*. Recuérdese que los datos almacenados en el registro *M*, constan de dos bits. En las Figuras G.7 a G.12, tales datos están representados por QOUT01 y QOUT00 para cada celda.

Nuestro *BAC* provee una señal (*STOPW*) que indica cuando el *buffer* se satura. Dicha señal puede ser utilizada por

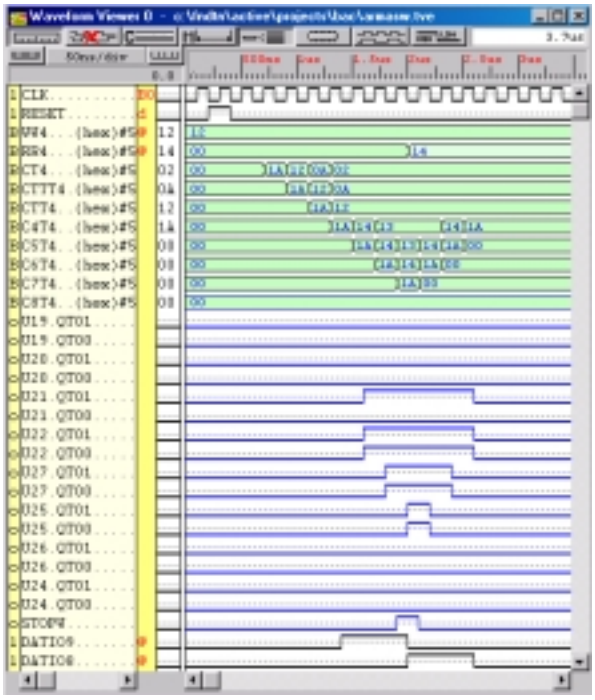


Figura 5.21. Prueba de la señal que indica que el *buffer* está lleno (STOPW).

La Figura 5.22. nos muestra, de manera esquemática, la interconexión de ocho localidades del *BAC*. La figura también contiene un bloque adicional utilizado para inicializar el registro de direcciones *RD* y el registro de almacenamiento de datos *M*. Los diagramas de tiempo, correspondientes a las pruebas mostradas en el capítulo cinco, fueron obtenidos utilizando esta configuración del *BAC*

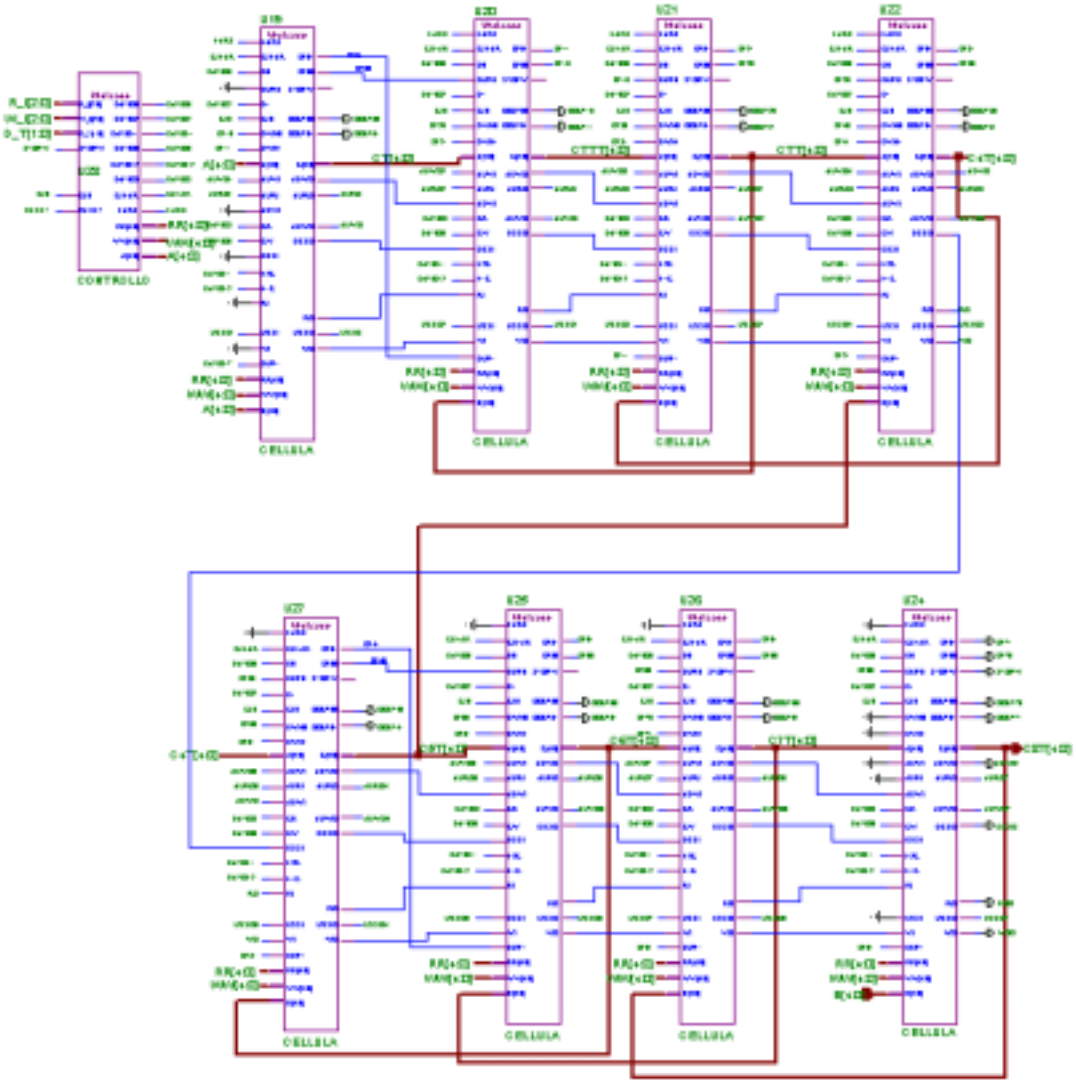


Figura 5.22. *BAC* con ocho localidades.

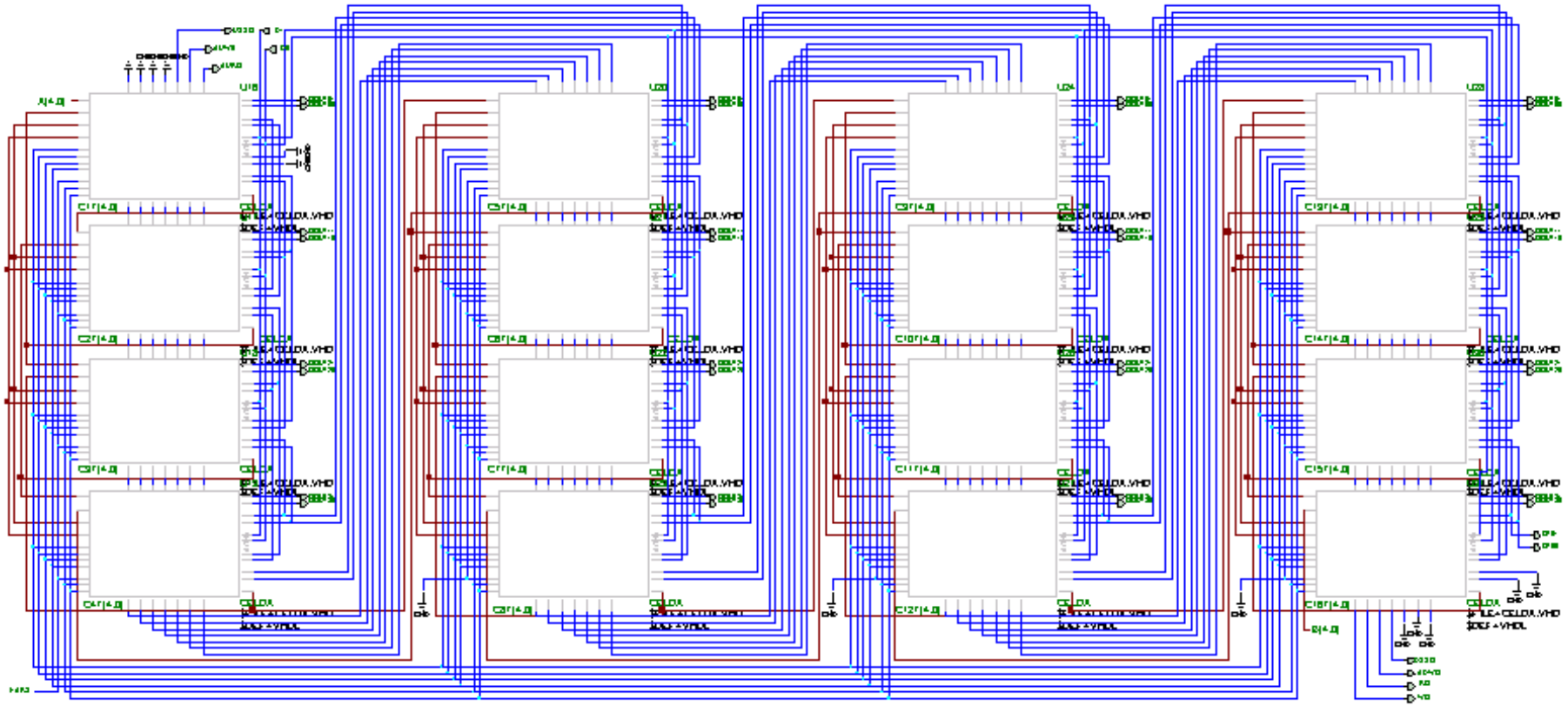


Figura 5.23 *BAC* con 16 localidades.

5.4 Conclusiones

En este capítulo hemos presentado los detalles de diseño del *buffer autocompactante*, descrito formalmente con VHDL. Mostramos las formas de onda que permitieron visualizar gráficamente el comportamiento del *BAC*. En estas gráficas pudimos constatar que nuestro modelo ha cumplido con los requerimientos que nos planteamos en el inicio. Las condiciones de prueba, mismas de los ejemplos del capítulo 3, permitieron crear los 38 casos posibles en los que pueden encontrarse las localidades del *BAC*.

Como resultado, obtuvimos la implementación del *BAC* al nivel de simulación lógica y comprobamos el funcionamiento correcto de cada una de las operaciones básicas simples y paralelas. Verificamos que las operaciones del *BAC* se realizan en un solo período de reloj, incluso las operaciones paralelas sobre una misma localidad. Inicialmente probamos una celda del módulo de control (*CCD*) y después le agregamos los módulos de almacenamiento y direccionamiento para completar una localidad del *BAC*. Finalmente, conectamos varias localidades y realizamos las pruebas necesarias. Las pruebas contemplan todas las situaciones posibles del *BAC*, por lo que con ellas se demuestra su exitoso funcionamiento.

Φ

Capítulo 6

Conclusiones y Trabajos Futuros

Este trabajo de tesis se inició con una investigación sobre aspectos de diseño de los elementos de comunicación que mejoran el rendimiento de los sistemas paralelos. Analizamos las alternativas de diseño de los *switches* y los ruteadores y centramos nuestra atención en el elemento donde se almacenan los mensajes, por ser la parte más crítica.

Nuestro trabajo consistió en diseñar un dispositivo de memoria o *buffer* para almacenar temporalmente los datos de comunicación entre los procesadores del sistema. Dicho dispositivo no puede construirse a partir de componentes comerciales y al mismo tiempo obtener un buen desempeño. Por lo que, nos propusimos diseñar una memoria especial con características de alto rendimiento.

Después de explorar algunos aspectos importantes de las máquinas paralelas, identificamos dos tipos de máquinas MIMD: la multicomputadora y el multiprocesador de memoria compartida. En el primer tipo la latencia de comunicación se incrementa, porque se invocan llamadas a funciones de paso de mensajes cada vez que un elemento de procesamiento requiere un dato [26]. Sin embargo, la frecuencia con la que se transfiere la información se reduce debido a la baja granularidad (unidad mínima de procesamiento de datos) de los programas. En el

segundo caso la granularidad es mayor (grano más pequeño), por lo que el tráfico de datos en la red de interconexión se incrementa y se hace más evidente la necesidad de un elemento de comunicación rápido.

Hemos observado que existe una clasificación de redes de interconexión que distingue cuatro categorías: las de medio compartido, las directas, las indirectas y las híbridas. En los sistemas de procesamiento paralelo que requieren alto desempeño, las redes directas e indirectas ofrecen una mayor escalabilidad (crecimiento del desempeño con el aumento de nodos en la red). Los *switches* y ruteadores son utilizados en este tipo de redes.

Los tipos de topologías de red que pueden manejar los *switches* y los ruteadores están determinados por su número de canales de entrada y salida. Nuestro dispositivo de almacenamiento es aplicable en *switches* y ruteadores con cuatro canales. Si queremos incrementar su número, sólo agregamos un *bit* en cada celda de su registro de dirección por cada vez que se duplique el número de canales.

Hemos discutido que el *retardo de enrutamiento* es el tiempo que transcurre desde que llega un mensaje al ruteador hasta que se determina el puerto de salida, sobre el cual va a ser enviado. También discutimos que la *latencia de control de flujo es* la razón a la cual pueden enviarse los mensajes a través del ruteador [80]. Dicha razón está determinada por el retardo de propagación a través del *crossbar* (contenido en el *switch* y el ruteador) y la velocidad para sincronizar la transferencia de datos a través de los *buffers*. Los retardos de enrutamiento y de control de flujo en *switches* y ruteadores determinan conjuntamente la latencia de mensaje que se alcanza a través del ellos [77], [78].

Nosotros nos hemos interesado en obtener un diseño que ayude a reducir el tiempo de control de flujo. Por tal razón hicimos un análisis de las alternativas que existen para administrar los *buffers* de manera rápida y eficiente. Observamos que es importante incrementar el tamaño de *buffer* para reducir su probabilidad de saturación. Pero, cuando la administración del *buffer* es ineficiente, el desempeño del ruteador puede ser pésimo aún con *buffers* grandes. Por lo tanto, nos avocamos a investigar cual es la mejor opción de administración y, de acuerdo con los

resultados de otros trabajos y con los que nosotros obtuvimos, nos encontramos que es la que utiliza *buffers* en las entradas y asigna los espacios dinámicamente [62] . Nuestro diseño es un *buffer* de asignación dinámica para usarse en las entradas y facilita su implementación con tamaños (número de localidades) a la medida de las necesidades.

El objetivo de esta tesis ha sido el diseño de una memoria especial para usarse como *buffer* en *switches* y ruteadores y reducir los tiempos de comunicación de datos entre los elementos de procesamiento de un sistema paralelo. Dicho *buffer*, al cual hemos denominado *Buffer AutoCompactante* o *BAC,* maneja colas asignadas dinámicamente. El diseño se realizó completamente en *hardware* para potenciar su desempeño, el cual se apoya en la autocompactación de sus espacios de almacenamiento de datos [62], [79]. Para ello, se propuso un modelo abstracto basado en cinco operaciones básicas y se caracterizaron sus principales propiedades. El modelo abstracto se concretó en un sistema basado en una serie de operaciones primitivas y se identificaron 38 situaciones posibles para cada localidad del *BAC*. Posteriormente, en una tabla se representan los 38 casos posibles en función de las operaciones primitivas. Finalmente, con la ayuda de dicha tabla, se implementa el modelo en lenguaje VHDL, y se realiza la simulación lógica.

Como resultado de esta tesis se obtuvo el diseño de un *buffer* autocompactante con espacios de almacenamiento independientes asignados dinámicamente. Nuestro diseño del *BAC*, permite realizar operaciones de escritura y lectura simultaneas, incluso en una misma localidad. Cualquier operación de escritura y/o lectura se realiza en un ciclo de reloj. Por lo que, su velocidad de operación depende de la tecnología en la que se implante. En su implementación, el tamaño en *bits* de los datos y el número de localidades del *buffer* puede ser variable, manteniendo constantes la complejidad del circuito de control y su desempeño.

Durante el desarrollo del *BAC* realizamos actividades de verificación y pruebas de funcionamiento (al nivel de simulación lógica) que permitieron corroborar las expectativas de nuestro diseño. Con la ayuda del paquete de *software* para diseño digital *Design Environment Foundation*, distribuido por *Xilinx* [83], [84], describimos en VHDL el funcionamiento de los

módulos que conforman al *BAC*. Se probó primero el módulo de control, para una localidad y, mediante diagramas de tiempo, se verificaron los 38 casos identificados. Enseguida, agregamos los módulos de almacenamiento y direccionamiento, e hicimos pruebas sobre una localidad. Después, aumentamos el número de localidades e hicimos pruebas con operaciones fundamentales simples de escritura y lectura. También, probamos el *BAC* con las operaciones paralelas de escritura/lectura y lectura/escritura. Finalmente, probamos la operación de escritura-lectura, y observamos el correcto funcionamiento con la inserción de un dato y la extracción simultánea de otro, sobre una misma localidad.

Nuestras pruebas experimentales fueron realizadas siguiendo las condiciones dadas en los ejemplos del capítulo tres. En ellas se contemplan todas las situaciones en que pueden encontrarse las colas, durante cada tipo de operación. Por consiguiente, con estas pruebas se crearon los 38 casos posibles de las localidades del *BAC*. El correcto funcionamiento del modelo, bajo las condiciones de prueba propuestas, nos demuestra el exitoso funcionamiento de las celdas del *BAC* para cada caso.

Concluimos finalmente que las contribuciones de esta tesis son las siguientes: Un análisis de las posibles configuraciones arquitecturales para un *buffer* de propósito específico, desde la perspectiva de su desempeño; Un diseño específico completo de un modelo de *buffer* (*BAC*) de baja latencia y acceso concurrente; Una implementación en VHDL del *BAC*; Un diseño y verificación funcional de los componentes que integran el *BAC*. En resumen, nuestra contribución principal se encuentra en la concepción de una memoria de propósito específico con características arquitecturales de alto desempeño.

Trabajos Futuros

Después de haber realizado este trabajo pudimos observar ciertas limitaciones que podrían ser tratadas en proyectos posteriores. Una de esas limitaciones es la no escalabilidad del *BAC*, debido a que no está diseñado para interconectarse con otros *BACs* con el fin de aumentar su capacidad. A cambio de ello, es necesario definir previamente el tamaño del *buffer* de acuerdo

con las necesidades de la aplicación. Aunque el *BAC* provee al usuario de una señal para evitar la escritura en *buffer* lleno, carece de otra señal que evite las operaciones de lectura cuando el *buffer* está vacío. Una posible solución podría obtenerse a través de la verificación del estado de las colas del *BAC*. Dado que nuestro diseño ha sido descrito en VHDL, está orientado hacia aplicaciones de bajo volumen y, por lo tanto, está limitado por el número de CLBs disponibles en los dispositivos FPGAs. Una solución a este problema se podría obtener con una implantación del tipo ASIC mediante el uso de herramientas para desarrollo de prototipos de aplicación rápida, como *Sinopsis*, las cuales aceptan como entrada el código VHDL para sintetizarlo en un circuito VLSI basado en celdas.

Pensamos que nuestro diseño del *BAC* es interesante, porque además de las características que ya tiene, se podrían agregar algunas otras para mejorar aún más el desempeño de los *switches* y ruteadores. Como resultado de discusiones en nuestro equipo de trabajo hemos contemplado, como parte de los trabajos futuros, el diseño de un *buffer* que pueda ser compartido por dos de los *n* canales de un *switch* o ruteador. Un diseño con esa capacidad permitiría utilizar más eficientemente el espacio de almacenamiento de los datos, como sucede en aquellos casos en que uno de los dos canales demanda más espacio que el otro.

Nuestro diseño contempla el uso de un *BAC* por cada canal de entrada. Así, los canales pueden estar recibiendo y enviando información simultáneamente. Uno de los requisitos de este nuevo diseño sería que el desempeño alcanzado con ese paralelismo no disminuya. Esto significa que si dos canales comparten un mismo *buffer*, la velocidad del nuevo *buffer* debería ser dos veces más rápida que la del *BAC* o debería permitir la concurrencia de dos escrituras y dos lecturas.

Bajo esta disyuntiva, consideramos más factible un diseño que aproveche la potencialidad arquitectural del *BAC* y permita la concurrencia de dos escrituras y dos lecturas. Imaginemos al nuevo *buffer* como un único espacio lineal en forma de "U" o de "herradura". Si el nuevo diseño contemplara la inicialización del *buffer* por ambos extremos en forma simultánea, tal como el *BAC* lo hace por un solo extremo, entonces tendríamos el espacio libre en la parte central de la

“herradura”. De esta manera, cada canal utilizaría el espacio libre del *buffer* de acuerdo con sus necesidades.

Quienes emprendan este nuevo diseño del *buffer* deberán analizar nuevos casos de comportamiento de sus celdas de dirección y de almacenamiento, para concebir una nueva versión de las celdas de control. El comportamiento de cada localidad del *buffer*, probablemente dependerá de sus dos vecinos superiores y sus dos inferiores. Ésta será una de las interrogantes que habrá que plantearse.

Las características del *BAC* lo hacen potencialmente útil en una amplia gama de aplicaciones que manejan arreglos de datos de tipo FIFO y que requieren memorias con tiempos de acceso muy cortos. Por ejemplo, la captura de imágenes en el área de espectrografía, la toma de imágenes en estudios de fotogrametría y el análisis óptico de partículas en movimiento, requieren de instrumentación fotográfica de alta velocidad. Muchos de los experimentos que se realizan en estas áreas utilizan cámaras fotográficas digitales de alta resolución para tomar secuencias rápidas de imágenes en movimiento con restricciones de tiempo de alta precisión. Por lo que, la captura de imágenes en estos casos debe realizarse con cámaras capaces de almacenar cantidades grandes de *bytes* en intervalos de tiempo muy cortos.

Nuestro diseño del *BAC* no tiene restricciones acerca del número de *bits* por localidad ni del número de localidades. La única limitación proviene de la capacidad del dispositivo utilizado para su implantación. Por lo que, podríamos transferir tantos *bits* como se requieran en un solo período de reloj. Esto lo hace idealmente aplicable en el manejo de imágenes, en tiempo real. Debido a restricciones biológicas de la visión humana se requiere que cada cuadro (fotografía) sea presentado durante intervalos de tiempo bien definidos (aproximadamente 30 cuadros por segundo). El procesamiento de imágenes en movimiento, con métodos tradicionales, consume mucho tiempo debido a la gran cantidad de *bits* que se requieren para almacenar un cuadro. El paralelismo del *BAC*, su tiempos de respuesta definidos por un ciclo de reloj y la posibilidad de implementarlo con longitudes de palabras grandes, son razones suficientes para que sea aplicable en éstas y otras áreas, además de la de comunicación de datos para la que fue diseñado.

Resumiendo, en esta tesis se presentó el diseño de un *buffer* autocompactante con asignación dinámica (*BAC*), aplicable en *switches* y ruteadores para sistemas de procesamiento paralelo. Con base en los resultados obtenidos, podemos afirmar que es posible construir el *BAC* y aplicarlo en una amplia gama de aplicaciones, tales como la captura y reproducción de imágenes y video, así como la comunicación de datos, entre otras.

Φ

Apéndice A

Notación Utilizada en el Modelo del *BAC*

A.1 Notación Utilizada en el Modelo

Las colas del *BAC* pueden describirse informalmente mediante arreglos de datos en memoria, manejados como una pila a través de las operaciones tradicionales de inserción y retiro de información. Una manera abstracta de describir los arreglos de datos y las respectivas operaciones asociadas con el manejo de información es mediante conceptos emanados de la teoría de conjuntos. En esta sección damos una breve explicación de la notación utilizada en la especificación del modelo abstracto del *BAC* a través de elementos de la teoría de conjuntos. El número de conceptos que presentamos es muy reducido, pero suficiente para dar una especificación abstracta del *BAC*. Una presentación completa de la aplicación de teoría de conjuntos en la especificación y construcción de programas puede ser encontrada en [72].

A.2.1 El Concepto de Función

Una forma de especificar un arreglo de información es a través del concepto matemático de *función total*. Una función total es una relación particular entre dos conjuntos denominados *rango* y *dominio* de la función. Esta relación asocia a cada elemento del dominio de la función un *único* elemento del rango. Dados dos conjuntos cualesquiera A y B, el conjunto de todas las funciones totales que pueden construirse con dominio A y rango B es denotado por:

$$A \rightarrow B$$

Para indicar que *f* es una función total con dominio *A* y rango *B*, debe estipularse que *f* pertenece al conjunto de funciones totales de dominio *A* y rango *B*, esto es:

$$f \in A \rightarrow B$$

Vista como un conjunto, una función *f* no es otra cosa que un subconjunto del producto cartesiano $A \times B$, es decir un conjuto de parejas ordenadas (a,b) para cada elemento *a* de *A* y algún elemento *b* de *B*. La manera clásica de escribir una función es a través de una ecuación de la forma $f(x) = y$, donde *x* pertenece a un cierto conjunto *A* e *y* a otro conjunto *B*, posiblemente igual a *A*. Sin embargo, vista como un conjunto, esta función se describe por $\{(x,y) \mid x \in A \wedge y \in B \wedge f(x) = y\}$.

Ejemplo A.2.1

En la sección 3.3.1, el espacio de almacenamiento del *BAC* se especifica como una función total *M*, tal que $M \in 1..m \rightarrow MEM$. El dominio de esta función es el intervalo de enteros $1..m$, donde *m* representa el número máximo de celdas en el *BAC*, y su rango es el conjunto *MEM*, que denota el conjunto de los valores que pueden almacenarse en las celdas del *BAC*. Sin pérdida de generalidad, *MEM* puede verse como el conjunto de los numeros naturales, dado que cada valor en las celdas de almancenamiento del *BAC* se representará por una secuencia de bits.

Supóngase que $m = 10$ y que *MEM* es el conjunto de valores que se pueden representar en ocho bits, de esta manera, uno de los valores posibles de *M* es:

$$M = \{(1,0),(2,1),(3,2),(4,3),(5,4),(6,5),(7,6),(8,7),(9,8),(10,9)\}$$

o cualquier otro valor de función donde a cada valor del intervalo 1..10 se le asocie un valor en el intervalo de 0..255.

Fin del ejemplo

Dado que una función es un subconjunto del producto cartesiano entre los elementos del dominio y los elementos del rago, podemos utilizar los operadores clásicos de unión ($\cup$) o intersección ($\cap$) para construir funciones.

Ejemplo A.2.2

El espacio de almacenamiento del ejemplo A.2.1 puede verse como la unión de las celdas pares y de las celdas impares:

$M = \{(1,0),(3,2),(5,4),(7,6),(9,8)\} \cup \{(2,1),(4,3),(6,5),(8,7)(10,9)\}$

Nótese que este mismo valor del espacio de almacenamiento puede denotarse de la manera siguiente:

$M = (1..10 \times 0..255) \cap \{(1,0),(2,1),(3,2),(4,3),(5,4),(6,5),(7,6),(8,7),(9,8),(10,9),(11,1),(1,256)\}$

Fin del ejemplo

Una función donde no todos los elementos de su dominio están asociados con un elemento del rango se le conoce como funcion *parcial*. Si A y B son dos conjuntos cualesquiera, el conjunto de funciones parciales de dominio A y rango B se denota por

$$A \nrightarrow B$$

Ejemplo A.2.3

Del ejemplo A.2.2 podemos concluir que las celdas impares del espacio de almancenamiento $\{(1,0),(3,2),(5,4),(7,6),(9,8)\}$ corresponden a una función parcial MI tal que $MI \in 1..10 \nrightarrow 0..255$.

Fin del ejemplo

A.2.2 Restricción de Funciones

La restricción de función es una operación que permite la construcción de una función mediante la restricción del dominio o del rango de otra función. En esta sección sólo consideramos la restricción de dominio. Dada un función f, parcial o total, cuyo dominio se encuentra en A y rango en B, así como un subconjunto S de A, la restricción del dominio de f, denotado por $S \lhd f$, denota una nueva función cuyo dominio está incluido en S y rango está incluido en B. Formalmente tenemos

$$S \lhd f = \{(a,b) \mid (a,b) \in f \wedge a \in S\}$$

Ejemplo A.2.4

Continuando con el ejemplo A.2.3, la función parcial *MI*, función que representa las celdas impares del espacio de almacenamiento, puede representarse de la manera siguiente:

$$MI = \{1,3,5,7,9\} \lhd M$$

Con esta misma notación, el ejemplo A.2.2, donde se ve al espacio de almacenamiento como la unión de celdas pares e impares, puede denotarse de la manera siguiente:

$$M = \{1,3,5,7,9\} \lhd M \cup \{2,4,6,8,10\} \lhd M$$

Fin del ejemplo

A.2.3 Composición Secuencial

Sean *A*, *B* y *C* tres conjuntos cualquiera, *f* una función con dominio en *A* y rango en *B*, y *g* otra función con dominio en *B* y rango en *C*, la composición secuencial de funciones, denotada por *f ;g*, denota una nueva función con dominio incluido en *A* y rango incluido en *C* y está formada por aquellas parejas (*a*,*c*) tales que existe un elemento *b* de *B* tal que $(a,b) \in f$ y $(b,c) \in g$, formalmente:

$$f;g = \{(a,b) \mid \exists\, c.(c \in C \wedge (a,b) \in f \wedge (b,c) \in g)\}$$

Nótese que para cualquier valor *x* del dominio de *g*, tal que *g*(*x*) esté en el dominio de *f* se tiene:

$$(f;g)(x) = f(g(x))$$

La composición secuencial es una operación sumamente importante ya que permite *especificar* cambios en todos los elementos de una función sin dar indicaciones de *cómo* se hacen esos cambios. En particular esto es importante en la especificación de las operaciones del *BAC* donde se requiere especificar un corrimiento en el contenido del *buffer*. El siguiente ejemplo muestra la manera de especificar corrimientos en el contenido de una función.

Ejemplo A.2.5

Considérese la función *M* que denota el arreglo de las celdas de almacenamiento del ejemlo A.2.1 y considérese la función *add1* definida de la manera siguiente:

$add1(x) = x+1$

para cualquier valor de x en los números naturales. La función $add1;M$ es una nueva función con dominio igual al intervalo 0..9 y rango incluido en el intervalo 0..255, donde

$(add1;M)(x) = M(add1(x)) = M(x+1)$

así, si $M = \{(1,0),(2,1),(3,2),(4,3),(5,4),(6,5),(7,6),(8,7),(9,8),(10,9)\}$, entonces la función $add1;M$ es:

$add1;M = \{(0,0),(1,1),(2,2),(3,3),(4,4),(5,5),(6,6),(7,7),(8,8),(9,9)\}$

Aquí podemos ver claramente, considerando a M como un arreglo con información, que la función $1..9 \lhd (add1;M)$ especifica las primeras nueve celdas de ese arreglo, con la información recorrida una posición hacia la *izquierda*. De manera similar, considerando la función *sub1* definida por

$sub1(x) = x-1$

para x definida en los números naturales positivos, la expresión $2..10 \lhd (sub1;M)$ especifica las últimas nueve celdas de ese arreglo, con la información recorrida un lugar hacia la derecha.

Fin del ejemplo

Ф

Apéndice B

```
------------------------------------------------------------------
--   Celda CD(i): Una celda del modulo de Control Distribuido   --
------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity Celda_CD_i is
port(SS,CAR,CAW,CRW,CWR,CR,CW: in  std_logic;
     SSCA,SSCS,CAWA,CSRA,CSWA:     in  std_logic;
     RD,RR,WW:                in  std_logic_vector(7 downto 0);
     EVD,ENVD,RO,WO,HM,HRD:        out std_logic;
     S1M,S0M,CARR,CADD,S0RD:       out std_logic;
     ISWA,IARA,IAWA,HLM:           out std_logic;
     ISSS,IASS:                    out std_logic
     );
end Celda_CD_i;

architecture funcion of Celda_CD_i is
   signal vsal: std_logic_vector(14 downto 0);
   signal CPW: std_logic;

begin
process(vsal,CPW,CAR,CAW,SSCA,CSRA,CAWA,SS,RR,RD,WW,CR,CW,CW,CRW,C
WR,SSCS,CSWA)begin
CPW<=RD(1) and not RD(0);
if (CW='1' AND RD=WW AND SS='0')
then vsal <="011000000000100"; ---------------------------- caso 1
end if;
if (CW='1' AND RD=WW AND SS='1')
then vsal <="000000000000101"; ---------------------------- caso 2
end if;
if (CW='1' AND RD>WW AND CAWA='1' AND SSCA='0')
then vsal <="110000110000000"; -----------------------------caso 3
end if;
if (CW='1' AND RD>WW AND CAWA='1' AND SSCA='1')
then vsal <="110001010000000"; ---------------------------- caso 4
end if;
if (CW='1' AND RD/=X"00" AND RD>WW AND CAWA='0')
then vsal <="110000000000000"; ---------------------------- caso 5
end if;
if (CW='1' AND RD=X"00" AND CAW='1')
```

```
then vsal <="110000000000000"; ---------------------------- caso 6
end if;
if (CW='1' AND RD=X"00" AND CAR='1')
then vsal <="110000000000000"; ---------------------------- caso 7
end if;
if (CW='1' AND RD=X"00" AND CAWA='1')
then vsal <="110000110000000"; ---------------------------- caso 8
end if;
if (CW='1' AND RD=X"00" AND CAW='0' AND CAR='0')
then vsal <="000000000000000"; ---------------------------- caso 9
end if;
if (CW='1' AND RD/=X"00" AND RD<WW  )
then vsal <="000000000000000"; ---------------------------- caso 10
end if;
if (CR='1' AND RD=RR)
then vsal <="110010011101000"; ---------------------------- caso 11
end if;
if (CR='1' AND RD/=X"00" AND RD<RR AND CSRA='0')
then vsal <="000000000000000"; ---------------------------- caso 12
end if;
if (CR='1' AND RD/=X"00" AND RD<RR AND CPW='1' AND CSRA='1')
then vsal <="000100000000000"; ---------------------------- caso 13
end if;
if (CR='1' AND RD/=X"00" AND RD<RR AND CPW='0' AND CSRA='1')
then vsal <="000010000000000"; ---------------------------- caso 14
end if;
if (CR='1' AND RD>RR)
 then vsal <="110010011000000"; ---------------------------- caso 15
end if;
if (CR='1' AND RD=X"00")
then vsal <="000000000000000"; ---------------------------- caso 16
end if;
if (CWR='1' AND RD/=X"00" AND RD<WW)
then vsal <="000000000000000"; ---------------------------- caso 17
end if;
if (CWR='1' AND RD=WW AND SS='0')
then vsal <="011000000000100"; ---------------------------- caso 18
end if;
if (CWR='1' AND RD=WW AND SS='1')
then vsal <="000000000000101"; ---------------------------- caso 19
end if;
if (CWR='1' AND RD>WW AND RD<RR AND CAWA='1' AND SSCA='0')
then vsal <="110000110000000"; ---------------------------- caso 20
end if;
if (CWR='1' AND RD>WW AND RD<RR AND SSCA='1' AND CAWA='1')
then vsal <="110001010000000"; ---------------------------- caso 21
end if;
```

```
if (CWR='1' AND RD>WW AND RD<RR AND CAWA='0')
then vsal <="1100000000000000"; ---------------------------- caso 22
end if;
if (CWR='1' AND RD=RR AND CAW='1')
then vsal <="1101000001000000"; ---------------------------- caso 23
end if;
if (CWR='1' AND RD=RR AND CAW='0')
then vsal <="0100000001000000"; ---------------------------- caso 24
end if;
if (CWR='1' AND RD>RR)
 then vsal <="0000000000000000"; ---------------------------- caso 25
end if;
if (CWR='1' AND RD=X"00")
then vsal <="0000000000000000"; ---------------------------- caso 26
end if;
if (CRW='1' AND RD/=X"00" AND RD<RR AND CSRA='0')
then vsal <="0000000000000000"; ---------------------------- caso 27
end if;
if (CRW='1' AND RD/=X"00" AND RD<RR AND CPW='1' AND CSRA='1')
then vsal <="0001000000000000"; ---------------------------- caso 28
end if;
if (CRW='1' AND RD/=X"00" AND RD<RR AND CPW='0' AND CSRA='1')
then vsal <="0000100000000000"; ---------------------------- caso 29
end if;
if (CRW='1' AND RD=RR)
then vsal <="1100100111010000"; ---------------------------- caso 30
end if;
if (CRW='1' AND RD>RR AND RD<WW AND CSWA='0')
then vsal <="1100100110000000"; ---------------------------- caso 31
end if;
if (CRW='1' AND RD>RR AND RD<WW AND CSWA='1' AND SSCS='0')
then vsal <="1110100000000000"; ---------------------------- caso 32
end if;
if (CRW='1' AND RD>RR AND RD<WW AND CSWA='1' AND SSCS='1')
then vsal <="1100100110000000"; ---------------------------- caso 33
end if;
if (CRW='1' AND RD=WW AND SS='0')
then vsal <="1100001100100000"; ---------------------------- caso 34
end if;
if (CRW='1' AND RD=WW AND SS='1')
then vsal <="1100010100100100"; ---------------------------- caso 35
end if;
if (CRW='1' AND RD>WW)
then vsal <="0000000000000000"; ---------------------------- caso 36
end if;
if (CRW='1' AND RD=X"00")
then vsal <="0000000000000000"; ---------------------------- caso 37
```

```
end if;
if (CWR='1' AND RD=RR AND CAWA='1')
then vsal <="0100000010010000"; --------------------------- caso 38
end if;

ISSS<=vsal(0);
IASS<=vsal(1);
ISWA<=vsal(2);
IARA<=vsal(3);
IAWA<=vsal(4);
HLM <=vsal(5);
S1M <=vsal(6);
S0M <=vsal(7);
CARR<=vsal(8);
CADD<=vsal(9);
S0RD<=vsal(10);
EVD <=vsal(11);
ENVD<=vsal(12);
HM  <=vsal(13);
HRD <=vsal(14);
WO<=RD(1) and not RD(0);
RO<=RD(2);

end process;
end funcion;
```

Apéndice C

Código VHDL de una Localidad del BAC

```
------------------------------------------------------------------------------------------------------------------------
--                                    CODIGO DE UNA LOCALIDAD DEL BAC                                                 --
------------------------------------------------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity LOCALIDAD is
        port (CLK, CR, CW, CWR, CRW, CAW, CAR, CSWA, CSRA, CAWA,
              SSCS, SSCA, DWN0, DUP0, d0, d1, DWN1,
              DUP1, CLEAR, EARX : in STD_LOGIC;
              IARA, ISWA, IAWA, IASS, ISSS,  WO,RO, qout01,
              qout00, STOPW: OUT STD_LOGIC;
              Qt00, Qt01: buffer STD_LOGIC;
              A,B, WW, RR: in STD_LOGIC_VECTOR (4 downto 0);
              C: BUFFER STD_LOGIC_VECTOR(4 downto 0));
end LOCALIDAD;

architecture  DIR_CONT_MEM of LOCALIDAD is
signal data0, data1, ENVD, EVD,HM, HRD,HLM,TMP0,TMP1, Q0, S0M,S1M,S0RD, CADD, CARR,
               SSS: STD_LOGIC;
signal RD: STD_LOGIC_VECTOR(4 downto 0);

function mux_sch (AA,BB,C: STD_LOGIC_VECTOR(4 downto 0);
              X0,X1,X2: STD_LOGIC) return STD_LOGIC_VECTOR is
              variable SEL: STD_LOGIC_VECTOR (2 downto 0);
              variable CC: STD_LOGIC_VECTOR (4 downto 0);
   begin
                SEL(0) := X0;
                SEL(1) := X1;
                SEL(2) := X2;
                case SEL is
                when "000" => CC := AA;
                when "001" => CC := BB;
                when "010" => CC(0):='1';CC(1):='1';CC(2):='0';CC(3):= AA(3); CC(4):= AA(4);
                when "100" => CC(0):='0';CC(1):='0';CC(2):='1';CC(3):= AA(3); CC(4):= AA(4);
                when "101" => CC(0):='1';CC(1):='1';CC(2):='0';CC(3):= C(3) ; CC(4):= C(4) ;
                when "011" => CC(0):='0';CC(1):='0';CC(2):='1';CC(3):= C(3); CC(4):= C(4) ;
                when others => CC:= AA;
                end case;
               return CC;
   end ;

function fx3c(CR, CRW, CSWA, SSCS: STD_LOGIC;
          RD, WW, RR: STD_LOGIC_VECTOR) return STD_LOGIC is
   begin
     if ((CR='1' AND (RD=RR))
     OR ( CR='1' AND (RD>RR))
     OR (CRW='1' AND (RD=RR))
```

```
    OR (CRW='1' AND (RD>RR) AND (RD<WW) AND ( CSWA='0'))
    OR (CRW='1' AND (RD>RR) AND (RD<WW) AND CSWA='1' AND SSCS='1')) then
    return'1';else return '0';
    end if;
   end;

function fx4c (CW, CR, CWR, CRW, CAWA, CSWA, SSCS, SSCA: STD_LOGIC;
            RD, WW, RR: STD_LOGIC_VECTOR)return STD_LOGIC is
    begin
    if ((CW='1' AND (RD>WW) AND CAWA='1')
    OR (CW='1' AND (RD="00000") AND CAWA='1' AND ( SSCA='0'))
    OR (CR='1' AND (RD=RR))
    OR (CR ='1'AND (RD>RR))
    OR (CWR='1' AND (RD>WW) AND (RD<RR) AND (CAWA = '1') )
    OR (CWR= '1' AND (RD=RR) AND (CAWA ='1'))
    OR (CRW='1' AND (RD=RR))
    OR (CRW='1' AND (RD>RR) AND (RD<WW) AND ( CSWA='0'))
    OR (CRW='1' AND (RD>RR) AND (RD<WW) AND  CSWA='1' AND SSCS='1')
    OR (CRW='1' AND (RD=WW))) then return'1'; else return '0';
    end if;
    end;

 function fmux( d0, d1, d2, d3, s0,s1 :STD_LOGIC) return STD_LOGIC is
 variable salida: STD_LOGIC;
 variable sel: STD_LOGIC_VECTOR(1 downto 0);
 begin
 sel(0) := s0;
 sel(1) := s1;
  case sel is
      when "00" => salida := d0;
      when "01" => salida := d1;
      when "10" => salida := d2;
      when "11" => salida := d3;
      when others => salida := '0';
  end case;
  return salida;
 end;

 begin

 P_HM:  process (CR, CW, CWR, CRW, RD, RR, WW,  SSS, CAWA,  CAW, CSRA, SSCS,
SSCA,CAR,CSWA )

    begin
    if ((CW='1' AND (RD=WW) AND ( SSS='0'))
    OR (CW='1' AND (RD="00000") AND CAWA='1' AND ( SSCA='0'))
    OR (CW='1' AND (RD>WW) AND CAWA='1')
    OR (CW='1' AND (NOT (RD="00000")) AND (RD > WW) AND ( CAWA='0'))
    OR (CW='1' AND (RD="00000") AND CAW='1')
    OR (CW='1'  AND (RD="00000") AND CAR='1')
    OR (CR='1' AND (RD=RR))
    OR (CR='1' AND (NOT(RD= "00000")) AND (RD < RR) AND (RD(1)/= RD(0)) AND CSRA='1')
    OR (CR='1' AND (RD>RR))
    OR (CWR='1' AND (RD=WW) AND (SSS='0'))
    OR (CWR='1' AND (RD=RR) AND (CAWA ='1'))
    OR (CWR='1' AND (RD>WW) AND (RD<RR))
```

```
    OR (CWR='1' AND (RD=RR))
    OR (CRW='1' AND (NOT (RD= "00000" )) AND (RD < WW) AND (RD < RR) AND (RD(1)/=
        RD(0)) AND CSRA='1')
    OR (CRW='1' AND (RD=RR))
    OR (CRW='1' AND (RD>RR) AND (RD<WW) AND (NOT CSWA='1'))
    OR (CRW='1' AND (RD > RR) AND (RD< WW) AND CSWA='1' AND (SSCS='0'))
    OR (CRW='1' AND (RD=WW))
    OR (CRW='1' AND (RD>RR) AND (RD<WW) AND (CSWA='1'))) then HM<='1'; else HM<= '0';
    end if;
    end process P_HM;

P_HRD: process (CW, CRW, CWR, CR, RD, RR, WW, CAW, CAR, CAWA, SSCA, CSRA,SSS)

     begin
     if ((CW='1' AND (RD>WW) AND (CAWA='1') AND (SSCA='0'))                    --------- 3
     OR (CW='1' AND (RD>WW) AND (CAWA='1') AND (SSCA='1'))                     --------- 4
     OR (CW='1' AND (NOT (RD="00000")) AND (RD>WW) AND (CAWA='0'))             --------- 5
     OR (CW='1' AND (RD="00000") AND CAW='1')                                  --------- 6
     OR (CW='1' AND (RD="00000") AND CAR='1')                                  --------- 7
     OR (CW='1' AND (RD="00000") AND (CAWA='1') AND ( SSCA='0'))               --------- 8
     OR (CR='1' AND (RD=RR))                                                   --------- 1
     OR (CR='1' AND (RD>RR))                                                   --------15
     OR (CWR='1' AND (RD=WW) AND (RD<RR) AND (CAWA='1') AND(SSCA='0'))         --------20
     OR (CWR='1' AND (RD=WW) AND (RD<RR) AND (CAWA='1') AND(SSCA='1'))         --------21
     OR (CWR='1' AND (RD=WW) AND (RD<RR) AND (CAWA='0'))                       --------22
     OR (CWR='1' AND (RD=RR) AND (CAW='1') AND (CAWA='1'))                     --------23
     OR (CRW='1' AND (NOT(RD="00000"))AND(RD<RR)AND(NOT(RD(1)='1'AND
          RD(0)='0'))AND(CSRA='1'))                                            ------- 29
     OR (CRW='1' AND (RD=RR))                                                  --------30
     OR (CRW='1' AND (RD>RR) AND (RD<WW)AND (CSWA= '0'))                       ------- 31
     OR (CRW='1' AND (RD>RR) AND (RD<WW)AND (CSWA= '1') AND (SSCS='0'))        ------- 32
     OR (CRW='1' AND (RD>RR) AND (RD<WW)AND (CSWA= '1') AND (SSCS='1'))        ------- 33
     OR (CRW='1' AND (RD=WW) AND(SSS='0'))                                     --------34
     OR (CRW='1' AND (RD=WW) AND(SSS='1')))                                    --------35
     then HRD <= '1'; else HRD<= '0';
     end if;
     end process P_HRD;

 P_ENVD: process (CW, RD, WW,SSS, CWR, CSWA, RR, SSCS, CRW)

    begin
    if ((CW='1' AND (RD=WW) AND ( SSS='0'))
    OR (CWR='1' AND (RD=WW) AND ( SSS='0'))
    OR (CRW='1' AND (RD>RR) AND (RD<WW) AND (CSWA='1') AND (SSCS='0'))) then
       ENVD<='1'; else ENVD <='0';
    end if;
    end process P_ENVD;

P_EVD: process (CR, RD, RR, WW, CSRA, CWR, CAW, CRW, CSWA, SSCS)

    begin
    if ((CR='1' AND (NOT (RD="00000")) AND (RD<RR) AND (RD(1)/= RD(0)) AND CSRA='1')
    OR (CWR='1' AND (RD=RR) AND CAW='1')
    OR (CRW='1' AND (NOT (RD="00000")) AND (RD<WW) AND (RD<RR) AND  (RD(1)/= RD(0))
         AND CSRA='1') )
    then EVD <='1'; else EVD <= '0';
```

```
    end if;
    end process P_EVD;

P_HLM: process (CLK, CR, CWR, CRW, RD, RR,CAWA )

    begin
    if ((CR='1' AND (CLK ='0') AND (RD=RR))
    OR (CWR='1' AND (CLK ='0') AND (RD=RR))
    OR (CWR='1' AND (CLK='0') AND(RD=RR) AND (CAWA ='1'))
    OR (CRW='1' AND (CLK ='0')AND (RD=RR)))
    then HLM <= '1'; else HLM <= '0';
    end if;
    end process P_HLM;

P_S0RD: process (CR, CRW, RD, RR, WW, CSRA, CSWA)

    begin
    if ((CR='1' AND (NOT (RD="00000")) AND (RD<RR) AND  NOT(RD(1)= '1' and RD(0)='0') AND
        CSRA='1')
    OR (CR='1' AND (RD>RR))
    OR (CR ='1' AND RD = RR)
    OR (CRW='1' AND (NOT (RD="00000")) AND (RD<WW) AND (RD<RR) AND  NOT(RD(1) = '1'
         and RD(0) ='0') AND CSRA='1')
    OR (CRW='1' AND (RD=RR))
    OR (CRW='1' AND (RD>RR) AND (RD<WW))
    OR (CRW='1' AND (RD= WW))) then S0RD <= '1'; else S0RD <= '0';
    end if;
    end process P_S0RD;

P_CADD: process ( CW, CWR, CRW, RD, RR, WW, CAWA, SSS, SSCA, CSWA)

    begin
    if ((CW='1' AND (RD>WW) AND CAWA='1' AND SSCA='1')
    OR (CWR='1' AND (RD>WW) AND (RD<RR) AND ( CAWA='1') AND (SSCA ='1'))
    OR (CRW='1' AND (RD=WW) AND SSS='0')
    OR (CRW ='1' AND RD>RR AND RD<WW AND CSWA='0'))
    then CADD <= '1'; else CADD <= '0';
    end if;
    end process P_CADD;

P_CARR: process (CW, CWR, CRW, CAWA, SSCA, SSS,RD, WW, RR)

    begin
    if ((CW='1' AND (RD>WW) AND CAWA='1' AND ( SSCA='0'))
    OR (CW='1' AND (RD="00000") AND CAWA='1' AND (SSCA='0'))
    OR (CWR='1' AND (RD>WW) AND (RD<RR) AND CAWA='1' AND SSCA='0')
    OR (CRW='1' AND (RD=WW) AND (SSS='1'))) then CARR <='1'; else CARR <='0';
    end if;
    end process P_CARR;

S0M <= fx3c(CR,CRW,CSWA,SSCS, RD,WW, RR);

S1M <=fx4c(CW, CR, CWR, CRW, CAWA, CSWA,SSCS,SSCA,RD,WW,RR);

P_IAWA: process ( CRW, RD, WW)
```

```
      begin
      if (CRW='1' AND (RD=WW)) then
       IAWA <= '1'; else IAWA <='0';
      end if;
      end process P_IAWA;

P_IARA:  process ( CR, CRW, RD, RR)

      begin
      if ((CR='1' AND (RD=RR))
      OR (CRW='1' AND (RD=RR))) then
       IARA <='1'; else IARA <= '0';
      end if;
      end process P_IARA;

P_ISWA: process ( CW, CWR, RD, WW)
       begin

      if ((CW='1' AND (RD=WW))
      OR (CWR='1' AND (RD=WW) ) ) then
       ISWA <= '1' ; else ISWA <='0';
      end if;
      end process P_ISWA;

P_IASS: process ( CRW, RD, WW, SSS)

      begin
      if (CRW='1' AND (RD=WW) AND SSS='1') OR (CW='1' AND (RD=WW) AND SSS='1') then
       IASS <= '1'; else IASS <= '0';
      end if;
      end process P_IASS;

P_ISSS: process ( CW, RD, WW, SSS,CWR)

       begin
       if ((CW='1' AND (RD=WW) AND  (RD(1) /= RD(0)) AND SSS='1')
       OR (CWR='1' AND (RD=WW) AND  (RD(1) /= RD(0)) AND SSS='1')) then
        ISSS<= '1'; else ISSS <= '0';
       end if;
       end process P_ISSS;

P_WO: process ( RD)

      begin
      if (RD(1)='1' AND RD(0)='0')then
       WO <= '1'; else WO <= '0';
      end if;
      end process P_WO;

P_RO: process (RD)

      begin
      if (RD(2) = '1' AND RD(1) = '0' and RD(0)= '0')then
       RO <= '1'; else RO <= '0';
      end if;
      end process P_RO;
```

```
data0 <= fmux(dup0, d0, d0, dwn0 , x4c, x3c);
data1 <= fmux(dup1, d1, d1,  dwn1,  x4c, x3c);

 M_EM0:  process (CLK)

        begin
        if (CLK'event and CLK ='1') then
        if ((EVD ='1') or (CLEAR = '1')) then
         Q0 <= '0';elsif
         (HM = '1') then
         Q0 <= Data0;
        end if;
        end if;
        end process;

       process (CLK)
       begin
       if (CLK'event and CLK ='0') then
            Qt00 <= Q0;
       end if;
       end process;

 S_ALIDA0  : process (HLM, Qt00)

       begin
         if HLM = '1' then
           Qout00 <= Qt00; else Qout00 <= 'Z';
         end if;
       end process;

 M_EM1:  process (CLK)

       begin
        if (CLK'event and CLK ='1') then
          if (ENVD ='1') then
            SSS <= '1'; elsif
            ((EVD ='1') or (CLEAR = '1'))  then
            SSS <= '0';elsif
            (HM = '1') then
            SSS <= Data1;
           end if;
        end if;
       end process;

       process (CLK)
       begin
       if (CLK'event and CLK ='0') then
           Qt01 <=SSS;
       end if;
       end process;

 S_ALIDA1: process (HLM, Qt01)
          begin
          if HLM = '1' then
          Qout01 <= Qt01; else Qout01 <= 'Z';
```

```
        end if; end process;

C_C: process (CLK)
        begin
        if (CLK'event and CLK ='1') then
       if ((HRD ='1') or (EARX ='1')) then
        C <= mux_sch (A,B,C, S0RD,CADD,CARR);
        RD <= mux_sch(A,B,C, S0RD,CADD,CARR); end if;
       if (CLEAR ='1') then
       C <= "00000";
       RD <= "00000";
        end if;
        end if;
        end  process;

S_STOPW: process (CLK,A)
        begin
       if (CLK ='1' ) then
       if (A /= "00000")
       then
       STOPW <='1'; else STOPW <='0';
       end if;
       end if;
       end process;


end DIR_CONT_MEM;
```

Ф

Bibliografía

[1] Rexford Jennifer, John Hall, Kang G. Shin, "A Router Architecture for Real-Time Communication in Multicomputer Networks", IEEE Transactions on Computers, vol. 47, pp. 1088-1101, 1998.

[2] Almasi G. S., and A. Gottlieb, *Highly Parallel Computing*, Redwood City, CA: Benjamin/Cummings, 2nd ed., 1994.

[3] Hwang Kai, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, 1993.

[4] Flynn M., *Computer architecture: Pipelined and Parallel Processor Design,* Jones and Bartlet Boston, MA., 1995.

[5] Ni L. M., "Issues in designing truly scalable interconnection networks", Proceedings of the 1996 ICPP Workshop on Challenges for Parallel Processing, pp. 74-83, August 1996.

[6] Hsin-Chou Chi and Yuval Tamir, "Starvation Prevention for Arbiters of Crossbars with Multi-Queue Input Buffers", IEEE International Computer Conference - COMPCON, San Francisco, CA, pp. 292-297, February 1994.

[7] Hillis W. D. and L. W. Tucker, "The CM-5 Connection Machine: A Scalable Supercomputer", Communications of the ACM 36 (11), pp. 30-40, November 1993.

[8] Varma Anujan, *Interconnection networks for multiprocessors and multicomputers: theory and practice*, Manuscript for IEEE tutorial text / Anujan Varma, C.S. Raghavendra, pp. 392-404, 1994.

[9] Chien A. A., "A cost and speed model for k-ary n-cube wormhole routers", Proceedings of Hot Interconnects´93, August 1993.

[10] Dally W. J. and C. L. Seitz, "The Torus Routing Chip", Journal of Distributed Computing, vol 1, no. 3, pp. 187-196, 1986.

[11] Rimoni Y., I. Zisman, R. Ginosar, and U. Weiser, "Communication Element for the Versatile MultiComputer", 15th IEEE Conference on Israel, April 1987.

[12] Duato J., A. Robles, F. Silla, R. Beivide, "A Comparison of Router Architectures for Virtual Cut-Through and Wormhole Switching in a NOW Environment", IEEE Journal of Parallel and Distributed Computing, 1999.

[13] Tamir Y., G. L. Frazier, "High-Performance Multi-Queue Buffers for VLSI Communication Switches", Proc. 15th Ann. Symp. Computer Architecture, pp. 343-354, 1988.

[14] Mneimneh S., V. Sharma, Kai-Yeng S., "Switching Using Parallel Input-Output Queued Switches With No Speedup", IEEE/ACM Transactions on Networking, vol. 10, No. 5, October 2002.

[15] Culler D. E., J. P. Singh, and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann Publishers, Inc., 1999.

[16] Duato J., S. Yalamanchili, and Lionel Ni, *Interconnection Networks: an Engineering Approach*, IEEE Computer Society Press, 1997.

[17] Kermani P., L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique", Computer Networks, vol. 3, pp. 267-286, 1979.

[18] Mohapatra Prasant, "Wormhole Routing Techniques for Directly Connected Multicomputer Systems", Department of Electrical and Computer Engineering, Iowa State University.

[19] Martínez J. M., P. López, J. Duato, "Impact of Buffer Size on the Efficiency of Deadlock Detection", IEEE Computer Society, DISCA, UPV, Spain. 1999.

[20] Crowter W., J. Goodhue, R. Gurwitz, R. Rettberg, and R. Thomas, "The Butterfly Parallel Processor", IEEE Computer Architecture Newsletter, pp. 18-45, December 1985.

[21] Seitz C. L., "The Cosmic Cube", Communications of the ACM, vol. 28, no. 1, pp. 22-23, January 1985, Massachusetts, U. S.

[22] Whitby-Strevens C., "The Transputer", Proceedings of the 12th Annual Symposium on Computer Architecture, pp. 292-300, June 1985.

[23] Stevens K. S., S. V. Robinson, A. L. Davis, “The Post Office Communication Support for Distributed Ensemble Architectures”, The 6th International Conference on Distributed Computing Systems, Cambridge, MA, pp. 160-166, May 1986.

[24] Delgado-Frías J. G., Richard Díaz, “A VLSI Self-Compacting Buffer for DAMQ Communication Switches”, IEEE 8th Great Lakes Symposium on VLSI, Lafayette, Louisiana, p.p. 128-133, February 1998.

[25] Chien A. A., “A Cost and Speed Model for k-ary n-cube Wormhole Routers”, Proceedings of Hot Interconnects ´93, August 1993.

[26] Jiménez F. A., A. De Luca P., “Diseño de un Ruteador Flexible y Adaptativo para Sistemas Paralelos”, Congreso Internacional de Computación, Centro de Investigación en Computación, IPN, México, Noviembre de 1999.

[27] Fujimoto R. M., “VLSI Communication Components for Multicomputer Networks”, CS Division Report No. UCB/CSD 83/136, University of California, Berkeley, CA, 1983.

[28] Reed D. A. and R. M. Fujimoto, *Multicomputer Networks: Message-Based Parallel Processing*, The MIT Press, 1987.

[29] Karol M. J., M. G. Hluchyj, and S. P. Morgan, “Input vs Output Queueing on a Space-Division Packet Switch”, IEEE Global Telecommunications Conference, Houston, TX, pp. 659-665, December 1986.

[30] Seitz C. L., W. K. Su, “A Family of Routing and Communication Chips Based on Mosaic”, Proc. of Univ. of Washington Symposium on Integrated Systems, Cambridge, MA: MIT Press, pp. 320-337, 1993.

[31] Karol M. J., M. G. Hluchyj, and S. P. Morgan, “Input vs Output Queueing on a Space-Division Packet Switch”, IEEE Transactions on Communications 35(12): 1347-1356. 1987.

[32] Li S. Y., “Theory of Periodic Contention and Its Application to Packet Switching”, *Proc. INFOCOM ’88*, pp. 320-325, March 1988.

[33] Makedon F., A. Symvonis. “Optimal Algorithms for the Many-to-one Routing Problem on Two-dimensional Meshes”, Butterworth-Heinemann Ltd, Vol. 17, No. 16, 1993.

[34] Park J., S. Vassiliadis, J. G. Delgado-Frías, “Router Architecture for Oblivious Routing Algorithms”, Parallel Computing Technologies (PaCT-93), Obninsk, Rusia, 1993.

[35] Blake Ross G., Davis S. Mazel, “A Scalable Multicomputer”, IBM Federal Systems Company, IEEE, 1993.

[36] Delgado-Frías J. G., Rovy Sze, D. H. Summerville, V. Aikens, “A VLSI CAM-Based Flexible Oblivious Router for Multiprocessor Interconnection Networks”, Great Lakes Symposium on VLSI '94, 1994

[37] Sriskanthan N., A. Das, P. Loh, A. H. Leong. “An Adaptive Switching Architecture for Multiprocessor Networks”, Microprocessors and Microsystems, Vol. 18, No. 6, 1994.

[38] AlKasabi S., S. Hariri, “A Dynamically Reconfigurable Switch for High-Speed Networks”, IEEE Proceedings of the Fourteenth Annual International Phoenix Conference on Computer and Communications, pp. 508-514, Mar 1995.

[39] Stenstrom P., M. Balldin, J. Skeppstedt, “The Design of a Non-Blocking Load Processor Architecture”, Microprocessors and Microsystems 20, pp. 111-123, 1996.

[40] Steenkiste P., “Network-Based Multicomputers: A Practical Supercomputer Architecure”. IEEE Transactions on Parallel and Distributed Systems, vol. 7, no. 8, 1996.

[41] Delgado-Frías J. G., J. Nyathi, Ch. L. Miller, D. H. Summerville, “A VLSI Interconnection Networks Router Using a D-CAM with Hidden Refresh”, 6th Great Lakes Symposium on VLSI, Binghamton, NY, 1996.

[42] Corradi A., C. Stefanelli, “A deadlock Prevention Strategy for Adaptive Routing Systems”, Microprocessors and Microsystems, vol. 20, pp. 97-103, 1996.

[43] Duato J., P. Lopez, F. Silla, S. Yalamanchili, “A High Performance Router Architecture for Interconnection Networks”, IEEE International Conference on Parallel Processing, pp. 161-168, 1996.

[44] Rosu M. C., K. Schwan, R. Fujimoto, “Supporting Parallel Applications on Cluster of Workstations: The Intelligent Network Interface Approach”, Georgia Institute of Technology, College of Computing, Atlanta, *GA. HPDC-6*, 1997.

[45] Rexford J., W. Feng, J. Dolter, K. Shin, “PP-MESS-SIM: A Flexible and Extensible Simulator for Evaluating Multicomputer Networks”, IEEE Transactions on Parallel and Distributed Systems, vol. 8, no. 1, 1997.

[46] Silla F., J. Duato, “Improving the Efficiency of Adaptive Routing in Networks with Irregular Topology”, IEEE Proceedings of the Fourth International Conference on High-Performance Computing, pp. 330, 1997.

[47] Ho W. H., T. M. Pinkston, “A Clustering Approach in Characterizing Interconnection Networks”, SMART Interconnects Group., IEEE Proceedings of the Fiftheen International Conference of High Performance Computing, pp. 277-284, December 1998.

[48] Delgado-Frias J. G., J. Nyathi, D. H. Summerville. “A Programmable Dynamic Interconnection Network Router With Hidden Refresh”, IEEE Transactions on Circuits and Systems 1. Vol. 45, pp. 1182-1190, November 1998.

[49] Smai A. H., L. E. Thorelli, “Global Reactive Congestion Control in Multicomputer Networks”, IEEE Fifth International Conference on High Performance Computing, p. 179, 1998.

[50] Tanembaum Andrew S., *Modern Operating Systems*, Prentice Hall, second ed., 2001.

[51] Ai J., Y. Li, T. Wang, K. Shum, “100 x 100 Optoelectronic Cross-Connect Interconnects Using OPTOBUS”, IEEE Journal of Lightwave Technology, vol. 17, no. 5, 1999.

[52] Afsahi A., N. J. Dimopoulos, “Hiding Communication Latency in Reconfigurable Message-Passing Environments”, Technical Report ECE-99-3, University of Victoria, Victoria, B. C., Canada.

[53] Hamblen J. O., H. L. Owen, S. Yalamanchili, B. Dao, “An Undergraduate Computer Engineering Rapid Systems Prototyping Design Laboratory”, IEEE Transactions on Education. vol. 42, no. 1, 1999.

[54] Delgado-Frías J. G., Andy Yu, J. Nyathi. “A Dynamic Content Addressable Memory Using A 4-Transistor Cell”, IEEE Third International Workshop on Design of Mixed-Mode Integrated Circuits and Applications, pp. 110-113, Puerto Vallarta, Mexico, July 1999.

[55] Baydal E., P. López, J. Duato, “A Simple and Efficient Mechanism to Prevent Saturation in Wormhole Networks”, IEEE Proceedings of the 14th International Symposium on Parallel and Distributed Processing, p. 617, 2000.

[56] Galatopoullos D. G., E. S. Manolakos, “Parallel Processing in Heterogeneous Cluster Architectures Using JavaPorts”, ACM Crossroads, vol. 5, no. 3, pp. 26-32, 1999.

[57] Yang Y., L. Zhang, J. K. Muppala, S. T. Chanson, “Bandwidth-delay Constrained Routing Algorithms”, Computer Networks: The International Journal of Computer and Telecommunications Networking, vol. 42, no. 4, pp. 503-520, July 2003.

[58] Lyer S., N. W. McKeown, “Analysis of the Parallel Packet Switch Architecture”, IEEE/ACM Transactions on Networking, vol. 11, no. 2, pp. 314-324, April 2003.

[59] Mneimneh S., V. Sharma, Kai-Yeung S, “Switching Using Parallel Input-Output Queued Switches With No Speedup”, IEEE/ACM Transactions on Networking, vol. 10, no. 5, 2002.

[60] Nyathi J., J. G. Delgado-Frías, “A Hybrid Wave Pipelined Network Router”, IEEE Transactions on Circuits and Systems-1, vol. 49, no. 12, 2002.

[61] Leonardi E., M. Mellia, F. Neri, M. A. Marsan, “On the Stability of Input-Queued Switches with Sapeed-Up”, IEEE/ACM Transactions on Networking, vol. 9, no. 1, 2001.

[62] Jiménez A. F., De Luca P. A., “Memoria Autocompactante para Almacenar Mensajes en Sistemas Paralelos”, 1er Congreso Nacional de Ingeniería en Computación y Sistemas Electrónicos, Universidad Autónoma de Tlaxcala, Apizaco, Tlax., octubre 2001.

[63] Lotker Z., B. Patt-Shamir, “Nearly Optimal FIFO Buffer Management for Two Packet Classes”, Computer Networks, vol. 42, pp. 481-492. Elsevier, 2003.

[64] Awerbuch B., Y. Azar, S. Plotkin, O. Waarts, “Competitive Routing of Virtual Circuits with Unknown Duration”, Journal of Computer and System Sciences vol. 62, pp. 385-397. Ideal Library, 2001.

[65] Pasqua D´Ambra, Marco Danelutto, Daniela di Serafino, “Advanced Environments for Parallel and Distributed Computing”, Parallel Computing, vol. 28, no. 12, pp. 1635-1636, December 2002.

[66] Su Xun, G. de Veciana, “Predictive Routing to Enhance QoS for Stream-Based Flows Sharing Excess Bandwidth”, Computer Networks: The International Journal of Computer and Telecommunications Networking, vol. 42, no. 1, pp. 65-80, May 2003.

[67] Song Y., A. Wool, B. Yener, “Combinatorial Design of Multi-ring Networks with Combined Routing and Flow Control”, Computer Networks, vol. 41, pp. 247-267. Elsevier Science B. V. 2003.

[68] Patterson D. A., J. L. Hennessy, *Computer Architecture*: *A Quantitative Approach*, Morgan Kaufmann, 1996.

[69] Tanenbaum Andrew S., *Computer Networks*, Prentice Hall, 4th ed. 2003.

[70] Tanenbaum Andrew S., *Organización de Computadoras: un Enfoque Estructurado*, Prentice Hall, Cuarta Edición, 2000.

[71] Leighton T., “Average Case Análisis of Greedy Routing Algorithms on Arrays”, 2nd Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 2-10, Crete, Greece, 1990.

[72] Abrial J. R., *The B Book*, Cambridge University Press, 1996

[73] De Luca P. A., M. Peña G., A. Jiménez F., “Memoria Escalable Multisegmentada Diseñada con VHDL”, II Simposium Internacional, “Las Humanidades en la Educación Técnica Ante el Siglo XXI”, ESIQIE-IPN, Septiembre 2000.

[74] De Luca P. A., M. Peña G., A. Jiménez F., “Memoria de Alto Rendimiento Para Enrutamiento de Mensajes”, Sociedad Mexicana de Instrumentación, SOMI XV, Guadalajara Jal., octubre 2000.

[75] Jiménez F. A., “Aplicaciones de los Microprocesadores en la Industria”, II Semana de Ingeniería Eléctrica, Escuela Superior de Ingeniería Mecánica y Eléctrica del IPN, México D. F., Octubre 1995.

[76] Jiménez F. A., F. Polanco., J. Hernández T., M.Castro, “Simulador NxSPARC”, VII Congreso Interuniversitario de Eléctrica, Computación y Electrónica, Universidad Autónoma de Baja California, Ensenada B.C., Marzo 1997.

[77] Jiménez F. A., De Luca P. A., “Diseño de un Modelo Arquitectural de un Controlador de Mensajes para Sistemas Multicomputadores”, X Reunión de Otoño de Comunicaciones y Computación y Exposición Industrial ROC&C99, Acapulco Gro., Noviembre 1999.

[78] Jiménez F. A., De Luca P. A., “Diseño del Control de Flujo de un Ruteador Flexible y Adaptable en Sistemas Multicomputadores”, 7ª Conferencia de Ingeniería Eléctrica, Centro de Investigación y de Estudios Avanzados del IPN, México D.F., Septiembre 2001.

[79] Jiménez F. A., De Luca P. A., “Buffer Autocompactante para Control de Flujo en Sistemas Paralelos”, Sociedad Mexicana de Instrumentación, SOMI XVI. Querétaro Qro, Octubre 2001.

[80] Jiménez F. A., De Luca P. A., “Flow Control Design For a Flexible and Adaptive Router in Parallel Systems”, Journal of Applied Sciences & Technology, Vol. 2, No. 1, March 2004.

[81] Suryanarayana B. T., Delgado-Frias José G., “A VLSI Self-Compacting Buffer for Priority Queue Scheduling”, IASTED International Conference on Circuits, Signal and Systems (CSS 2003), Cancún, Mex, May 2003.

[82] Peña Guerrero M., “Captura de Múltiples Eventos MIDI en Tiempo de Ejecución”, Inédita, México, Tesis presentada para aspirar al grado de Doctor en Ciencias en Ingeniería Eléctrica, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, 2005.

[83] Xilinx, *VHDL*: *Reference Guide*, Xilinx Inc., USA, 1999.

[84] Xilinx, *The Programmable Logic Data Book 1999*, Xilinx Inc., USA, 1999.

[85] Nutt, G., *Operating Systems*, Pearson Education, Inc., 3rd ed., 2004.

[86] Caci, *User´s Manual for MODSIM II*, CACI Products Company, USA, 1995.

Glosario

Ancho de Banda: Razón a la cual se realizan las operaciones.

Arquitectura Harvard: Arquitectura que utiliza una memoria para los datos y otra para las instrucciones del programa en ejecución.

ASIC: *Application Specific Integrated Circuit* (circuito integrado de aplicación específica). Circuitos electrónicos integrados que pueden ser diseñados en forma directa por los ingenieros de aplicaciones.

BAC: *Buffer AutoCompactante*, memoria utilizada como *buffer* que mantiene compactada, por si misma, su área ocupada por los datos almacenados en ella.

Buffer: Memoria que se utiliza como dispositivo de almacenamiento temporal de los mensajes que transitan entre los nodos de un sistema de procesamiento paralelo. Un *buffer* puede estar asociado con un canal físico de entrada o salida dentro de un *switch* o un ruteador.

Caché: Memoria relativamente pequeña, de alta velocidad, usada para mejorara la latencia de memoria. Mantiene los datos e instrucciones requeridos por el procesador durante la ejecución de un programa.

CAD Tool: *Computer Aided Design* (diseño asistido por computadora). Conjunto de programas integrados en un ambiente computacional que interactúa con el usuario y facilita el desarrollo de algún sistema con ayuda de la computadora.

CLB: *Configurable Logic Block* (bloque lógico configurable). Es el bloque de construcción básico de la familia de arreglos de celdas lógicas de Xilinx.

Cola FIFO: *First Input First Output* (primero-que-entra primero-que-sale). Arreglo de datos donde en primer dato que se deposita en la cola es el primero que se extrae de ella.

Crossbar: arreglo de pares de contactos operados individualmente, en el cual hay un par para cada combinación entrada-salida.

Escalabilidad: Un sistema escalable intenta evitar los límites de diseño en su desempeño, propios del aumento de recursos agregados a dicho sistema.

Ethernet: Red de difusión basada en bus con control descentralizado, que por lo general funciona de 10 Mbps a 10 Gbps. Las computadoras que están en una red ethernet pueden transmitir siempre que lo deseen; si dos o más paquetes entran en colisión, cada computadora espera un tiempo aleatorio y lo intenta de nuevo más tarde.

Flit*:* Unidad de control de flujo. Es la unidad mínima de información, transferible sobre un enlace, que puede aceptarse o rechazarse.

FPGA: *Field Programmable Gate Array* (arreglo de compuertas programables en campo). Circuito integrado de alta densidad que contiene arreglos de compuertas lógicas que pueden ser configurados como sistemas digitales en una sola pastilla para implementar un circuito de propósito específico (ASIC).

Interleaving: Técnica que permite el traslape de operaciones de memoria. Consiste en dividir la memoria en dos o más bancos. Cada banco realiza una operación en cada ciclo de memoria. Cada banco puede realizar una operación sobre una palabra diferente al mismo tiempo.

Latencia: Intervalo de tiempo entre el inicio y la terminación de una operación.

MIMD: *Multiple Instruction stream over Multiple Data stream* (flujo de múltiples instrucciones sobre múltiples datos). Tipo de computadora paralela en donde cada procesador tiene su propia unidad de control y sus propios datos.

ModSim: Lenguaje para modelado y simulación de eventos discretos.

OSI: *Open System Interconection* (Interconexión de Sistemas Abiertos). Modelo basado en una propuesta desarrollada por la ISO (Organización Internacional de Estándares) como un primer paso a la estandarización internacional de los protocolos utilizados en varias capas. Trata sobre sistemas abiertos a la comunicación con otros sistemas.

Phit*:* Unidad de información que puede transferirse a través de un canal físico en un solo paso o ciclo.

Pipeline*:* Técnica para mejorar el número de instrucciones ejecutadas por segundo. Consiste en dividir la tarea de ejecución de cada instrucción en un conjunto fijo de actividades pequeñas llamadas etapas de procesamiento. Cada etapa opera sobre un objeto diferente, por lo que es posible procesar más objetos en un período de tiempo.

Prefetch: Búsqueda de instrucciones antes de que sean requeridas por la unidad de procesamiento.

Ruteador: Dispositivo de red que permite establecer una conexión indirecta entre los nodos de procesamiento de un sistema paralelo. Envía y recibe datos con base en la dirección fuente y destino. Opera en la capa de red del modelo OSI.

Superescalar: Procesador que utiliza múltiples unidades funcionales para ejecutar varias instrucciones por ciclo del procesador.

Switch: Dispositivo de red que permite establecer una conexión indirecta entre los nodos de procesamiento de un sistema paralelo. Envía y recibe datos con base en la dirección destino. Opera en la capa de enlace del modelo OSI.

TCP/IP: Conjunto de protocolos completo que incluye IP (*Internet Protocol*), TCP (*Transmission Control Protocol*) y los protocolos de aplicación asociados a ellos.

VHDL: *VHSIC Hardware Description Language* (lenguaje de descripción de *hardware* para VHSIC). Lenguaje computacional de alto nivel para diseñar sistemas electrónicos digitales.

VHSIC: *Very High Speed Integrated Circuits* (circuitos integrados de velocidad muy alta).

VLIW: *Very Long Instruction Word* (Palabras de instrucción muy largas). Cada palabra especifica varias instrucciones, de menor tamaño, que se ejecutan en forma simultánea.

VLSI: *Very Large Scale Integration* (integración a escala muy alta). Técnica de fabricación de circuitos electrónicos integrados en espacios muy reducidos.

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL I.P.N.

Los abajo firmantes, integrantes del jurado para el examen de grado que sustentará el M. en C. Armando Jiménez Flores, declaramos que hemos revisado la tesis titulada:

"Buffer AutoCompactante para Switches y Ruteadores en Sistemas Paralelos"

Y consideramos que cumple con los requisitos para obtener el grado de Doctor en Ciencias, en la especialidad de Ingeniería Eléctrica opción Computación.

ATENTAMENTE

Dr. José Delgado Frías

Dr. Miguel Ángel León Chávez

Dr. Sergio Victor Chapa Vergara

Dr. José Antonio Moreno Cadenas

Dr. Adriano De Luca Pennacchia

Av. Instituto Politécnico Nacional # 2508 Col. San Pedro Zacatenco México, D.F. C.P. 07360
Tel. 50-61-38-00 Fax: 57-47-70-02