Universitat Autònoma de Barcelona
Escola Tècnica Superior d'Enginyeria

"SÍNTESIS DE CIRCUITOS DIGITALES ASÍNCRONOS APLICADOS A COMUNICACIONES: UNA APROXIMACIÓN MICROPIPELINE PARA QoS-ATM"

Tesis presentada por
Rubén V. Alarcón Matutti
para optar al grado de
Doctor en Ing. Electrónica

Bellaterra, julio de 2003

El Dr. Jordi Carrabina Bordoll, profesor titular de la Escuela Técnica Superior de Ingenieria, Universitat Autònoma de Barcelona

CERTIFICA:

Que la presente Tesis ha sido realizada bajo mi dirección por el Sr. Rubén V. Alarcón Matutti, para optar al grado de Doctor en Ing. Electrónica.

Dr. Jordi Carrabina i Bordoll

Bellaterra, julio de 2003

Mi más sincero agradecimiento a los amigos de UAB,
especialmente de la Unidad de Microelectrónica,
y al Instituto Catalán de Cooperación Iberoamericana.

"L'ordre és el caos en repòs"
David Escamilla, Poeta Catalán

INDICE

LISTA DE FIGURAS

LISTA DE TABLAS

CAPÍTULO I

INTRODUCCIÓN

El diseño de circuitos asíncronos es un área emergente y es una alternativa para los problemas que se presentan en el clásico diseño síncrono. Actualmente en los sistemas digitales síncronos se tiene un constante incremento de frecuencia del reloj y, por consiguiente, mayor consumo de potencia. Por lo cual y añadido a otros factores se imponen severas restricciones en el diseño como la distribución de la señal de reloj o problemas derivados del skew y jitter.

Por otro lado, actualmente en el clásico diseño digital síncrono y su aplicación al área de comunicaciones se tienen mayores demandas en las prestaciones de los circuitos que sirven de soporte en dicha área, por ejemplo compatibilidad electromagnética, velocidad y consumo de potencia entre otros.

En campo de redes de comunicaciones y los conmutadores ATM (Asynchronous Transfer Mode), la QoS (Quality of Service) es un aspecto crucial para la gestión del tráfico y se define como un conjunto de normas, establecidas por organismos como ITU-T y ATM forum, que permite ya sea a un proveedor, a una compañía o a un usuario de telecomunicaciones recibir un nivel de servicio predecible en cuanto a parámetros de ancho de banda (throughput) y retardo (latencia). El problema de la QoS actualmente es un desafío para los diseñadores, concretamente la implementación eficiente del subsistema QoS e igualmente superar o minimizar los problemas que presenta el empleo de circuitos síncronos.

El contexto anterior, la presente tesis tiene dos vertientes principales: los circuitos asíncronos y la problemática de la QoS para el protocolo de comunicaciones ATM. La tesis se centra en la investigación de una técnica de diseño asíncrona, lo cual implica la eliminación del reloj global, y que facilite la implementación de arquitecturas que eviten o al menos minimicen los problemas encontrados en los equivalentes circuitos síncronos como alineamiento de fase, sincronización, skew y jitter.

Dentro de los objetivos principales de la tesis se establece una metodología de diseño asíncrono independiente de la tecnología, y como circuito demostrador se aborda la implementación asíncrona de la Unidad de Clasificación que es el subsistema principal en la gestión QoS-ATM. Se utiliza el estilo de diseño micropipeline que, además de otras ventajas intrínsecas, permite un adecuado balance de la relación área/velocidad y facilidades para test. La arquitectura implementada en FPGA permite satisfacer los requerimientos de throughput y latencia de las normas de transmisión OC-12 (622 Mbits/seg) y OC-48 (2.5 Gbits/seg).

1.1 Motivación de la tesis

La presente tesis se fundamenta en la investigación de las técnicas de circuitos asíncronos y su aplicación a circuitos de comunicaciones, concretamente se aborda la problemática del subsistema de calidad de servicio QoS (Quality of Service) para el protocolo ATM (Asynchronous Transfer Mode). Dentro de las motivaciones de la presente tesis se consideran los siguientes puntos:

A. Respuesta alternativa.

Los circuitos asíncronos se presentan como una respuesta alternativa a los actuales problemas del clásico diseño digital síncrono, por ejemplo aspectos como el consumo de potencia, compatibilidad electromagnética, distribución de la señal de reloj, diseño modular, son cruciales en las actuales aplicaciones de los circuitos digitales.

B. Desarrollo de la metodología de diseño.

Los circuitos asíncronos son materia de investigación en universidades y empresas, actualmente se exploran diferentes aproximaciones desde metodologías de ámbito académico hasta metodologías más cercanas a las herramientas comerciales existentes en el mercado.

Por otro lado, para lograr la diseminación y empleo de las técnicas de diseño asíncrono es necesario contar con las herramientas de soporte en todas las etapas de la metodología de diseño. Por lo cual es muy importante el desarrollo de una metodología que permita la implementación de circuitos asíncronos tomando en cuenta la realidad del mercado actual de diseño digital.

C. Justificación de la aplicación en comunicaciones.

En la actualidad no existen herramientas de diseño asíncrono que estén específicamente orientadas a la síntesis de circuitos de comunicaciones, algunas herramientas tienen dicha posibilidad pero aún su desarrollo es incipiente. Las cualidades y ventajas de los circuitos asíncronos están estrechamente ligadas a los campos específicos de aplicación donde se empleen. Por lo cual es necesario desarrollar aplicaciones de los circuitos asíncronos en comunicaciones, donde la justificación de la implementación implica una previa selección del circuito y la evaluación de su funcionalidad.

Los circuitos de soporte en comunicaciones digitales presentan características determinadas por el tipo de información y tratamiento que se realizan sobre los datos. Por ejemplo, en ciertos circuitos es importante la propiedad de un comportamiento típico en vez de un comportamiento para el peor caso. En este sentido y al nivel de sistema los circuitos asíncronos presentan cualidades de adaptarse, en un instante determinado, a los requerimientos de procesamiento de los datos de tráfico.

D. Problemática de la conmutación ATM.

En el protocolo ATM y específicamente en los conmutadores existen aspectos no resueltos, debido a los requerimientos actuales de consumo de potencia y throughput,

y la dificultad de lograr implementaciones óptimas en aspectos como por ejemplo la latencia, sincronización, skew y jitter.

El empleo de circuitos asíncronos que implementen las funciones de un conmutador permite encontrar soluciones óptimas para resolver dichos aspectos. Aunque la presente investigación se limita al subsistema de QoS, los resultados podrían aprovecharse para otros subsistemas de un conmutador ATM.

1.2 Visión general de la tesis

La presente tesis se enmarca en el campo de la investigación de circuitos asíncronos y su aplicación a circuitos de comunicaciones digitales. Dentro de las ideas básicas que soportan la investigación y que dan una visión general de esta tesis se pueden citar:

A. Un punto de vista de aplicación.

Se investiga el empleo de las técnicas de los circuitos asíncronos considerando sus ventajas y limitaciones, desde un enfoque de ingeniería que permita transportar conceptos teóricos en procedimientos prácticos y realizables con el actual estado del arte del diseño digital.

En la actualidad se tiene en la literatura informes muy diversos sobre los circuitos asíncronos aunque, en su gran mayoría, la implementación de aplicaciones sólo ha tenido interés académico. Por esto es necesario dar un mayor peso a la aplicación y encontrar nichos de aplicación de los circuitos asíncronos.

B. Empleo de herramientas de diseño comerciales.

A lo largo de la investigación se da la mayor importancia al hecho que se pueda tener el soporte de las herramientas comerciales convencionales y minimizar la pérdida de eficiencia en la implementación de un circuito asíncrono.

Un aspecto que impide el empleo masivo de los circuitos asíncronos es que las técnicas de los circuitos asíncronos requieren herramientas específicas desde las etapas de síntesis alto nivel hasta etapas de diseño físico a las cuales los diseñadores no tienen acceso o a las cuales no están familiarizadas. Por lo cual una metodología en la cual se mantenga, en todas sus etapas, cerca de las herramientas de diseño comerciales permite un mayor acceso y diseminación del diseño de circuitos asíncronos.

C. Prototipaje independiente de la tecnología.

Para la validación de las hipótesis tanto de aplicación y de metodología empleada se realiza la implementación del circuito demostrador. La implementación en una tecnología actual permite la evaluación real del rendimiento de los principales parámetros y así mismo proponer soluciones a las dificultades encontradas en la implementación y evaluación del prototipo. Un aspecto determinante en el presente trabajo es que la implementación sea lo mayor posible independiente de la tecnología y como subsistema se pueda conectar de forma modular a un conmutador ATM de bajo

costo, además, que la implementación propuesta optimice el rendimiento global del sistema.

1.3 Estructura de la tesis

Los capítulos de la tesis se presentan estructurados de la siguiente forma:

En el capítulo II, se revisan brevemente los conceptos teóricos importantes de los circuitos asíncronos, las actuales herramientas de diseño asíncrono, alternativas para la implementación y se describen las ventajas y limitaciones. Luego se establece el diagrama de flujo de la metodología de diseño, adicionalmente este capítulo permite establecer la nomenclatura adecuada que será utilizada en los capítulos posteriores.

En el capítulo III se fundamenta la necesidad del empleo de circuitos asíncronos en comunicaciones y centrando sobre el protocolo ATM, igualmente se define el concepto de Calidad de Servicio y se especifica la arquitectura asíncrona de la Unidad de Clasificación QoS-ATM.

En el capítulo IV, se presenta la implementación de los principales módulos que constituyen la biblioteca básica para la realización óptima de circuitos asíncronos, se establecen los requisitos, especificación y la validez funcional del diseño micropipeline.

En el capítulo V, se realiza el análisis del throughput de las arquitecturas paralela y serie que implementan el subsistema de Unidad de Clasificación. Se reseña aspectos que facilitan el test respectivo y se establece las prestaciones del subsistema con sus ventajas y limitaciones.

Finalmente en el capítulo VI, se representan los resultados de las implementaciones y las mediciones de los principales parámetros de prestaciones, en el prototipo demostrador se emplean FPGAs de Altera y Xilinx a fin de realizar una evaluación comparativa y selección para la implementación de la Unidad de Clasificación.

En las conclusiones se resumen los aportes principales de la tesis y las perspectivas de un trabajo futuro. En los anexos se tienen los más importantes detalles, ampliaciones de conceptos y puntos de apoyo. En las referencias se citan las principales fuentes utilizadas. Adjunto a la tesis se presenta el apéndice denominado “Biblioteca de Módulos de Control y Procesamiento” donde se tiene los listados de los programas elaborados en la presente tesis.

CAPÍTULO II

FUNDAMENTOS GENERALES DE DISEÑO ASÍNCRONO

2.1 Introducción

El presente capítulo contiene los fundamentos generales de diseño asíncrono que están relacionados con el tema de tesis. Al mismo tiempo, permite establecer la nomenclatura que será utilizada en los capítulos posteriores. Se revisan brevemente los conceptos teóricos importantes como la comunicación asíncrona entre dos bloques, aspectos del control de flujo, tipos de canal de datos, protocolos de señalización y las formas de realizar la comunicación.

Se aborda los estilos de diseño asíncrono de acuerdo a los modelos más importantes, se hace hincapié en el modelo micropipeline que es la base del diseño implementado. Se revisan las herramientas de especificación como los STGs (Signal Transition Graphs), se describen las herramientas CAD reportadas en la literatura que tienen orientación al diseño asíncrono, y se remarcan las características que puedan facilitar o limitar su empleo para el diseño de circuitos de comunicaciones.

Otro aspecto de importancia es la implementación física del circuito, en el presente trabajo se intenta que el diseño sea lo menos dependiente de la tecnología y esté cerca de las herramientas CAD comerciales. Por lo cual se revisan las tecnologías que actualmente se emplean en la implementación de circuitos asíncronos y sirven de referencia para la implementación del prototipo demostrador.

Finalmente, se considera la aplicación específica en los circuitos de comunicaciones y sus requerimientos. Se describen las cualidades generales de los circuitos asíncronos y se establece el diagrama de flujo de la metodología de diseño empleada en el presente trabajo.

2.2 Definiciones de diseño asíncrono

Dentro del diseño asíncrono, se tienen definiciones acerca la comunicación asíncrona entre dos bloques, aspectos como el control de flujo, tipos de canal de datos, protocolos de señalización y las formas de realizar la comunicación son descritos estableciendo las características que facilitan la implementación de un diseño asíncrono.

2.2.1 Comunicación de datos

Para el caso más simple, comunicación de datos asíncrona entre dos bloques, se tiene un receptor y un transmisor. Se tiene un bloque activo el cual inicia la comunicación y un bloque pasivo que responde al requerimiento del bloque activo. El receptor debe conocer en que instante el dato enviado es válido.

En esta comunicación asíncrona, el control de flujo es implementado considerando que el bloque que recibe los datos activa una señal "data release" para indicar que ya no necesita los datos, el transmisor no puede enviar datos nuevos si no ha recibido dicha señal. El bloque que inicia envía una señal de Req (request) y el bloque que responde envía una señal de Ack (acknowledge). Los canales de datos pueden ser tipo "push" o tipo "pull", definidos a continuación.

Canal push: Los datos son enviados por el bloque activo, el cual envía una señal de Req al bloque pasivo para indicar que puede leer los datos. El bloque que recibe los datos activa la señal "data release" que es la señal Ack.

Canal pull: En este caso el bloque activo recibe los datos del bloque pasivo. El bloque que envía los datos activa la señal Ack. El bloque activo envía la señal "data release" es la señal Req.

2.2.2 Protocolos de señalización

Se pueden definir dos protocolos de señalización: la señalización de transición o de 2 fases y la señalización de nivel o de 4 fases. A fin de una mejor claridad de los conceptos, para los niveles lógicos de una señal se asume lógica positiva, donde un nivel lógico "0" corresponde al menor valor de voltaje y el nivel lógico "1" corresponde al mayor valor de voltaje.

Señalización de transición.

Un evento es reconocido por el cambio del valor lógico de la señal. Un cambio desde un de nivel lógico 0 a 1 es equivalente un cambio lógico de 1 a 0. Un handshake NRZ (non-return-to-zero) se basa en la transición de las señales, consiste de dos fases, en la primera fase el receptor lee los datos y en la segunda fase el transmisor prepara los nuevos datos.

Para un canal push luego de que los datos son validos en las líneas de datos, el transmisor de datos activa el handshake mediante una transición en la línea de request. Los datos deben permanecer estables hasta que el receptor haya leído los datos y envía la señal de "data release" mediante una transición en la línea de acknowledge.

En la Figura 2.1 se muestra dos handshakes para un canal push, en la primera ambas líneas de control realizan una transición de subida y en la segunda realizan una transición de bajada. Se asume que las líneas de control inicialmente se encuentran en estado lógico 0, sin embargo, puede ser cualquier estado inicial, lo importante son las transiciones de estados.

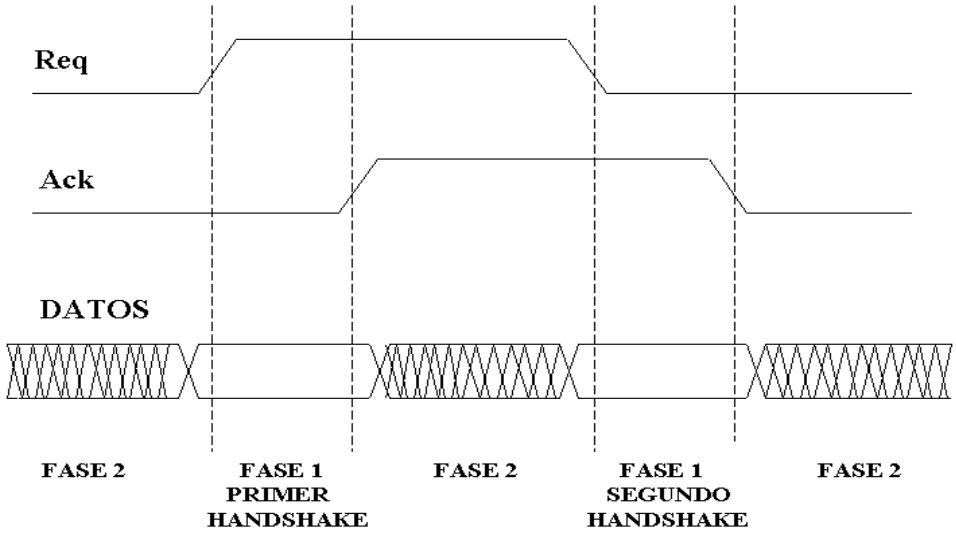

Fig. 2.1: Dos handshakes en un canal push

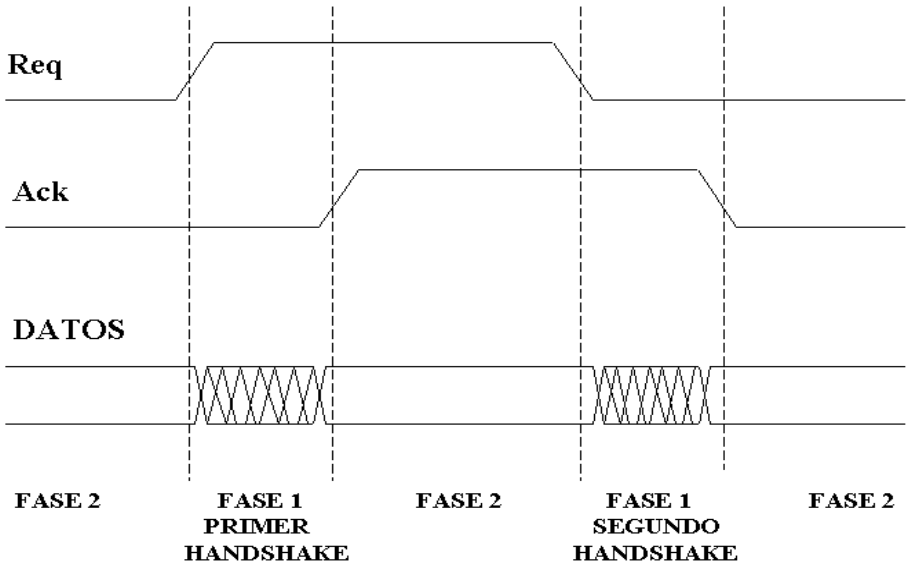

Fig. 2.2: Datos válidos para un canal pull

En la Figura 2.2 para un canal pull se muestra que los datos deben ser válidos entre los handshakes. En el caso de un canal push los datos deben ser válidos durante el handshake.

Señalización de nivel.

En handshake RTZ (return-to-zero) se basa en los niveles de las señales, el nivel lógico indica el significado de la señal. Consiste de una transición de subida y de una transición de bajada en las líneas de control, se tiene 4 fases para completar el ciclo completo de handshake. Se tiene dos transiciones redundantes lo cual implica un mayor consumo de potencia y menor velocidad pero el uso de niveles es más simple que el uso de transiciones ya que requiere menos circuitos de control [41].

Para la validez del protocolo se tienen varias alternativas y dependiendo del canal push o pull. Los protocolos de validez fijan los intervalos durante los cuales los datos deben ser válidos respecto a las transiciones.

2.2.3 Interfaces bundled-data y dual rail

En un sistema asíncrono no se tiene una señal de reloj, el receptor debe tener alguna señalización de cuando el transmisor inicia el envío de datos y cuando finaliza la transferencia. Entre las formas de realizar la comunicación se tiene:

Bundled data.

La Figura 2.3 muestra un canal de comunicación bundled data tipo push. Se tiene un bit extra que indica cuando los datos son válidos, esto significa una línea adicional a las líneas de datos. Dicha señal "data release", para todos los bits de datos, se envía en una línea separada de las líneas de datos. Dependiendo del canal de comunicaciones push o pull será una señal de request o de acknowledge. La restricción de tiempos implica que la señal data release se activa después que los datos se han estabilizado y pueden transferidos correctamente.

Este tipo de codificación bundled-data permite un ahorro de lógica y líneas de conexión, la dificultad radica en que el bloque de retardo introducido debe ser calculado como el retardo mínimo necesario y suficiente en cada etapa.

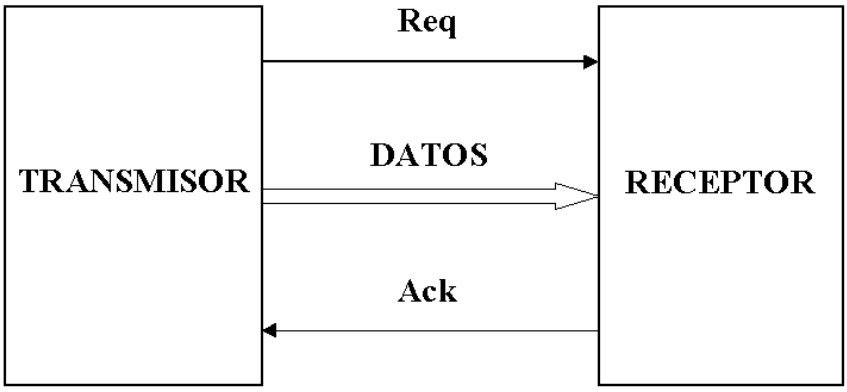

Fig. 2.3: Canal bundled-data tipo push

Dual rail.

Una forma de interfaz "dual rail" se tiene cuando se añade un valor lógico "null" a cada bit de datos, este valor lógico sirve como un separador de datos. Puede ser implementado con una lógica de 3 niveles a costo de incrementar la complejidad y que puede ser muy vulnerable al ruido. Otra posibilidad es usar dos líneas para cada bit de datos y codificar los valores lógicos null, 0, 1 en dichas líneas. La señal de datos válidos

está implícita en los datos. En un handshake de transiciones el 0 lógico es enviado mediante la activación de la línea 0 y el 1 lógico mediante la activación de la línea 1. Informes de diseños en [1], [28], indican que la desventaja radica en el excesivo número de líneas de conexión y usualmente se tiene una mayor área y mayor consumo de potencia comparado por ejemplo con una comunicación bundled-data y single rail.

2.3 Estilos de diseño asíncrono

Los estilos de diseño asíncrono se establecen por los modelos empleados, a continuación se describen los principales modelos estableciendo sus puntos favorables y desventajas. El modelo micropipeline ofrece un balance en la relación coste/prestaciones, se enfoca este modelo como base del diseño implementado y las ventajas que ofrece en cuanto al test.

2.3.1 Modelos de diseño asíncrono

Modelo bounded delay.

El modelo "bounded delay" asume que los retardos en todos los elementos del circuito y líneas de conexión son conocidos o al menos están acotados. Usualmente este modelo está relacionado con el modo de operación fundamental y se denominan circuitos Huffman. El requisito principal y necesario es que el sistema sea estable antes que ocurra alguna transición de las entradas externas. Esto caracteriza un circuito de modo fundamental. Los circuitos son diseñados de manera similar a los circuitos síncronos y se emplean las técnicas usuales.

Existen algunos casos particulares donde se puede relajar las restricciones de un circuito de modo fundamental [2]. Otra metodología de tipo burst-mode [3], donde se puede emplear varias señales independientes de reloj locales.

Modelo delay insensitive.

Se asume que los retardos tanto de los elementos y conexiones no están acotados. Esto significa que el receptor de una señal debe comunicar al transmisor que ha recibido la información, esta función es realizada por un circuito de detección de finalización en el receptor. El transmisor debe esperar hasta que le llegue la señal de término y poder enviar el siguiente dato. En este modelo se puede usar un esquema de handshaking de 2 fases o de 4 fases para los circuitos de control. La transferencia de datos puede ser dual rail.

El empleo de las puertas comunes de una sola salida tales como AND, OR, XOR, limitan los tipos de circuitos que se pueden implementar [4]. Una posibilidad es crear un conjunto de módulos básicos que obedecen a las propiedades delay-insensitive y que incluyan salidas múltiples, por ejemplo el modelo I-net [5] o el modelo NLC utilizado por Theseus Logic Inc [29] donde se usa dos líneas para cada señal. Dichos módulos básicos deben ser estandarizados y certificados, generalmente son simples para facilitar la síntesis. No es posible realizar optimizaciones de la estructura de estos módulos y

para cada implementación tecnológica es necesario generar un nuevo conjunto de módulos.

Modelo speed independent.

El modelo "speed independent" asume que mientras los retardos de puertas no están acotados, todos los retardos de las líneas de conexión son de valores despreciables o al menos son menores que el retardo mínimo de puerta. Una variación es el modelo QDI (quasi-delay-insensitive) donde se asume que ambos retardos de puertas y líneas son no acotados pero se define los "isochronic fork", que son derivaciones de líneas de conexión donde la diferencia de los retardos para los diferentes destinos es despreciable. Estos supuestos pueden ser difíciles de implementar en la práctica en las diferentes tecnologías y se restringe a pequeñas áreas concretas.

2.3.2 Modelo micropipeline

El modelo micropipeline, combina señales de termino de operación, señalización de transiciones y elementos de procesamiento bounded-delay y bundled-data, se usa una sola línea para cada bit de dato y una línea adicional de control para cada palabra de datos, se asume que el retardo en la línea de control es mayor que el retardo en cada línea de datos.

Para diseñar circuitos basados en micropipelines, se requieren unos pocos bloques de control los cuales fueron propuestos por Sutherland [6]. Para construir los circuitos de micropipeline se emplea latches controlados por transiciones que permiten mantener los datos estables. Estos latches pueden ser replicados para formar un registro FIFO (firts-in-first-out). Los datos serán desplazados por el registro a una velocidad determinada solo por los parámetros tecnológicos sobre la cual se construye los circuitos lógicos.

En los FIFOs asíncronos como se muestra en la Figura 2.4, usualmente los datos son dibujados desplazándose desde izquierda hacia la derecha. Una vez que la etapa i recibe los datos, esta envía un request a la etapa (i +1) para transferir los datos. Si la etapa (i +1) está vacía captura los nuevos datos y envía un acknowledgment a la etapa i, esto comunica a la etapa i que la transferencia está completa y la etapa i puede aceptar nuevos datos.

Como se observa un micropipeline no tiene reloj global, las diferentes etapas pueden operar a diferentes velocidades y cada etapa completa su operación dependiendo del tipo de procesamiento que realiza. Esta propiedad de elasticidad que permite operar la entrada de datos a una velocidad independiente de la velocidad de salida de datos.

Sutherland sugiere una implementación en cual las etapas vecinas se comunican por dos señales: una es request y otra es acknowledgment. La señal request es denominada Rout en la salida de la etapa i (transmisor) y se denomina Rin en la entrada de la etapa (i +1) (receptor). Igualmente la señal acknowledgment se denomina Ain en la entrada la etapa (i +1) (transmisor) y Aout en la salida de la etapa i (receptor). Se considera un protocolo handshaking de dos fases y en el cual ambas transiciones, bajo-alto y alto-bajo en cualquier señal de control, tienen el mismo significado.

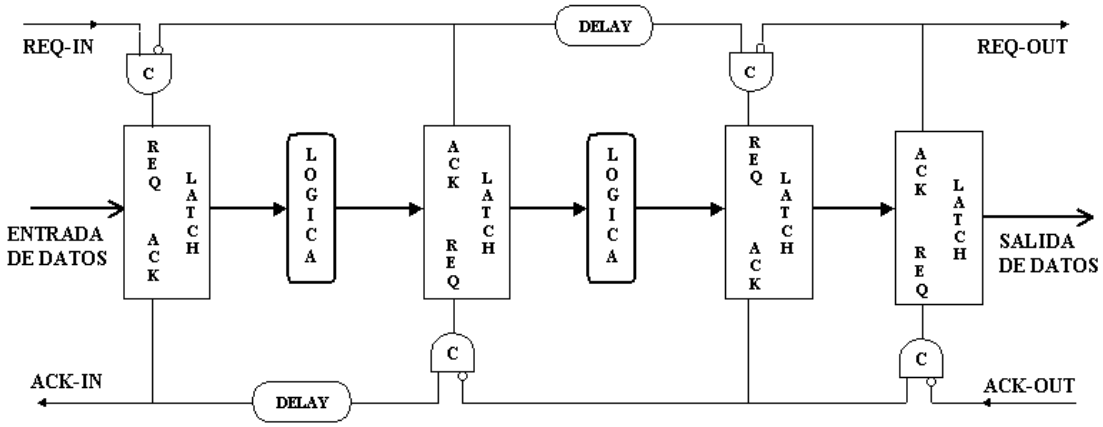

Fig. 2.4: Esquema FIFO del micropipeline asíncrono

2.3.3 Test de circuitos asíncronos

Consideraciones del test.

El test es necesario para validar las implementaciones físicas, por lo cual el test y el diseño para testabilidad (DFT) son un papel importante en la producción industrial de los CI. Por otro lado, el test de circuitos asíncronos es complicado de debido a las restricciones de los estilos de diseño y necesario tener en cuenta algunas características generales de los circuitos asíncronos:

- La no presencia de un reloj global de sincronización disminuye el nivel de controlabilidad sobre los estados y detener la operación en un determinado estado puede ser complicado.
- El empleo de elementos de memoria dificulta la generación del test y las técnicas DFT pueden requerir una gran cantidad de circuitos adicionales.
- La detección de "hazards" y "races" (descritos en la sección 4.2.1) es un aspecto crítico en circuitos asíncronos.
- La lógica redundante introducida para eliminar dichos hazards dificulta detectar fallos stuck-at.

Las alternativas de técnicas de test representan diferentes compromisos entre la calidad y el costo del test. La elección de test apropiado depende de la estructura del circuito, del modelo de fallos utilizado y de los supuestos de retardos del diseño. En general el test necesita emplear varias técnicas que permiten mejorar la observabilidad del circuito final y puede ser ventajoso combinar algunas de las técnicas conocidas.

Test de circuitos micropipeline.

Los circuitos asíncronos micropipeline usan handshakes locales en lugar de un reloj global a fin de sincronizar las operaciones, un fallo s-a (stuck-at) en las señales del handshake causa que los módulos de comunicación entren en un estado de espera en

forma indefinida (deadlock), lo cual es fácilmente observable. Esta propiedad es conocida como autotest (self-cheking o self-diagnostic).

Los circuitos de control que se construyen con elementos C-muller tienen también la propiedad de self-cheking. Cada transición en el circuito debe ser concedida por el receptor de dicha transición. En el caso de una derivación (fork) donde una línea se distribuye a varias puertas, cada una de las puertas debe activar el acknowledge cuando recibe la transición de la señal y solo después puede ocurrir una nueva transición en el punto común de entrada en la derivación.

Una forma de reducir el número de pines I/O de test es almacenar los valores de los puntos de test en un registro interno cuyo contenido puede ser desplazado en ambos sentidos de manera serie. El registro de desplazamiento puede ser implementado de forma síncrona o asíncrona. Los registros en el circuito conforman una cadena de registros scan. De esta forma el problema del test se reduce a generar el test para los bloques combinacionales. El costo de simplificar el test implica un aumento del área y tiempo del test. En esta línea, un aspecto a considerar es el test de redundancia de la lógica desde los pines externos del chip, donde se requiere que el análisis de redundancia sea transmitida al anillo de pads del chip mediante métodos de “scan path”.

En [7] se estudia una técnica de test que se aplica a circuitos micropipeline de macromódulos, la idea es que mientras el circuito opera de manera asíncrona en el modo normal, el modo de operación scan es síncrono y el reloj se propaga en dirección contraria a lo largo del micropipeline, el costo de los circuitos añadidos son similares que para los circuitos síncronos.

Para el compilador Tangram [8] se han desarrollado técnicas y herramientas de “partial scan” de circuitos handshaking. La testabilidad puede ser garantizada desde la especificación de alto nivel mediante una modificación de dicho compilador.

Test de circuitos asíncronos secuenciales.

Para el test de circuitos asíncronos secuenciales, en [9] se establece un procedimiento similar. El ASC (asynchronous sequential circuit) trabaja como un micropipeline: el bloque combinacional realiza las operaciones lógicas y los registros del lazo de realimentación almacenan los estados del ASC.

Los datos hacia al ASC son generados en las entradas primarias por el transmisor mediante la señal Rin, la cual es retardada hasta que los datos de salida se estabilicen en las salidas primarias y en las salidas internas. Luego se produce la señal Rout hacia la siguiente etapa, una vez que se ha guardado el nuevo estado en un registro y se ha recibido la señal Aout, el circuito genera la señal Ain para la anterior etapa, simultáneamente se transfiere el contenido del registro hacia otro registro. Cuando se recibe una nueva señal Rin se repite el proceso.

La técnica desarrollada permite el test de fallos s-a y de retardo en un ASC dentro de un diseño micropipeline. Las entradas internas y salidas de los bloques combinacionales son totalmente controlables y observables a través del scan path. La lógica adicional puede ser estimada solo para un particular caso ya que depende de la complejidad del circuito combinacional.

Test de retardo.

El test debe considerar los retardos de los bloques y se debe determinar si los circuitos fabricados tienen los márgenes de retardo asumidos, para lo cual se necesita un modelo de retardos ya que el modelo s-a no es apropiado.

En el modelo de fallos "path delay" las conexiones entre registros son consideradas. Si el path tiene un retardo fuera del intervalo especificado, el circuito tiene este tipo de fallo. El test implica el uso de equipos complejos, se realiza aplicando los vectores adecuados en secuencias de tiempos apropiados, se captura la salida del circuito y, si difiere de la especificación, significa que hay un fallo de retardo. Debido a que el test se realiza a la velocidad de operación, los hazards puede ser un problema, por lo cual se debe evitar que los propios vectores de test introduzcan hazards.

2.4 Soportes de diseño asíncrono

El proceso de diseño de asíncrono hace uso de diferentes herramientas, desde estructuras para la especificación inicial hasta herramientas CAD para la implementación. Se tienen varias posibilidades de elección que dependen de la aplicación concreta, aunque para el caso de circuitos de comunicaciones no se tiene una herramienta específica. Actualmente se tienen diversos desarrollos que sirven de referencia al trabajo realizado y se describen los más importantes.

2.4.1 Herramientas de especificación

Signal Transition Graphs (STGs).

Especifican los circuitos asíncronos mediante redes de Petri [10], donde las transiciones representan nombres de las señales y expresan cambios de valor en las señales del circuito. Permiten expresar por ejemplo las relaciones causales entre los eventos, la elección entre varios de ellos, su ocurrencia en paralelo. Representan una formalización de los diagramas de tiempos para el caso asíncrono.

La Figura 2.5 muestra las notaciones para las dependencias y las causalidades en los STGs. En la notación STG se tiene:

- Una transición es representada como (+) (transición de subida) o como (-) (transición de bajada).
- Cualquier transición de subida o bajada se indica con asterisco (*), que es equivalente a un cambio de nivel.
- Un cambio al valor opuesto de nivel se indica como (~).
- El estado inicial o reset se indica mediante un punto (•).
- Los arcos de línea de puntos muestran el comportamiento exterior al circuito y los arcos de línea sólida muestran el comportamiento del mismo circuito.

La implementación de STGs, debe estar acompañada de un método para usar los circuitos de control generados por los STGs y realizar automáticamente el

procesamiento general o los algoritmos para asegurar que se cumplen las propiedades de los STGs. Se tienen ampliaciones del modelo general como los STG/IC (Input choice) o STG/NC (non-input choice). El modelo STG/IC permite múltiples transiciones en las entradas y salidas. Los STG/NCs permiten construir todo tipo de STG/IC además de los propios NC.

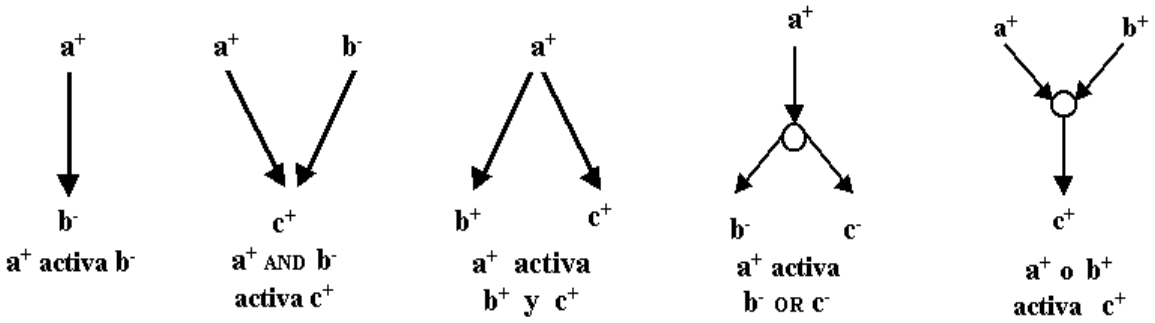

Fig. 2.5: Dependencias y causalidades en los STGs

Es posible sintetizar varios algoritmos mediante STGs pero implica incrementar exponencialmente el costo de la expansión de estados. En [11] se estudia el método de obtener circuitos speed-independent usando simples puertas como AND, OR, C-Muller. En [12] se estudia una aproximación para implementar STG/IC con una sola asignación de estados en un modelo bounded-delay.

CD (Change Diagrams).

Son similares a los STG pero evitan algunas de las restricciones, los CDs tienen una marca inicial y los tokens son puestos en las transiciones. Este modelo no es lo suficiente general para obtener todos los circuitos importantes speed-independent.

CPC (Communicating Process Compilation).

La técnica de CPC traduce programas, escritos en un lenguaje similar a CSP (Communicating Sequential Process), en circuitos asíncronos. Un primer paso es reducir complejas estructuras en combinaciones de procesos simples terminando en una generación reglas que permiten una realización física del circuito. Normalmente los circuitos requieren puertas complejas que no siempre pueden ser descompuestas en puertas simples.

Especificación formal.

La especificación formal es usada para validar diseños, permiten la optimización y verificación de los diseños asíncronos, pero debido a la gran variedad de aproximaciones al diseño asíncrono, es difícil encontrar un método unificado al análisis y verificación de todos los circuitos asíncronos.

Entre los numerosos desarrollos, por ejemplo para el procesador AMULET [13], se ha desarrollado una especificación con modelos detallados para diferentes clases de instrucciones. Se han construido herramientas de verificación para circuitos SI y DI basados en la denominada teoría de “traces” [15]. Para una maquina de estados

asíncrona tipo bounded-delay, en [16] se proponen métodos de análisis de tiempos y de verificación.

2.4.2 Herramientas CAD generales

Se describen algunas de las más importantes herramientas de CAD y los prototipos implementados para aplicaciones asíncronas, se toma en cuenta su relación con la presente aplicación. En las referencias [17][68] se citan otras herramientas CAD adicionales orientadas al diseño de circuitos asíncronos.

Herramienta Tangram.
Philips Research Laboratories, Holanda.
www.semiconductors.philips.com

El compilador de silicio Tangram usa circuitos handshake, ofrece construcciones como paralelismo, comunicación de canales, comunicación a través de variables compartidas, compartir bloques de hardware, implementar RAMs, ROMs y registros. Para las interfaces externas emplea diferentes construcciones como espera de arribo de flancos o condiciones de señales, manipulación directa y el muestreo de señales externas. El circuito handshake soporta comunicaciones handshake y no-handshake y permite varios formatos finales de 4 fases, de 2 fases, single-rail y dual-rail.

Para el test tiene varias herramientas: emplea el analizador de cobertura de test que es útil en el diseño de patrones de test funcional, tiene una opción scan que posibilita una alta cobertura para el test stuck-at usando pocos patrones de test, tiene una herramienta de ATPG y se añaden al netlist facilidades de modo scan aunque el área adicional puede ser considerable.

Herramienta Balsa.
Universidad de Manchester, UK.
www.cs.man.ac.uk/amulet

Los diseños son descritos en el lenguaje propietario Balsa y sintetiza en componentes handshake de una manera similar al compilador Tangram de Philips. Tiene dos posibilidades para la codificación de datos: single-rail y dual-rail, estos componentes handshake son expresados en un formato denominado Breeze, las descripciones pueden ser implementadas en celdas estándar VLSI usando una combinación de Balsa y CAD comercial como Compass, Cadence, Viewlogic para las implementaciones y para las simulaciones Avant TimeMill, Cadence Verilog-XL y Viewlogic Fusion.

Balsa proporciona una interfaz hacia LARD para una simulación funcional en su lenguaje de modelo. No tiene herramientas de generación de test.

Herramienta Lard.
University of Manchester.
www.cs.man.ac.uk/amulet

LARD (Languaje for Asynchronous Research and Development) es un lenguaje desarrollado para describir sistemas asíncronos, aunque se puede considerar un lenguaje

de propósito general. Es una herramienta para análisis y comparación de arquitecturas y el diseñador puede considerar el efecto en el consumo de potencia o en el rendimiento.

En Lard los modelos son simulados para determinar la arquitectura básica, el sistema puede ser implementado mediante herramientas comerciales o mediante Balsa. Si es sintetizado mediante Balsa, Lard puede ser usado para realizar simulaciones de comportamiento y verificar la funcionalidad antes de la etapa final de síntesis. Luego de obtener el netlist, Lard puede ser usado para enlazar al simulador Timemill y poder aplicar los vectores de test.

Lard fue utilizado para modelar el Amulet3i, el controlador DMA, el bus Marble y memorias RAM, ROM. También fue usado para aplicar los vectores de test y validar el diseño del bus Marble.

Herramienta Petrify.
Universitat Politécnica de Catalunya, Spain.
www.lsi.upc.es/~jordic/

Petrify es una herramienta para la síntesis de redes de Petri y controladores asíncronos. Lee una descripción en STG (Signal Transition Graph) y genera una netlist del circuito. Petrify puede sintetizar ya sea circuitos speed-independent o circuitos temporizados, extrae las restricciones de tiempos o el diseñador las suministra, y que deben ser verificadas después de síntesis.

Petrify proporciona diferentes tipos de netlist: ecuaciones booleanas, elementos C generalizados o mediante puertas de una biblioteca. Incluye herramientas para visualizar STGs y SG (State Graphs). No es apropiado para la síntesis del data-path debido a que no se puede garantizar la obtención de la netlist del circuito. No genera vectores de test.

Se ha empleado para sintetizar varios circuitos de RAPPID [18] y también se ha usado para la síntesis de varios controladores del microprocesador AMULET.

Herramienta 3D.
Universidad de California, San Diego.
http://paradise.ucsd.edu/3D/

Usa un diseño de estilo denominado XBM (extended burst-mode), cubre un amplio espectro de circuitos secuenciales desde circuitos delay-insensitive hasta circuitos síncronos. Puede sintetizar maquinas asíncronas de estados finitos con entradas de cambios múltiples, además de circuitos que caen en el límite entre los circuitos síncronos y asíncronos los cuales son difíciles o imposibles de sintetizar con los métodos existentes. Las entradas son descripciones textuales de controladores XBM y las salidas son ecuaciones lógicas independientes de la tecnología y también pueden ser una netlist incorporada a la tecnología particular.

Esta implementación de máquinas XBM usa lógica combinacional. Está integrado a HFMIN, que permite minimizar lógica a dos niveles, es adaptable a la mayoría de los procesos CMOS comerciales de celdas estándar y bibliotecas de puertas programables por máscaras. Entre las implementaciones se tiene un chip de comunicaciones infrarrojo para transferencia de datos.

2.4.3 Herramientas CAD ad-hoc

Se describen herramientas CAD específicas ya sea por la metodología de diseño o por la orientación de su aplicación relacionada al tema del presente trabajo, se presenta sus principales características y logros significativos. En las referencias [17], [68] se citan otras herramientas CAD adicionales.

Herramienta Fulcrum
Fulcrum Microsystems, USA
www.fulcrummicro.com

Los diseños de esta compañía están basados en una lógica dominó dual-rail y un mecanismo de detección de conclusión entre las puertas lógicas, utiliza una topología de bloques lógicos subsecuentes uno tras otro. La secuencia de operaciones siguen el modelo delay-insensitive usando un protocolo handshaking de 4 fases.

Reportan como ventajas propias de su metodología el evitar glitches y condiciones de estados lógicos indefinidos, también mediante una técnica propia minimizan problemas de metaestabilidad y consiguen una mayor inmunidad al ruido.

En el campo de redes de comunicaciones, anuncian la fabricación ASIC de un crossbar totalmente asíncrono, el cual comparado con diseños síncronos similares tiene la ventaja que el suministro de corriente varía linealmente con la actividad del circuito en lugar de variar con la frecuencia de reloj. El proceso de fabricación es en la tecnología de 130 manómetros y especifican un throughput agregado de 800 Gbits/seg.

Herramienta Theseus NCL.
Theseus Logic, USA.
www.theseus.com

Desarrolla circuitos delay insensitive mediante NCL (Null Convention Logic), usa una biblioteca de puertas denominadas de umbral para implementar diseños multi-rail. Típicamente se usa estructuras dual-rail y mediante las celdas umbral se implementa las funciones lógicas, igualmente los diseños NCL implementan estructuras secuenciales como una máquina de estados finitos. Los diseños delay insensitive son portables de una tecnología a otra, debido a que no usan reloj tienen una mejor característica EMI, permite ahorro de potencia, la propiedad de histéresis de las celdas umbral proporciona inmunidad al ruido, los fallos son inherentemente detectables debido a la naturaleza de la tecnología.

El flujo de síntesis NCL proporciona el soporte de bibliotecas para hacer factible la síntesis de un diseño especificado en VHDL hacia netlist NCL. De esta forma diseñador puede desarrollar un circuito asíncrono usando un HDL comercial conocido, para la verificación se puede usar simuladores comerciales, se tienen restricciones simples para el diseño durante la síntesis. Con una herramienta adicional se puede verificar la naturaleza delay insensitive del diseño, permite usar tanto celdas umbral simples como celdas más complejas.

Las desventajas son que el diseño puede ser 3 a 5 veces mayor en área respecto a un circuito síncrono. También puede disminuir el throughput del diseño, o si se intenta una

transformación directa de un diseño síncrono puede incrementarse el consumo de potencia.

El mejor demostrador es el NCL08, que es la versión asíncrona compatible del microcontrolador de Motorola HC08. También se desarrolla un chip de test que realiza la transformada wavelet y que es similar al diseño de un 8051. La metodología NCL, ha servido para implementar smartcards. Esta tecnología es compatible con herramientas de ATPG, se puede tener el equivalente de un registro scan y de poca área adicional respecto al caso síncrono. Permite la verificación funcional post-síntesis. Mediante un subconjunto de vectores de simulación funcional se puede realizar el test físico juntamente con técnicas BIST para los bloques de memoria.

Herramienta PipeFitter.
Politécnico de Torino.
htpp://linus.polito.it/blunno/pipefitter

Está dirigido a la síntesis automática de circuitos asíncronos micropipeline con unidad de control de 4 fases. Considera un data-path síncrono con matched delays. El lenguaje de especificación es un pequeño número de sentencias Verilog, se deriva el netlist para la unidad de control y otra para cada registro del data-path. Puede generar una especificación STG para Petrify.

Cada módulo del data-path es sintetizado en celdas estándar mediante una herramienta de síntesis RTL como el compilador Synopsys o Cadence o Mentor. Se realiza un análisis de tiempos del data-path para determinar los retardos de peor caso y se inserta un bloque de retardo para cada registro y bloque del data-path, de esta forma se genera las correspondientes señales para la unidad de control.

Permite el diseño de ASICs de baja emisión electromagnética. Para el test del data-path se puede usar scan chains y una aproximación de scan para el controlador.

Herramienta Butler:
MBDA / Matra BAe Dynamics
www.mbda.co.uk

Los diseños genéricos tienen una estructura de diseño modular y es construido como una ensambladura de diversos subdiseños. Cada subdiseño es un bloque de construcción que contiene lógica y estructura, la lógica es expresada mediante un número mínimo de puertas interconectadas.

Los diseños genéricos no emplean reloj y son dirigidos eventos, cada diseño es analizado mediante métodos formales matemáticos. Pueden ser integrados con otros circuitos e implementados en diferentes tecnologías ya que no son dependientes de parámetros de tiempos críticos. El circuito no tiene un consumo de potencia dependiente de la demanda.

Por ejemplo un diseño genérico es el denominado “butler”, puede ser usado con cualquier microprocesador. Administra variables de control asignadas para cada tarea a ejecutarse en el microprocesador y durante los tiempos de ejecución identifica la próxima tarea a ejecutarse. Se pueden asignar niveles de prioridad para tareas

individuales o grupos de tareas. Las tareas dentro de un grupo y de la misma prioridad son seleccionadas en una característica “round robin”.

Otro diseño genérico es el denominado “route-table”, puede ser usado con cualquier medio de comunicación físico. Junto con el diseño genérico “butler” administran la comunicación y los recursos de procesamiento de un sistema.

Herramienta CADP (Caesar/Aldebaran Development Package).
INRIA Rhone-Alpes.
www.inrialpes.fr

Está orientado hacia la eficiente compilación, simulación, verificación formal y test de descripciones escritas en lenguaje LOTOS (ISO norma 8807). Los lenguajes LOTOS y CADP han sido usados para modelar sistemas de comunicaciones. Es apropiado para describir redes de procesos que ejecución en paralelo y comunicación mediante envío de mensajes.

Se ha realizado entre otros, la verificación de un árbitro de bus, la coherencia de protocolos de “cache” para el servidor en arquitecturas de multiprocesadores, la capa de enlace del bus IEEE 1394, el protocolo de árbitro de bus SCSI-2.

Se puede generar, desde las especificaciones, el apropiado test y asegurar la validez de la implementación. Se puede verificar la ejecución de los trazos obtenidos del sistema real y si son aceptadas por la verificación formal.

2.5 Alternativas de implementación

Actualmente la implementación física de un diseño asíncrono tiene que realizarse a través de fundiciones o plataformas concebidas para los clásicos circuitos síncronos. Se tienen varias posibilidades y están ligadas, en mayor o menor grado, a la herramienta CAD utilizada, a continuación se revisan las principales alternativas para la implementación. En presente trabajo se intenta que el diseño sea independiente de la tecnología y esté cerca de las herramientas CAD comerciales.

2.5.1 Tecnologías de fabricación

Asociada a Tangram.

El compilador de silicio propietario Tangram, permite implementaciones mediante biblioteca “standard cells” dedicadas o generales. Por ejemplo permite un formato final de 4 fases, single rail a través del FPGA Maxplus II de Altera. La implementación de las bibliotecas “standard cells” es mantenida mediante la herramienta Build Gates, para la caracterización de tiempos se emplea Pearl, puede enlazarse a las herramientas Cadence y varias tecnologías CMOS.

Entre las implementaciones se tiene: el Procesador Toy fue el primer circuito demostrador. El DCC (Digital Compact Casette) demuestra la ventaja en el bajo consumo de potencia donde la implementación handshake de single rail y celdas

estándar generales, tiene una mejor prestación en área y potencia. El controlador 80c51 permitió implementar una familia de ICs Pager. En telefonía se han implementado microcontroladores basados en el 80c51, un ejemplo es el P83CL882 que es componente de la familia VTELX de bajo consumo y bajo voltaje. Se han implementado smartcards asíncronas de bajo consumo.

Asociada a Balsa.

Balsa puede generar circuitos para tecnologías como FPGA de Xilinx, reglas de diseño genéricas ARM y bibliotecas de celdas como las usadas en la implementación de AMULET3 y también para la biblioteca en tecnología 0.35μ suministrada por Austria Mikro Systems. Como aplicación relevante se ha implementado el controlador DMA para AMULET3i.

2.5.2 Biblioteca de celdas dedicadas

Asociada a NCL.

Theseus Logic Inc, ha desarrollado las herramientas de síntesis denominadas NCL (Null Convention Logic) en la tecnología CMOS de 0.25μ, se puede diseñar circuitos asíncronos NCL de forma similar a diseño tradicional síncrono. Los principales aspectos en desarrollo que reportan es la verificación de los tiempos, identificar las propiedades de las derivaciones isócronas (orphans) y reducir el área adicional de los diseños.

Bibliotecas DCVSL.

Los bloques lógicos construidos mediante DCVSL (Differential Cascode Voltaje Switch) son enlazados mediante circuitos de interconexión que aseguran el handshaking de 4 fases entre las etapas. Los bloques DCVSL pueden ser conectados fácilmente en cascada para realizar circuitos speed-independent. El control del handshaking de 4 fases se realiza mediante la interconexión de circuitos sintetizados con STGs.

En la referencia [19] se ha desarrollado el diseño mediante una arquitectura pipeline constituida por FAM's (Functional Autonomous Module) y utiliza un protocolo de cuatro fases y un data-path dual-rail. La celda básica es un elemento de memorización (memoa), mediante la herramienta Petrify se realiza el diseño de la unidad de control tipo speed-independent. La biblioteca de celdas fue implementada en la tecnología de CMOS de 1μ.

2.6 Entorno de aplicación

La aplicación en un campo específico como son los circuitos de comunicaciones presenta requisitos que pueden ser cumplidos por los circuitos asíncronos. En esta sección se fundamentan las cualidades generales de dichos circuitos, se citan las actuales limitaciones y se establece el diagrama de flujo de la metodología diseño empleada en el presente trabajo.

2.6.1 Ventajas y limitaciones de los circuitos Asíncronos

Consumo de potencia.

El promedio del consumo de potencia en un sistema digital puede ser dividido en 4 componentes: potencia dinámica, potencia de corto circuito, potencia de fugas, potencia estática. En puertas CMOS los componentes de corto circuito y estática son de valores despreciables. Igualmente, la potencia de fugas en las actuales tecnologías comerciales es muy reducida, por lo cual se considera que la potencia dominante es la potencia de dinámica.

En la estimación de potencia se puede considerar el concepto de conmutación de capacidades en el cual se representa el sistema como una caja negra. Esta técnica está limitada por la representación específica de los datos que procesa y su dependencia es solo de las transiciones de la entrada. El consumo de potencia en los circuitos CMOS es debido en gran parte a la actividad de conmutación, en cualquier nodo dicha la potencia dinámica puede ser calculada por la ecuación:

$$P= f (CV^2) /2$$

Donde C es la capacidad del nodo, V es el voltaje de alimentación y f es la frecuencia de conmutación en dicho nodo. A partir de esta ecuación se pueden plantear alternativas para disminuir la potencia dinámica de los circuitos síncronos, a continuación se describen las alternativas y los problemas colaterales encontrados.

Se puede observar que una reducción de la fuente de alimentación tiene un efecto cuadrático, sin embargo, también reduce la velocidad de operación y reduce el margen de ruido. Estos efectos pueden ser severos si el voltaje alimentación se acerca a los voltajes de umbral del transistor MOS. También la corriente estática de fugas se incrementa y es un serio inconveniente para sistemas que usan baterías donde hay grandes tiempos de espera.

En aplicaciones donde la demanda es variable, ajustando el suministro de potencia de acuerdo a procesamiento de los datos puede ser un medio efectivo para minimizar la potencia promedio, pero estos cambios de velocidad de operación implican complejos circuitos de reloj. Si la demanda es fija, la fuente de alimentación debe ser la más baja posible y aún se consiga el requerido funcionamiento.

La reducción global de la capacidad de conmutación dentro del circuito puede lograrse con unas apropiadas dimensiones de los transistores del circuito. En las tecnologías submicrónicas la mayor parte de la capacidad de conmutación proviene de las capacidades parásitas del conexionado, estas capacidades pueden ser minimizadas evitando líneas de conexión largas en el circuito, localizando las transferencias de datos, usando memorias locales y mediante algoritmos de transformación.

La reducción del número de transiciones en cada nodo dentro del circuito puede ser realizada mediante algoritmos de transformación, explotando la correlación entre los datos, eliminando actividad redundante y transferencias de datos localizadas. La actividad redundante puede ser eliminada mediante una apropiada elección de la

representación numérica, desactivando porciones de la longitud total del data-path y evitando la generación y propagación de valores intermedios o glitches.

La señal de reloj que se distribuye a través de todo el sistema, tiende a presentar una alta capacidad. Las técnicas de regular el reloj para bloques inactivos o para desactivar el reloj global para periodos sin actividad pueden ahorrar potencia. El inconveniente es que presenta una complejidad considerable, ya que se requiere identificar los bloques inactivos y que se introduce un retardo para restablecer el generador de reloj antes que las salidas sean estables.

En los circuitos asíncronos no existe una señal de reloj global, por lo cual disminuye la potencia dinámica. La característica local de los bloques en actividad involucradas y el rendimiento tipo "averge-case" por la adaptación a diferentes requerimientos de velocidad de operación permiten un bajo consumo de potencia [20] [21].

Diseño Modular.

Una alternativa frente a los diseños síncronos es el diseño asíncrono, donde las operaciones son administradas mediante handshakes entre circuitos adyacentes. Para sistemas SOC (System-On-Chip), los módulos asíncronos ofrecen la ventaja que dichos módulos están definidos con precisión y facilita su interconexión sin ninguna referencia global de tiempos.

La tendencia hacia la tecnología SOC está siendo promovida por la industria electrónica. El primer objetivo es el diseño bloques IP (Intellectual Property) que puedan ser implementados en corto tiempo y garantizar que funcionarán correctamente desde el primer intento. Por lo cual son necesarios métodos globales de diseño que permitan un ensamblado simple de los bloques IP y basados en principios de "plug-and-play".

Los circuitos asíncronos que no usan señales de reloj, constituyen una atractiva aproximación para la tecnología SOC. Desde el punto de vista de arquitectura es mucho más fácil construir los circuitos mediante bloques asíncronos que mediante bloques síncronos. El desarrollo de IPs asíncronos sería un paso importante para la tecnología SOC.

Interferencia electromagnética.

La compatibilidad electromagnética (EMC) es un parámetro importante en el diseño de los sistemas electrónicos, una razón es la estricta regulación para mantener los ambientes sin interferencia electromagnética (EMI). Por otro lado, los niveles de radiación electromagnética se incrementan conforme el sistema usa mayores frecuencias de reloj. Se han planteado algunas técnicas a fin de minimizar la EMI de los circuitos síncronos, pero en los circuitos de comunicaciones se tienen limitaciones por las altas frecuencias de operación y causar un incremento del jitter y skew.

Una consecuencia del uso de circuitos asíncronos es que producen menor emisión electromagnética que los circuitos síncronos. En un circuito síncrono toda la actividad de datos está sincronizada al flanco de la señal de reloj, lo que causa grandes picos de corriente a la misma frecuencia del reloj y gran número de armónicos de dicha

frecuencia. En un circuito asíncrono toda la actividad está distribuida en el tiempo, de esta manera se genera muy poca radiación de armónicos. Por ejemplo para sistemas "wireless" una baja EMI es una ventaja ya que al no existir un reloj global el nivel de energía y rangos del espectro de radiación no son críticas.

Otros aspectos.

Los circuitos asíncronos tienen ventajas adicionales como una mejor tolerancia a variaciones de parámetros físicos o ambientales y a un nivel de sistema poder tener módulos que permitan una mejor adaptación a variaciones tecnológicas.

Por otro lado, la mayor dificultad que presenta el empleo de los circuitos asíncronos es el incipiente desarrollo de la metodología y la relativa mayor complejidad del diseño que el caso de circuitos síncronos. En un diseño asíncrono se debe prestar atención al estado dinámico del circuito, los hazards deben ser evitados para un correcto funcionamiento. La metodología de diseño asíncrono presenta limitaciones en las transformaciones algebraicas, además de las limitaciones de las herramientas CAD en el layout y síntesis de circuitos asíncronos.

Por lo cual la investigación en esta área requiere una aproximación global desde la metodología inicial hasta la aplicación y encontrar nichos de donde se justifique el empleo de los circuitos asíncronos.

2.6.2 Metodología de diseño

En la Figura 2.6 se muestra el diagrama de flujo de la metodología de diseño desarrollado en el presente trabajo y que se desarrolla en los siguientes capítulos. La metodología de diseño empleada se inicia de un entorno de diseño que permita la generación de los módulos LPMs (módulos parametrizables) y captura del diseño en lenguaje VHDL.

El estilo de diseño asíncrono micropipeline implica definir el data-path y los circuitos de control, en ambos se establecen las apropiadas regiones equi-potenciales y derivaciones isócronas mediante las facilidades de la herramienta de diseño (cliques, LOCs) y que se reflejaran en el place&route del diseño. La regularidad del diseño permite un fácil layout y lograr un mejor rendimiento. Luego se realiza la simulación post-síntesis, que permite verificar la funcionalidad del diseño y verificar la temporización de los circuitos de control respecto de los retardos del data-path.

Finalmente en la implementación se puede optar mediante un ASIC o mediante un dispositivo programable, en el presente trabajo se mantiene un enfoque general pero el prototipo demostrador se orienta a los FPGAs disponibles de Altera y Xilinx.

A parte de las definiciones y conceptos específicos que se desarrollan en los capítulos posteriores, se toma en cuenta consideraciones generales que se reflejan en dicha metodología como las que se detallan a continuación:

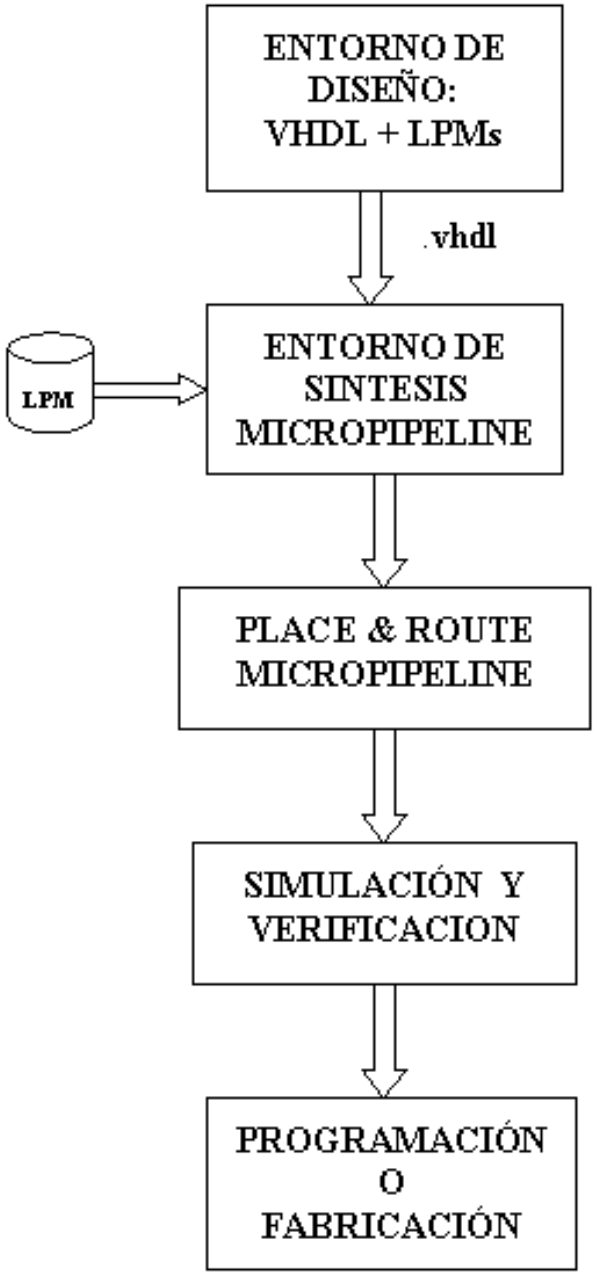

Fig. 2.6: Diagrama de flujo de la metodología de diseño

Flujo de diseño convencional.

La implantación de los circuitos asíncronos, cuya difusión depende de su empleo por parte de los diseñadores, tiene dificultades como requerir una significativa re-educación de los diseñadores y que sus conocimientos están fuertemente ligados a las herramientas comerciales de diseño síncrono clásico. Una aproximación a la solución de dichos problemas es intentar un flujo de diseño de circuitos asíncronos que esté lo más próximo posible a uso de las herramientas comerciales de diseño síncrono.

Por otro lado, el estilo de diseño denominado de retardos acotados (bundled delay), si bien impone restricciones sobre el layout, la ventaja es que consume menor potencia ya que necesita solo la mitad de las líneas de conexión. A diferencia de un estilo totalmente insensible a retardos, como por ejemplo el implementado por Theseus Logic Inc, que tiene la desventaja de usar dos líneas para cada señal.

Tendencia actual.

En forma general, actualmente como los objetivos y ventajas de una metodología de diseño asíncrona se pueden citar:

- Reducido tiempo de puesta en el mercado de productos mediante diseño automático, lenguajes de alto nivel y facilidades de re-uso.
- Ensamblado tipo “plug&play” evitando problemas asociados a la señal de reloj.
- Bajo consumo de potencia y baja EMI como propiedades intrínsecas.
- Estilo de diseño convencional y que permita manejar los retardos de las líneas de conexiones en las nuevas tecnologías como por ejemplo DSM (deep submicron).

CAPÍTULO III

MARCO GENERAL DE LA APLICACIÓN

3.1 Introducción

En este capítulo se detallan las consideraciones específicas a los circuitos de comunicaciones. Se plantea la problemática en comunicaciones y algunos ejemplos de la necesidad de los circuitos asíncronos aplicados a esta área.

Se enfoca el protocolo ATM y concretamente la Unidad de Clasificación del subsistema QoS, se aborda su implementación hardware y se detallan los problemas asociados tanto por la naturaleza de la función y también debido al empleo de los clásicos circuitos síncronos.

Un aspecto esencial es el perfil del tráfico y la caracterización del modelo de tráfico que sirven para el diseño de un conmutador ATM. Se considera un conmutador de memoria compartida y se establecen sus ventajas y limitaciones.

La estrategia de asignación de prioridades es un campo de investigación aparte y se usan resultados de estudios previos de caracterización del modelo de tráfico ATM. Tomando en cuenta todos estos aspectos, se aborda el diseño de la arquitectura asíncrona de Unidad de Clasificación de las células ATM basado en dicha asignación de prioridades.

3.2 Consideraciones de la aplicación en comunicaciones

En la aplicación de los circuitos asíncronos, se pueden distinguir aspectos críticos del diseño de circuitos síncronos convencionales para los cuales los circuitos asíncronos se consideran una alternativa. En la problemática actual por una parte se tiene consideraciones de carácter general, las cuales se han reseñado anteriormente, y por otro lado, se tienen consideraciones específicas a los circuitos de comunicaciones. A continuación se plantea la problemática en comunicaciones y algunos ejemplos de circuitos de aplicación.

3.2.1 Problemática actual

El diseño de circuitos asíncronos es hoy en día tema de investigación tanto en Universidades y Empresas [20], [21], [22], [29]. Esto es debido a las limitaciones que presentan los clásicos circuitos síncronos, y especialmente en el campo de aplicación de

los circuitos digitales de comunicaciones. Se pueden distinguir los siguientes problemas importantes:

A. Relación velocidad y consumo de potencia.

En el campo de las telecomunicaciones para los diferentes sistemas digitales constitutivos, se incrementa los requerimientos de una mayor velocidad de operación y un bajo consumo de potencia.

En el caso específico del diseño de conmutadores en las redes de comunicaciones, desde un nivel de sistema, siempre ha estado basado en la optimización de ciertos parámetros como throughput, la tasa de pérdida de células y el retardo de las células. Pero un aspecto importante es el consumo de potencia y que en ciertas aplicaciones puede ser un factor determinante. Por ejemplo en redes de comunicaciones que incluyan enlaces vía satélite y en equipos portátiles, el principal objetivo es minimizar la disipación de potencia dentro del conmutador.

B. Cualidades de EMC.

El diseño asíncrono permite un control de grano fino de actividad del circuito, sin necesitar complejos circuitos de regulación del reloj y, además, produce un nivel bajo de interferencia electromagnética. Por lo cual presentan inherentes cualidades de EMC para satisfacer diferentes estándares, se puede concluir que en este punto que los circuitos asíncronos ofrecen ventajas definitivas.

C. Problemas de latencia, sincronización, skew y jitter.

Como consecuencia del constante incremento de la relación velocidad/consumo de potencia. Las implementaciones de conmutadores ATM deben resolver de forma óptima los problemas asociados de sincronización y latencia. En una implementación asíncrona de un subsistema QoS-ATM, la sincronización puede ser resuelta fácilmente y a la vez mejorar la latencia, además de minimizar el skew y jitter asociados.

3.2.2 Ejemplos de la necesidad de sistemas asíncronos

Se pueden encontrar informes de algunas implementaciones asíncronas en el campo de los circuitos de comunicaciones y específicamente para circuitos de conmutación, a continuación se revisan algunos de ellos que sean representativos y recientes.

A. DRACO

Es un circuito integrado aplicado a telecomunicaciones, DRACO (DECT Radio Communications Controller) [23], es un controlador de estaciones base del sistema DECT (Digital European Cordless Telephone). Está compuesto de un subsistema de procesamiento asíncrono basado en el microprocesador AMULET3H y un subsistema periférico síncrono de telecomunicaciones. En forma global DRACO se puede considerar como un SoC con la característica que el subsistema de procesamiento es totalmente asíncrono.

B. CADRE (Configurable Asynchronous DSP for Reduced Energy).

Es una aplicación orientada al campo de telefonía móvil donde se demanda un alto rendimiento de las funciones DSP. Considerando que en el futuro es previsible que siga incrementándose por ejemplo el throughput requerido, por lo cual es importante encontrar alternativas de solución a dichas demandas y el consumo de potencia en las baterías que deben ser de larga duración.

Algunos informes, por ejemplo en [24], muestran que el DSP puede consumir alrededor de un 65% de la potencia total. Se estima que en el futuro los teléfonos móviles requerirán un throughput mayor de 100 MIPS de parte del DSP, lo cual significa, en el futuro, un incremento de la potencia.

CADRE [25] es una arquitectura de un DSP de bajo consumo y alto throughput. El estilo de diseño es el denominado de retardos acotados (bounded delay). Combina en el diseño un esquema de reducción de consumo de potencia de niveles múltiples y una arquitectura paralela. Un conjunto de instrucciones reducido le permite alcanzar alto throughput sin un excesivo ancho de banda de memoria del programa. Junto con la representación con signo de los datos permite reducir la actividad del circuito durante su operación.

La importancia de Cadre radica en que es un ejemplo de una aplicación que teniendo un cierto nivel de complejidad usa un estilo de diseño retardos acotados (bounded delay).

C. Clasificador self-timed.

Esta arquitectura [26], es una red de clasificación “self-timed” para conmutadores que soportan diversos tipos de tráfico. La implementación full-custom usa lógica dominó de pre-carga y técnicas para minimizar la latencia. Puede soportar enlaces hasta 10Gb/s para células ATM. La implementación del núcleo ha sido realizada en un estilo de diseño “full-custom” en HP CMOS de 0.35 micras, 4 capas de metal. Este trabajo pionero importante intenta resolver los aspectos claves en las redes de conmutación de células: alto throughput, bajo tiempo de latencia, clasificación de acuerdo a prioridades y arquitectura ampliable en cuanto a su capacidad de almacenar células.

Una desventaja de este diseño es el empleo de una lógica específica tipo dominó de pre-carga, ello hace que no esté cerca del uso de herramientas de diseño comerciales con todos los inconvenientes que ello implica.

D. Otros ejemplos.

En el campo de comunicaciones Fulcrum [22] anuncia un crossbar asíncrono descrito anteriormente. Existen otros desarrollos específicos de diseños asíncronos que se pueden considerar relacionados a esta área como:

- En INRIA Rhone-Alpes [27], conjuntamente los lenguajes LOTOS y la herramienta CADP han sido usados para modelar sistemas de comunicaciones, por ejemplo se ha realizado la verificación del árbitro de bus SCSI-2 y en la capa de enlace del bus IEEE 1394.

- En Philips Research Laboratories [28], mediante el compilador de silicio Tangram, se ha implementado el DCC (Digital Compact Casette) es una implementación handshake de single-rail mediante "standard cells" generales y que muestra una mejor prestación en área y potencia.

- En Theseus Logic [29], se han desarrollado circuitos "delay insensitive" mediante NCL (Null Convention Logic), por ejemplo el NCL08, que es la versión asíncrona compatible del microcontrolador de Motorola HC08. También informan el desarrollo un chip de test que realiza la transformada "wavelet".

3.3 El protocolo Modo de Transferencia Asíncrona (ATM)

Actualmente el protocolo ATM es una norma de comunicaciones madura y de pleno uso, subsistemas como la QoS en su implementación hardware, tiene problemas asociados tanto por la naturaleza de la función y también debido al empleo de los circuitos síncronos. A continuación se revisa los conceptos importantes de la arquitectura ATM y se detallan los puntos problema de la QoS.

3.3.1 Concepto del protocolo ATM

El protocolo ATM es considerado como un modo de transferencia específicamente orientado a paquetes basados en células de longitud fija. Cada célula consiste de un campo de información y una cabecera el cual es usando principalmente para determinar el canal virtual y realizar el apropiado encaminamiento. La integridad de la secuencia de células es preservada por el canal virtual.

En la Figura 3.1 se muestra la arquitectura de capas ATM. La capa física realiza las funciones al nivel de señal. Luego se tiene la capa ATM que gestiona la transferencia del paquete y es común a todos los servicios, esta es la capa que supervisa la calidad de servicio QoS (Quality of Services).

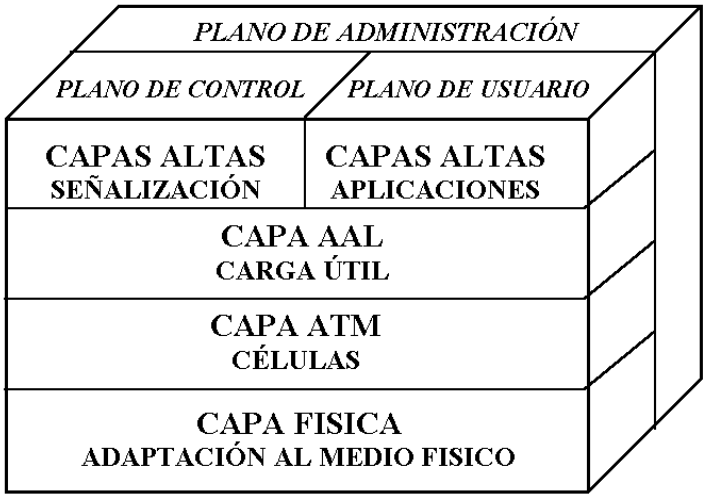

Fig. 3.1: Arquitectura del protocolo ATM

La capa AAL (ATM Adaptation Layer) tiene como función, entre otras, la gestión de las variaciones de retardo del paquete. Las capas más altas están relacionadas con las aplicaciones y administración.

La Figura 3.2 muestra la estructura de la célula ATM. Se tienen 44 ó 48 bytes de información (payload) dependiendo si se tienen los 4 bytes de la capa de adaptación ATM. En el presente trabajo se considera una longitud total de 53 bytes para la célula ATM, 48 bytes de información y 5 bytes de cabecera.

Las normas de ATM establecen, en la cabecera de la celda, un bit CLP de prioridad de pérdida de célula. Permite distinguir, para cada circuito virtual, que células tienen prioridad, una alta prioridad es indicada por el valor cero, para una baja prioridad el bit es puesto a uno.

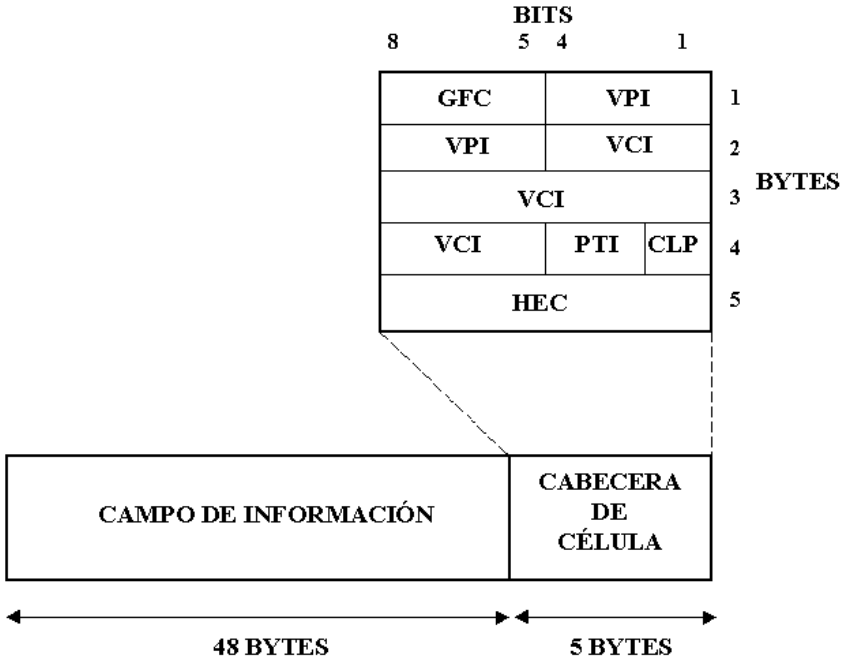

Fig. 3.2: Estructura de la célula ATM

Los otros campos como el de trayecto virtual (VPI) y el identificador de canal virtual (VCI), sirven para identificar el circuito virtual correspondiente. El conmutador ATM puede optar por conmutar a partir del identificador completo VPI y VCI o solo a partir del VPI.

El identificador de tipo de carga (PTI) sirve diferenciar entre aquellas células que transportan información de la capa superior y aquellas que cumplen una función determinada en la capa ATM. El campo de errores en cabecera HEC sirve para proteger contra errores solamente a los bits de la cabecera, permite corregir errores de 1 bit o detectar errores de varios bits.

En el anexo A se amplía algunos aspectos importantes del protocolo ATM que son utilizadas en el presente trabajo.

3.3.2 Definición del problema QoS-ATM

En el presente trabajo la QoS-ATM se basa en las normas de calidad de servicio QoS para las redes ATM, las cuales se define en "Traffic Management Specification" de ATM forum [30]. Se tienen diferentes categorías de servicio dependiendo de las diferentes aplicaciones y tipos de tráfico que a continuación se describen:

CBR (constant bit rate): Denominado Clase A, sirve para aplicaciones en tiempo real como voz y vídeo que requieren valores determinados de retardo y variación de retardo.

VBR (variable bit rate): Sirve para aplicaciones de vídeo comprimido o vídeo conferencia, presentan un comportamiento tipo burst. Se divide en servicios en tiempo real (VBR-rt) denominado Clase B, y servicios de tiempo no real (VBR-nrt) denominado Clase C.

UBR (Unspecified bit rate): Se considera dentro de la Clase D, diseñado para aplicaciones tolerantes a retardos y aplicaciones en tiempo no real.

ABR (available bit rate): Se considera dentro de la Clase D, diseñado para aplicaciones que tienen la habilidad de ajustar su tasa de transferencia de acuerdo a una capacidad reservada en la red.

GFR (guaranteed frame rate): Este servicio es intermedio entre ABR y UBR en cuanto a prioridad y proporciona QoS al nivel de tramas.

Se puede considerar el problema de QoS-ATM en dos ámbitos: el primero al nivel de la función de QoS en el protocolo de comunicaciones ATM, el segundo involucra los problemas que se tienen en la implementación de la QoS y de los circuitos síncronos clásicos que soportan este protocolo. A continuación se describen dichos ámbitos.

A. Problema asociado a la QoS en el protocolo ATM.

En la bibliografía existente [35] [36] [71] [72] se ha realizado estudios sobre la QoS-ATM, los factores que intervienen y su problemática. A continuación se resumen considerando dos principales factores: tráfico y eficiencia.

- Tráfico Aleatorio.

El problema radica en que debido a la naturaleza aleatoria del tráfico de banda ancha como por ejemplo: transferencia de archivos de datos y comunicaciones de vídeo de bit-rate variable, es difícil un control efectivo del tráfico del usuario a fin de impedir la congestión de la red o al menos ocurra muy rara vez. Las células son enviadas a una simple salida desde varias fuentes, se presenta una competencia y las células deben ser almacenadas en un buffer o descartadas.

A diferencia de un tráfico uniforme que es predecible y relativamente fácil de manejar, se tiene otros tipos de tráfico cuyas características hacen que sea difícil su administración. Por ejemplo las fuentes de tráfico tipo burst, donde las células arriban en grandes lotes seguidos de periodos vacíos, presentan un bajo tasa promedio y un alto

tasa de pico. El conmutador debe almacenar algunos lotes para una transmitirlos posteriormente y además de asegurar sus tiempos de transmisión.

En ATM Forum recomienda normas para la función de Control de Admisión del Protocolo ATM, pero se debe decidir si la solicitud de conexión de un nuevo canal virtual, con sus requerimientos de QoS, debe ser establecida en la red o caso contrario ser rechazada. Esta decisión se basa en el conocimiento del estado general de la red, es decir, la carga del tráfico y la disponibilidad de los recursos.

- Condiciones de QoS y eficiencia.

Las comunicaciones en tiempo real pueden ser clasificadas en dos categorías de acuerdo a las condiciones de QoS: determinísticos y estadísticos. En el primero, las condiciones de QoS son especificadas en términos absolutos y no se permite infracciones en el tiempo de retardo o en la pérdida de células. Para satisfacer estos requerimientos absolutos cada conexión debe reservar recursos basados en el comportamiento del tráfico para el peor caso, lo cual resulta en una severa infra-utilización de los recursos de la red, especialmente cuando la fuente del tráfico es tipo burst.

Por contrario, en el caso estadístico, se especifica las condiciones de QoS desde el punto de vista de probabilidades, es decir, que se tolera un porcentaje de pérdida de células y/o infracciones en el tiempo de retardo, esto permitirá una total ocupación de los recursos de la red al mismo tiempo que aumenta la eficiencia de los circuitos asociados.

Los actuales sistemas ATM proporcionan para las diferentes clases de servicio varios niveles de QoS, las células de entrada son usualmente clasificadas en las correspondientes colas en base de sus diferentes requerimientos de tasas de retardo y pérdida de células. Los amplios niveles de QoS son proporcionados por los sistemas ATM para satisfacer las necesidades de las aplicaciones. Esto permite compartir el ancho de banda de la red pero implica tener un subsistema de atención de QoS de propósito general, flexible, eficiente y que permita todas las estrategias de QoS necesarias.

Por lo anterior se puede inferir que para proporcionar un servicio de comunicaciones de tiempo real estadístico, y mejorar la eficiencia, se requiere conocer el modelo de la fuente de tráfico, mecanismos de reservación de recursos, realizar un apropiado esquema de conmutar células y administración de buffer.

B. Problemas asociados a la implementación.

A continuación se reseñan los principales problemas que han sido reportados en implementaciones reales mediante clásicos circuitos síncronos:

- Throughputh.

Con el advenimiento de redes de Gbits con QoS mejoradas de alto throughput, en la conmutación de alto rendimiento la tendencia es implementar la mayor parte de los sub-componentes en hardware. La actual tecnología proporciona la necesaria capacidad para implementar conmutadores avanzados en un simple chip, evitando el alto coste y limitaciones de organizaciones multi-chip.

Uno de los problemas radica en que usualmente las colas se administran por flujo y la tendencia a incrementar el número de puertos I/O. Para la administración de dichos flujos tradicionalmente se ha usado DRAMs o SRAM para los buffers y técnicas de pipeline donde se mantienen punteros hacia las DRAM o SRAM. En las referencias [31][33], se tienen ejemplos de este tipo de implementaciones.

Actualmente se tienen memorias de altas prestaciones como DDR SDRAM (double data rate synchronous dynamic random access memory), pero las limitaciones radican, aparte de la complejidad de su administración, en sus características técnicas. Las memorias tipo SRAM proporcionan alto throughput pero una capacidad limitada, las memorias tipo DRAM ofrecen similar throughput y alta capacidad pero en ambas el coste en área, consumo de potencia pueden ser factores en contra.

Por todo lo cual es deseable la implementación hardware del mayor número de subsistemas, ello disminuye las altas exigencias de requerimientos en las memorias o limita su uso en los subsistemas donde sean imprescindibles.

- Jitter.

El problema del "jitter" se puede considerar de dos tipos, el primer tipo uno al nivel de variaciones de la señal de reloj y el segundo tipo debido a los diferentes tiempos de servicio de los algoritmos de QoS.

Jitter de la señal de reloj.

En un sistema síncrono el "jitter" de la señal de reloj se define como la desviación producida de los tiempos ideales en los flancos de transición del reloj. Es una consecuencia debido a que en los circuitos de transmisión y recepción de alta velocidad, las señales conmutan cada vez a mayores velocidades y necesitan restricciones en los tiempos de transiciones.

Los flancos de señal de reloj están restringidos a un estrecho margen de tiempo para capturar los datos correctamente. La desviación de dicho margen afecta severamente a la transferencia de datos de alta velocidad por lo cual debe ser eliminada o minimizada, por ejemplo el parámetro BER (Bit Error Rate) del sistema está directamente afectado por cuan a menudo la señal de reloj no cumple con dicho margen aceptable.

Aunque existen mecanismos para reducir el jitter como PLLs (phase-locked loop) o DLLs (delay-locked loop), siempre se tiene un jitter intrínseco producido por los mismos mecanismos. También es inevitable un jitter aleatorio debido a la desviación imprevisible de la señal de reloj y que añade el jitter intrínseco.

Jitter en el tiempo de servicio.

La tendencia en la realización de la QoS es que debe permitir satisfacer la demanda de garantías que por ejemplo para conexiones con mínimo throughput y máximo retardo puedan pasar instantáneamente a disponer de todo el throughput posible para tráfico tipo burst y una total utilización de la capacidad restante para el tráfico de baja prioridad.

En un caso ideal cada circuito virtual (i) debe ser atendido cada un valor promedio(i) de tiempos de célula. Sin embargo, este valor promedio(i) no siempre es un número entero y desde cada algoritmo puede dar diferentes prioridades a las diferentes clases, el tiempo actual entre sucesivas atenciones para un VC (virtual circuit) determinado será algunas veces más corto o más largo que el valor de promedio(i).

El concepto de jitter está asociado a los algoritmos de servicio de la QoS, en la referencia [32] se comparan los algoritmos usuales round-robin, virtual clock, highest-priority-first. Dichos algoritmos difieren entre ellos respecto al jitter que introducen en los intervalos de tiempo del servicio periódico. Se muestra que comparando el retardo para cada clase de servicio, el algoritmo highest-priority-first, basándose en prioridades, es siempre el mejor algoritmo y tiene un retardo de menos de 1%. Otros algoritmos tienen un retraso de hasta 20%.

En este sentido, como se describe en las secciones respectivas, el presente trabajo se basa en un esquema de QoS en la Unidad de Clasificación basado en prioridades. Un circuito asíncrono puede reducir las diferencias entre los intervalos de servicio ya que su operación no está marcada obligatoriamente por los flancos de una señal de reloj. También al nivel de sistema, un circuito asíncrono se adapta fácilmente a las condiciones de operación en un instante determinado.

- Sincronización.

Un aspecto importante es que la célula ATM no puede llevar ninguna información de referencia de tiempos (timing). Por otro lado, debido a la alta velocidad de operación y que los tasas de datos de cada terminal de entrada son independientes, los canales de comunicación trabajan de manera asíncrona.

Para una operación asíncrona significa que el conmutador ATM no necesita un reloj central. El usuario puede elegir libremente el bit rate del puerto de entrada y del puerto de salida. Se necesita comunicaciones asíncronas (handshaking) entre cada puerto de entrada y el puerto de salida. Esto requiere en el caso de una transmisión paralela de bytes, una línea de reloj por cada canal.

Otra posibilidad es codificar tanto datos como el reloj dentro de la secuencia de bits, luego en el bloque de conmutación se extrae la señal de reloj. La conmutación de varias células que arriban de manera asíncrona y que se dirigen hacia un puerto de salida común hace necesario que implementen funciones de adaptación [34].

Numerosos conmutadores de ATM están construidos alrededor de un componente síncrono de bit-rate fijo. De esta manera el esfuerzo de sincronización se enfoca hacia los circuitos en los terminales de los puertos de la red. Cada bloque de conmutación debe ser programable, los buses comunes tienen limitaciones para altas frecuencias de operación. Por lo cual se prefiere tener los componentes en cascada, se debe tener en cuenta la distribución de las líneas de reloj y la regeneración del reloj en los bloques en cascada, especialmente en el caso que el conmutador se extienda a lo largo de varias tarjetas.

Un sistema asíncrono es apropiado para la interfaz con señales del mundo real ya que puede esperar hasta que se resuelva la metaestabilidad o en todo caso presenta mayor

seguridad y eficiencia que un circuito síncrono. En el capítulo V, se amplía esta consideración.

- Alineación de fase del reloj.

Otro problema es la alineación de fase, para permitir una operación asíncrona de los datos respecto al reloj, el conmutador necesita circuitos internos de alineamiento de fase. Cada circuito normalmente tiene varios bloques por ejemplo [33] se tiene: circuito de reconocimiento de fase, circuito de adaptación de fase y circuito de control.

Para la alineación de fase es necesario generar varias señales de reloj para la correcta operación. Cada una de ellas es separada de las adyacentes en una diferencia de fase que depende de las diferentes condiciones de operación.

La dificultad radica en que los flancos de los datos pueden tener una posición arbitraria de fase con relación a los flancos del reloj central, además, que esta posición arbitraria puede cambiar durante la operación en un rango respecto ciclo de tiempo del reloj central.

Un sistema asíncrono elimina la señal de reloj por lo cual, este problema es evitado.

- Skew

Un problema adicional es el cambio de fase (skew) que es causado por las capacidades parásitas y la resistencia del conexionado [34]. El skew respecto del reloj a los datos es un problema aún sin resolver, como una consecuencia el incremento de las tasas de transferencias de datos impone severas restricciones en sistemas con fuente de reloj síncrono, ello limita la frecuencia máxima de operación.

Igualmente el skew puede ocasionar variaciones en la temporización durante la operación del sistema, también mal funcionamiento debido a problemas por las diferencias del voltaje umbral y retardo en los buffers de salida, por las diferencias de los tiempos de transición entre el reloj y los datos de salida o las diferencias de longitud de las líneas de datos y de reloj.

El transporte en paralelo de las líneas de reloj hacia los diferentes chips, puede resolver el problema de cambio de fase. Sin embargo, si el reloj proviene de un reloj interno el cual a su vez proviene de una señal de reloj que entra al chip, luego de atravesar algunos chips se degrada el ciclo de servicio, los flancos y los niveles de la señal de reloj. Esta degradación de señal de reloj puede ser muy severa luego de pasar a través los diferentes chips. La degradación del reloj proviene el comportamiento asimétrico de los buffers que amplifican los flancos de subida y bajada de manera distinta.

El método de emparejar los trazos tiene desventajas como no soportar una topología flexible y que la señal de reloj debe ser transmitida desde cada dispositivo emisor. Por otro lado, se debe emparejar los trazos de forma precisa entre la señal de reloj y cada canal de datos y no se tiene tolerancia a variaciones de temperatura o condiciones ambientales, las cuales pueden cambiar el skew entre los trazos de las señales.

Otros mecanismos como CDS (clock-data synchronization) [39], permiten sincronizar el reloj de entrada y los datos, mediante muestreo de un patrón en el canal de datos se alinea el reloj de entrada con los datos. Usualmente proporcionan varias señales de reloj de diferentes fases para capturar los datos en cada canal de datos.

El mecanismo DCM (digital clock manager) [40], alinea la fase del reloj con los datos mediante un cambio de fase de la señal de reloj de entrada. Se genera dos señales de reloj desfasadas 90 grados, en el muestreo de los datos se usa ambos flancos de ambos relojes y una esas 4 muestras es seleccionada para ser dato válido de entrada.

Los anteriores mecanismos son circuitos propios del fabricante e implican área y potencia adicional, la velocidad del sistema está limitada por la rapidez de operación de dichos mecanismos, adicionalmente mantienen restricciones en el porcentaje de skew, en la longitud de los trazos entre los canales de datos o en la señal de reloj.

Un sistema asíncrono elimina la señal de reloj por lo cual, este problema es evitado y la construcción modular permite la interconexión en cascada sin pérdida de eficiencia.

3.4 El algoritmo de Calidad de Servicio (QoS)

El protocolo ATM tiene la ventaja de ofrecer garantía de QoS, la implementación hardware del subsistema QoS debe tomar en cuenta el perfil del tráfico, la caracterización del modelo de tráfico ATM ha sido bien estudiada y los resultados sirven para un diseño óptimo. La estrategia de asignación de prioridades es otro campo de investigación en esta problemática, en este capítulo se aborda la clasificación de las células ATM basado en dichas prioridades asignadas. El conmutador engloba todas estas funciones y su operación depende de su arquitectura. A continuación se detallan todos estos aspectos.

3.4.1 Concepto de QoS en ATM

Con el incremento de la competencia en el mercado de las telecomunicaciones, se incrementa la demanda de los operadores de redes de compañías de los denominados acuerdos de nivel de servicio (SLAs). De esta forma pueden obtener garantías de la calidad de servicio (QoS) de la red que utilizan. Al mismo tiempo los suministradores de servicio compiten por ofrecer servicios diferenciados con distintos niveles de QoS.

ATM ofrece servicios con garantía de throughput y retardo determinados, ATM soporta voz, vídeo y también tráfico IP (Internet Protocol), por lo cual es cada vez más frecuente los SLAs. Las características de tráfico son definidas por los Parámetros de Tráfico y estos describen el perfil del tráfico de la fuente. El usuario acepta generar tráfico dentro de un marco definido de características (conformat) y la red debe transportar dicho tráfico respetando la QoS especificada.

Los requerimientos de QoS son definidos por parámetros como la tasa de células perdidas y para aplicaciones sensibles de retardos otros parámetros como el retardo de células y variación de retardo de las células. Los parámetros de QoS describen las garantías ofrecidas por la red al tráfico conformado por usuario.

En la Figura 3.3 se muestra el esquema general del algoritmo QoS-ATM. En la arquitectura del algoritmo QoS-ATM, la Unidad de Clasificación es el bloque esencial en el subsistema QoS. Los demás bloques se pueden considerar no críticos y pueden existir varias alternativas en su implementación dependiendo que especificaciones concretas del subsistema QoS. El circuito Memoria Común de Bit-Slices QoS puede ser implementado mediante SRAM, el circuito Procesador de Prioridades puede estar basado en un simple restador aritmético y otros circuitos periféricos son de fácil implementación.

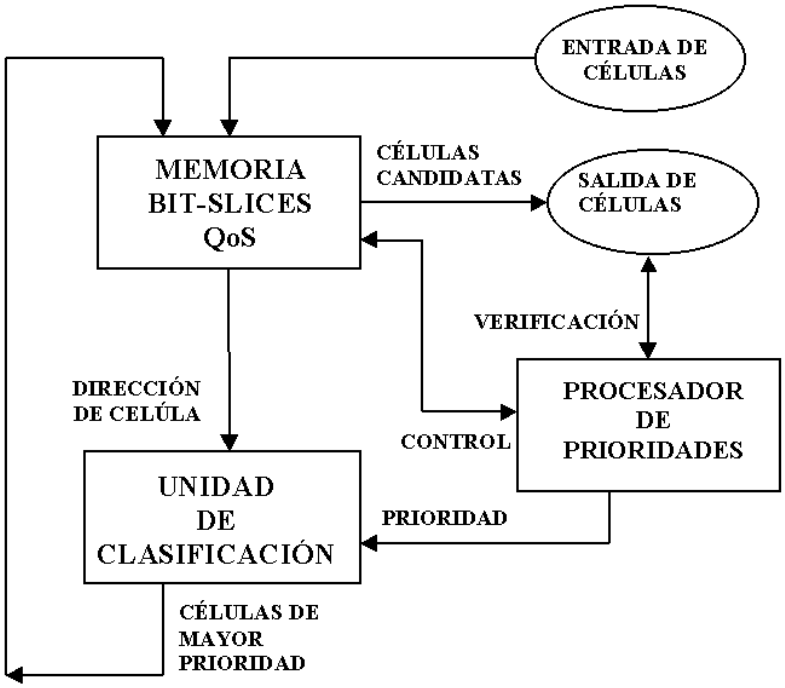

Fig. 3.3: Esquema del algoritmo QoS-ATM

Del esquema anterior se deduce que se tienen dos aspectos principales en la QoS-ATM, uno es la asignación de prioridades a las células de información, el otro es la clasificación de las células de acuerdo a sus prioridades mediante la Unidad de Clasificación, esto permite optimizar el uso de los recursos de la red.

El problema de asignación de prioridades es actualmente materia investigación [35] [36], el presente trabajo se enfoca al segundo aspecto, es decir, la clasificación de las células de acuerdo sus prioridades asignadas.

En el anexo A se muestra los parámetros de tráfico y parámetros QoS para cada categoría de servicio y se detalla las especificaciones de las Categorías de Servicio ATM.

3.4.2 El Conmutador de memoria compartida

Los conmutadores de células ATM de manera general, pueden ser clasificados en 3 categorías principales:

Tipo de memoria compartida (shared-memory).
Tipo de recurso compartido (shared-medium).
Tipo división por espacio (space-division).

En la Tabla 3.1 se muestra las principales características de dichos tipos de conmutadores. En el presente trabajo se concentra en el conmutador de memoria compartida.

PUERTOS DE ENTRADA: N PUERTOS DE SALIDA: N TASA POR PUERTO: V CEL/SEG	MEMORIA COMPARTIDA	MEDIO COMPARTIDO	DIVISIÓN POR ESPACIO
TASA DE OPERACIÓN	2NV	NV	V
MANEJO DE PRIORIDADES	SIMPLE	COMPLEJO	COMPLEJO
RENDIMIENTO DE MEMORIA	ALTO	BAJO	BAJO
TIPO DE CONTROL	CENTRALIZADO	CENTR./DISTR.	DISTRIBUIDO
MANEJO DE MULTICASTING	SIMPLE	SIMPLE	COMPLEJO
COMPLEJIDAD CONTROLADOR	ALTA	MEDIA	BAJA
FACTOR DE LIMITACIÓN	MEMORIA	BUS	ÁREA

Tabla 3.1: Características de los tipos de conmutadores

En la Figura 3.4 se muestra el diagrama de bloques de un conmutador de memoria compartida. El presente trabajo considera un conmutador de memoria compartida MxN y cuando M=N, aunque los resultados de la implementación realizada podrían ser ampliados a los otros tipos de conmutadores.

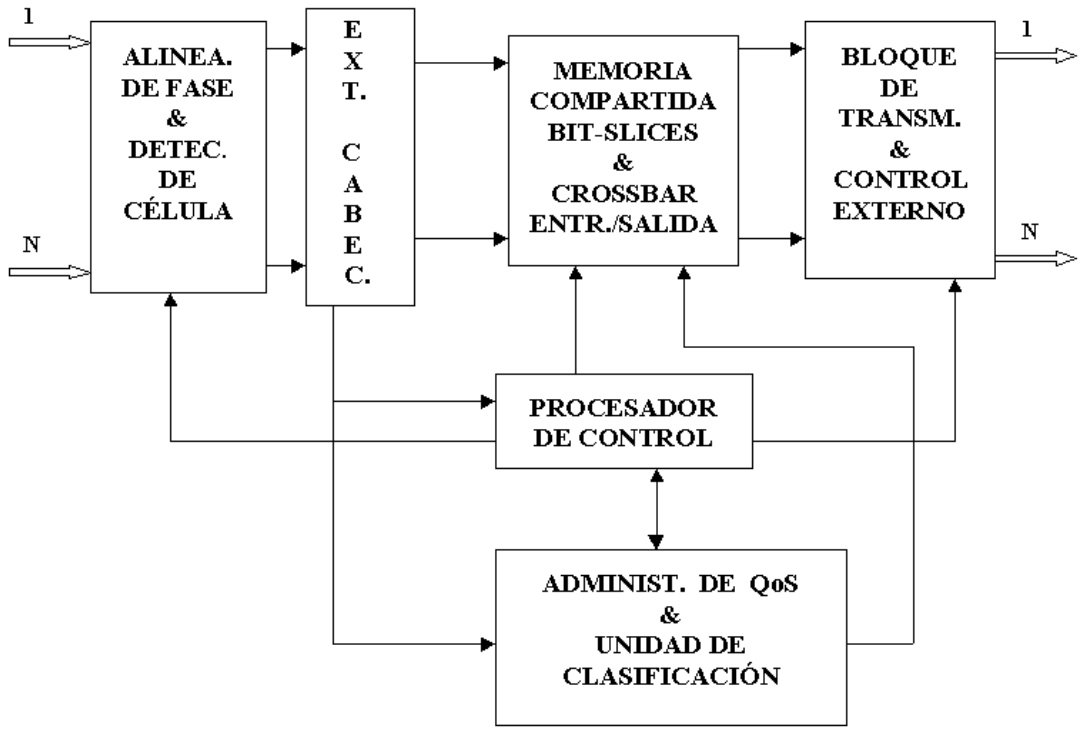

Fig. 3.4: Conmutador de memoria compartida

En este tipo de conmutador, la memoria es compartida por todas las líneas de entrada y salida. Las células que llegan a las N entradas son multiplexadas y almacenadas en la memoria común y luego son organizadas en colas independientes.

Esta arquitectura tiene una limitación por la capacidad finita de la memoria y debido a que no hay coordinación entre las células de llegada en las colas de entrada y sus destinos, algunas células no podrán ser aceptadas por el conmutador y deberán ser descartadas.

Para una CLP (cell loss probability) concreta, esta arquitectura tiene la ventaja de requerir un reducido tamaño de buffer por cada puerto de salida debido a que se comparte los recursos entre todos los puertos. La máxima probabilidad de células perdidas es una restricción en la dimensión de la memoria, junto a cómo la memoria es compartida por las colas de salida, el tamaño del conmutador, el patrón y carga de tráfico.

El principal problema de esta arquitectura es la frecuencia de acceso a la memoria compartida, la solución es que el conmutador sea implementado en bloques de slices en paralelo para soportar mayor throughput y proporcionar tolerancia a fallos. Es decir, que en el peor caso las salidas acezan a un solo bloque de manera simultánea mientras que las entradas pueden ser controladas para almacenar en bloques de memoria desocupados. De esta forma la frecuencia de acceso de la memoria es independiente del número de entradas.

Los avances en la tecnología de las memorias ofrecen altas frecuencias de acceso y mayores anchos de palabra. Por ejemplo la memoria SRAM IBM 041811TLAB-7 tiene un tiempo de ciclo de 7.0ns. Por lo cual se puede considerar resuelto el problema de acceso de memoria en este tipo de arquitectura.

En el anexo A, se establecen las formulaciones necesarias para el análisis de este tipo de conmutador y se amplían algunos conceptos adicionales.

3.4.3 Diseño del subsistema QoS-ATM

En la Figura 3.5 se muestra el diagrama de bloques de la Unidad de Clasificación QoS, se muestra un esquema funcional a fin de facilitar la descripción de su operación. Un concepto importante es que la Unidad de Clasificación es el subsistema principal de la QoS y tiene la función de evitar la congestión dentro del conmutador y utilizar los recursos de manera eficiente para las clases de tráfico con diferentes requerimientos de QoS.

En el presente trabajo se considera una Unidad de Clasificación que opera de forma asíncrona. Es una modificación sustancial del conmutador denominado tipo cilíndrico [70], cuya característica es que todas las entradas están conectadas al mecanismo de selección de células. La arquitectura asíncrona propuesta permite una implementación sencilla de ambos mecanismos de prioridad de espacio y prioridad de tiempo. Lo cual significa que la célula de más alta prioridad es atendida primero hasta que exista al menos una célula en el buffer. Cuando el buffer esta lleno y arriban nuevas células estas

pueden ser admitidas solo en el caso que se descarten las células del buffer que tengan asignadas las menores prioridades, sino dichas nuevas células se pierden.

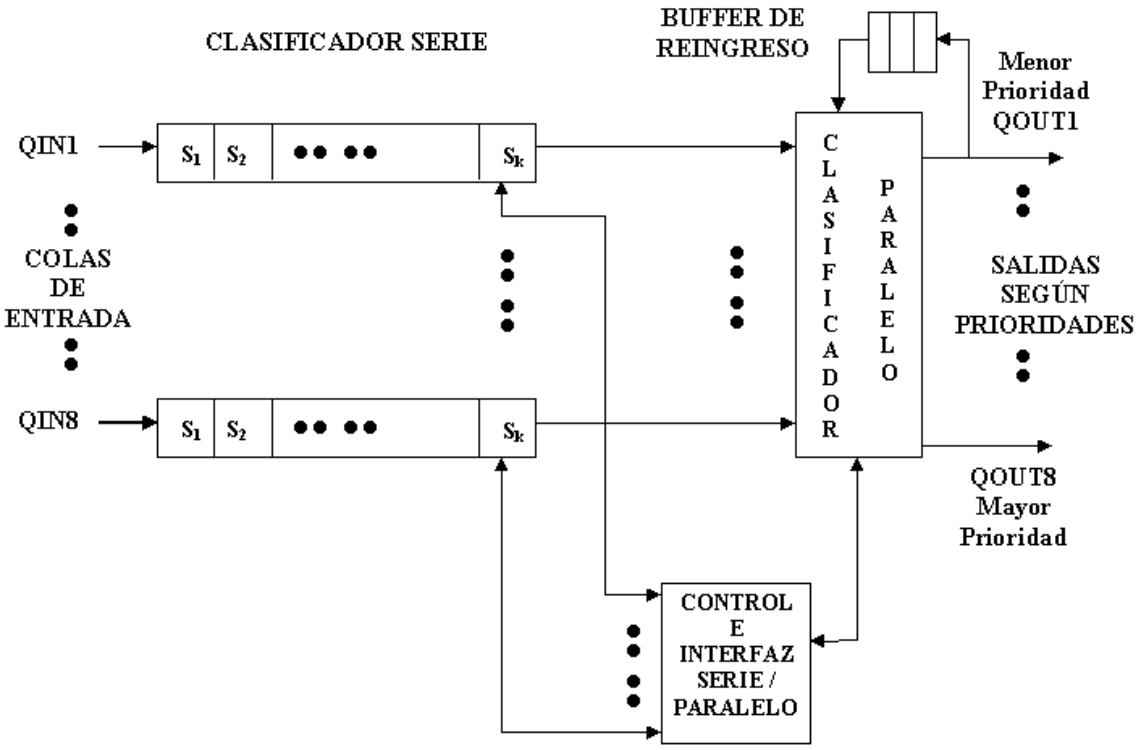

Fig. 3.5: Diagrama de bloques de la Unidad de Clasificación QoS

La Unidad de Clasificación esta basada en dos arquitecturas de circuitos de clasificación una es la arquitectura serie y otra es la arquitectura paralela. Tiene una capacidad de procesar células cuyo número es el mismo que el número de células que la memoria compartida puede almacenar. Si la memoria es expandida, la Unidad de Clasificación también puede ser expandida poniendo más bloques en cascada del circuito de clasificación serie. En el capítulo V se detallan dichas arquitecturas.

La Unidad de Clasificación opera de siguiente manera:

- A cada célula que arriba se asigna un valor de prioridad que se deriva de la cabecera de dicha célula y el algoritmo de gestión de QoS, en general dicha prioridad lleva información de clase de servicio, el VC (virtual circuit) que conforma y el puerto de destino. Junto al campo de prioridad se adjunta un campo de dirección de memoria donde se almacena la célula. Este vector formado por el valor de prioridad y la dirección, es transferido al circuito de clasificación serie respectivo, se tiene 8 colas de clasificación para cada puerto de ingreso.

- En cada cola serie, los vectores son clasificados conforme ingresan y a la salida se tienen los vectores de mayor prioridad. La competencia de acceso de los puertos de salida se resuelve siempre por prioridad, por lo cual dichos vectores de salida de los circuitos serie ingresan al clasificador paralelo. A la salida del

circuito paralelo se tiene los 8 vectores clasificados según sus prioridades y permite que las células puedan ser despachadas en cada ranura de tiempo.

- Se asume que se tiene un buffer por cada puerto de salida donde se guardan los vectores de prioridad para continuar del despacho de las células en las ranuras subsiguientes. El esquema de prioridades permite minimizar el posible bloqueo de los puertos de salida, también permite descartar células de forma sencilla removiendo de una lista las celdas de menor prioridad.

- Dependiendo del algoritmo de gestión de QoS por ejemplo para tráfico multicast, es decir, que una célula tenga como destino varios puertos, se puede tener un buffer de reingreso hacia el clasificador paralelo y competir en las subsiguientes ranuras para transmitir a todos los puertos requeridos.

Este método de clasificación es capaz de implementar una gran variedad de esquemas de prioridad y el tiempo de clasificación es independiente del estado de la cola. Esto permite soportar tráficos elevados con la actual tecnología VLSI. En anexo A, se amplían conceptos sobre las funciones típicas relativas a una Unidad de Clasificación.

3.4.4 Especificaciones técnicas de la aplicación

En la Tabla 3.2 se muestra las principales especificaciones del subsistema de QoS-ATM y concretamente de la Unidad de Clasificación. Se considera un diseño para la norma OC-48 y otro diseño similar para norma OC-12 los cuales a continuación detallan.

Principales Parámetros de diseño	Norma OC-48	Norma OC-12
Utilización	$\rho = 0.6$	$\rho = 0.77$
Perdida de células	$CLP < 10^{-8}$	$CLP < 10^{-8}$
Células por cada cola	20	36
Retardo total teórico	440.31ns	1892.84ns
Retardo total real	446.5ns	1282.5ns
Número de puertos I/O	N= 8	N= 8
Niveles de Prioridad	2^8	2^{16}
Capacidad de células	160	288

Tabla 3.2: Especificaciones para cada norma del conmutador ATM y para la Unidad de Clasificación

Existen muchos estudios sobre de modelo tráfico ATM como por ejemplo [37][71] [72], por lo cual se puede considerar que la caracterización del tráfico ATM es bien conocida y que se pueden usar dichos resultados para estimar los principales parámetros del subsistema QoS-ATM.

De forma resumida se puede considerar que la dimensión del buffer depende de la pérdida de celdas, la carga ofrecida y el tipo de tráfico. Otro factor para la dimensión del buffer está asociado a la QoS y depende de que forma se intente aceptar el tráfico para

la red y el mecanismo de control CAC (connection admission control). En el anexo A se amplían estos conceptos, pero a fin de ilustrar el procedimiento de diseño se muestra dos gráficos principales.

La Figura 3.6 muestra la variación de la CLP con la capacidad del buffer para un tipo de tráfico CBR y una carga dada. Por ejemplo para una CLP= 10^{-8} y una carga ρ= 0.6, el buffer debe tener una capacidad de 20 células (OC-48). Si se considera una carga es ρ= 0.77 y la misma CLP, el buffer debe ser de 36 células (OC-12).

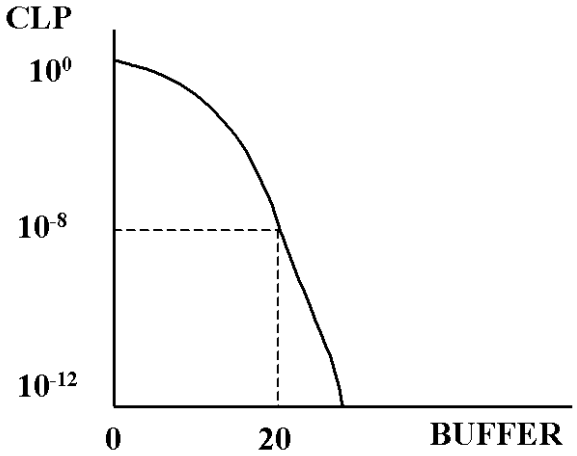

Fig. 3.6: Capacidad del buffer y CLP

La Figura 3.7 representa el retardo promedio en función de la carga total, muestra que el retardo se incrementa sustancialmente a partir un 80% de carga. De las figuras mostradas para tráfico CBR, como punto de operación y de manera práctica se puede estimar que el promedio del retardo total debe ser aproximadamente el doble del tiempo de célula y la utilización en cualquier cola se fija alrededor del 60%. Los cálculos exactos para ambas normas OC-12 y OC-48 se muestran más adelante.

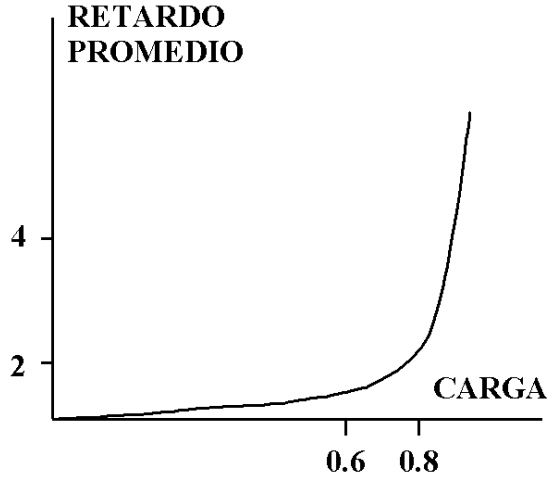

Fig. 3.7: Dependencia entre retardo y carga

En el presente trabajo se considera todas las colas de igual longitud. Aunque se puede considerar en el diseño dos tipos de colas: una cola de menor longitud para células tipo sensible a retardos y otra de mayor longitud para células tipo sensible a pérdidas. Pero

emplear ambos tipos de cola implica aumentar la complejidad de la gestión QoS. Si se considera ambos tipos de cola sensible a retardos y sensible a pérdidas se puede incrementar la capacidad total y en consecuencia la utilización para una CLP determinada.

Una interpretación adicional de la figura 3.7 es el problema de bloqueo HOLB (Head of Line Blocking) que limita el throughput al 58%. En el presente trabajo se considera una gestión QoS en la cual las células que arriban son organizadas en colas generales y la información de los bits de prioridad contienen la información de VC, de la categoría de servicio y de destino. La información por destino y el uso de un buffer de salida minimiza el HOLB.

El análisis del tráfico se basa en modelos de simples a fin de mostrar los principales parámetros del diseño. En la teoría de colas de usa la notación de Kendall, en general en un modelo A/B/X se tiene:

A: Especifica la distribución de tiempo entre llegadas. M es un proceso tipo Markov sin memoria, el tiempo entre arribos es tipo exponencial negativo.
B: Especifica la distribución de tiempo de servicio. D es determinístico, el tiempo de servicio es fijo.
X: Especifica el número de canales de servicio. Un 1 significa un solo servidor.

En el modelo de cola M/M/1, se asume que el patrón de arribos es un proceso Poisson, el tiempo entre arribos es aproximado por una exponencial negativa. El tiempo de servicio también es descrito por una distribución exponencial negativa.

En el modelo M/D/1 el patrón de arribos es el mismo proceso de Poisson anterior. En el tiempo de servicio se considera el hecho que las células ATM tienen longitud fija, se asume que las células entran en servicio tan pronto les permita el servidor y que no esperan por el próximo slot de célula.

En el anexo A se amplían conceptos relativos al modelo de tráfico ATM y la teoría de colas que sirven de apoyo para especificar la Unidad de Clasificación.

Para el diseño de la Unidad de Clasificación, se toma en cuenta los valores medidos de rendimiento de los circuitos de clasificación serie y paralelo. En el capítulo VI se detallan dichos resultados. A continuación con el propósito del diseño global se considera los resultados sobre la plataforma disponible APEX 20KE.

Estimación de los parámetros.

Para la norma OC-48 se tienen enlaces de 2.5 Gbits/seg, por lo cual el tiempo de célula o slot es s= 169.6ns= (53x8 /2.5G). Si consideramos que cada 27 slots de célula, una célula es usada para operaciones de OAM (Operation and Maintenance), luego para el tráfico disponible el tiempo de servicio por célula de (27/26)x169.6ns = 176.123ns. Se considera 256= 2^8 niveles de prioridades, que es suficiente para los principales tipos de servicio de tráfico de la norma ATM.

Para la norma OC-12 se tienen enlaces de 622 Mbits/seg, por lo cual el tiempo de slot es s= 681.672ns= (53x8/622M). Igualmente con la consideración anterior, el tiempo de

servicio por célula es (27/26)x681.672ns= 707.890ns. Se considera 65,536= 2^{16} niveles de prioridades, este gran rango de prioridades permite un manejo flexible para todos los tipos de servicio y clases servicio de tráfico de la norma ATM.

En el diseño se consideran modelos citados anteriormente asumiendo tráfico CBR, se usa el modelo M/M/1 para la norma OC-48. Para la norma OC-12 se usa el modelo M/D/1 ya que se considera un rango de utilización mayor y ofrece una mayor precisión. Se toma en cuenta que la Unidad de Clasificación al mismo tiempo que opera como un sistema de colas, también administra dichas colas basado en prioridades. Por lo cual se considera el tiempo total de retardo requerido para cada norma considerada. En el anexo A se detallan y amplían las características de dichos modelos de colas.

Para la norma OC-48:

El promedio del retardo total "t_q" del sistema, en el modelo M/M/1, depende de la carga ρ y el tiempo de slot de célula "s" para el enlace y está dado por:

$t_q = s + \rho s/(1-\rho)$

El tiempo de servicio de célula es dominante hasta un 60% de utilización. También dentro de este rango de utilización se mantiene la CLP requerida y en el cálculo del retardo es pequeña la diferencia con el modelo M/D/1 [71]. Por lo cual se considera para el punto de operación $\rho = 0.6$, y se tiene t_q= 440.308ns.

Con los valores estimados, que se detallan en el capítulo VI, para una implementación en el FPGA APEX 20KE y un "speed grade" de (–1), se tiene una mayor velocidad del circuito de clasificación paralelo (grupos de 8 vectores de salida cada 20ns) frente a la velocidad del circuito de clasificación serie (un vector de salida cada 25ns). El buffer micropipeline de las direcciones de memoria tiene una velocidad mayor que ambos circuitos de clasificación.

Con referencia a la Fig. 3.5 de la Unidad de Clasificación, el clasificador serie tiene 14 bloques y el clasificador paralelo tiene 6 etapas micropipeline, por lo cual en puede procesar 20 células por cada cola.

Para la cola con una longitud de células B= 14+6 = 20 y considerando los retrasos de la Unidad de Clasificación se tiene valor constante de T_q= 470ns= (14x25ns+6x20ns). Cuando la cola opera a su máxima velocidad es T_q= 20x25ns= 500ns. Luego en promedio T_q= 485ns. Este valor está dentro del rango y puede considerarse un valor del para el peor caso. Los valores de retardo pueden ser alrededor de un 5% menor en la implementación real comparado con los valores de la simulación sobre el FPGA considerado. Por lo cual se considera que el retardo total de la Unidad de Clasificación es T_q= (0.95)470ns= 446.5ns.

El máximo retardo del modelo es la capacidad del buffer "B" multiplicada por el tiempo de slot de célula "s" para el enlace dado. Considerando B= 20, el máximo t_q= 20x170ns = 3.4μs. Por el diseño micropipeline de la Unidad de Clasificación, se tiene que el retardo para la máxima velocidad de clasificación es: T_q= 20x25ns = 500ns, que es mucho menor que el valor considerado en el modelo.

Para esta capacidad del buffer B= 20 y ρ= 0.6, se tiene una garantía de una CLP= 10^{-8} o inclusive menos. La capacidad total para N= 8 puertos de entrada es: 20x8= 160 células.

Justificación.

En el modelo M/M/1, el patrón de arribos representa las células que arriban a la cola con una tasa de "λ" células por slot de tiempo. Se puede considerar que la salida de la Unidad de Clasificación es un proceso del modelo M/D/1, con "p" células por slot de tiempo. Para el caso que no hay pérdida de células (buffer infinito), λ <= p, la tasa de arribo de células λ es menor o igual a capacidad de la tasa de salida de células p. Si no se cumple, el número de células en el buffer crece sin límite y la cola del sistema es inestable.

Para la Unidad de Clasificación, la salida del circuito paralelo tiene la capacidad máxima de p= 170/25= 6.8 células/slot, que permite administrar la cola con un margen de seguridad aceptable y evitar la saturación de la cola.

Por otro lado, el punto de operación se fija para una utilización del 60% y puede ampliarse. El problema de bloqueo de células se evita por el modo de operación de la Unidad de Clasificación. Este diseño considera una política de prioridades para descartar células cuando la memoria está llena y, el uso de un buffer en cada puerto de salida permite extender la utilización a más de 60% dependiendo del algoritmo de asignación de prioridades y los recursos disponibles.

Para la norma OC-12:

Se considera el modelo M/D/1 para una mayor precisión al incrementar la utilización a más de un 60%, por otro lado, el mayor tiempo de slot de célula permite un diseño menos restrictivo. El retardo promedio "t_q" del sistema depende de la carga ρ y el tiempo de slot de célula "s" para el enlace dado:

$t_q = s + \rho s/2(1-\rho)$.

Si consideramos ρ= 0.77, se tiene t_q= 1892.836ns.

Con los valores estimados, que se detallan en el capítulo VI, para una implementación en el FPGA APEX 20KE es suficiente un "speed grade" de (-2). Se tiene una mayor velocidad de circuito de clasificación paralelo (grupos de 8 vectores de salida cada 25ns) frente a la velocidad del circuito de clasificación serie (un vector de salida cada 40ns), también en este caso el buffer micropipeline de las direcciones de memoria tiene una velocidad mayor que ambos circuitos de clasificación.

Con referencia a la Fig. 3.5 de la Unidad de Clasificación, en este caso el clasificador serie tiene 30 bloques y el clasificador paralelo tiene 6 etapas micropipeline, por lo cual en puede procesar 36 células por cada cola.

Para la cola de longitud B= 30+6 = 36 y considerando los retrasos de la Unidad de Clasificación se tiene valor constante de T_q= 1350ns= (30x40ns+6x25ns), que está dentro del rango del modelo M/D/1 para t_q promedio. Cuando la cola opera a su máxima capacidad, el máximo T_q= 36x40ns= 1440ns. Luego el promedio t_q= 1395ns,

que también es inferior al valor dado por el modelo. Los valores reales de retardo pueden ser un 5% menores que los valores en la implementación real sobre el FPGA considerado. Por lo cual se puede considerar T_q= (0.95)1350ns= 1282.5ns.

El máximo retardo del modelo es la capacidad del buffer "B" multiplicada por el tiempo de slot de célula "s" para el enlace dado. Considerando B= 36, el máximo t_q= 36x680ns = 24.480μs. El diseño micropipeline de la Unidad de Clasificación, el retardo para la máxima velocidad es T_q= 36x40ns= 1440ns que también es mucho menor que el valor del modelo.

Para la Unidad de Clasificación, la salida del circuito paralelo tiene la capacidad máxima de p= 680/40= 17 células/slot, que permite administrar la cola con un margen de seguridad amplio y evitar la saturación de la cola.

Para esta capacidad del buffer B= 36 y una utilización de ρ= 0.77, tiene una garantía de una CLP= 10^{-8} o menos. La capacidad total para N= 8 puertos de entrada es: 36x8= 288 células.

Igualmente, el bloqueo de células se evita por el modo de operación de la Unidad de Clasificación. La política de prioridades para descartar células cuando la memoria está llena y el uso de un buffer en cada puerto de salida permite extender la utilización a más del 77% dependiendo del algoritmo de asignación de prioridades y los recursos disponibles.

Consideraciones adicionales.

En ambos diseños para las normas OC-48 y OC-12, un análisis real debe considerar varias clases de prioridad, por ejemplo CBR y rt-VBR tienen ciertos requisitos de prioridad que difieren de la prioridad de nrt-VBR y estos a la vez difieren para tráfico ABR. En [37] se aborda un modelo que toma en cuenta dichas consideraciones para el tipo de conmutador ATM de memoria compartida, los resultados mostrados concuerdan con los parámetros de diseño usados en el presente trabajo. Para la Unidad de Clasificación, un modelo matemático preciso de simulación que tome en cuenta los diferentes requerimientos de las clases de servicio y el algoritmo de administración de la QoS queda como un trabajo futuro.

En el anexo A, se muestran las definiciones necesarias para el modelo de tráfico considerado y criterios adicionales utilizados para la especificación de la Unidad de Clasificación.

CAPÍTULO IV

IMPLEMENTACIÓN DE LOS MÓDULOS BÁSICOS

4.1 Introducción

El objetivo del presente capítulo es la especificación de la biblioteca de módulos de control y de los bloques de procesamiento. En la actualidad se tienen diferentes módulos de control, pero debido a la metodología de diseño del presente trabajo es necesario un enfoque unificado en el diseño de dichos módulos. Otra necesidad es el diseño y especificación de forma ad-hoc de algunos módulos de control imprescindibles dentro de la metodología.

Se abordan aspectos importantes como los hazards o el layout considerando el estilo de diseño micropipeline en una forma general y también para el caso de una implementación FPGA. A continuación se definen los parámetros de rendimiento y se emplea los STGs para formalizar las especificaciones.

Dentro del estilo de diseño micropipeline se consideran los circuitos de control y las restricciones que permiten facilitar su descripción, igualmente se trata el elemento de retardo y el método empleado para su implementación.

Otro aspecto tratado a continuación es la descripción de los bloques constitutivos utilizados en el presente trabajo, se establecen las ventajas y limitaciones del empleo del lenguaje VHDL y los LPMs. Finalmente se aborda la verificación funcional del diseño micropipeline. Se establecen los parámetros de evaluación los cuales, para el caso de la implementación FPGA, serán corroborados posteriormente mediante mediciones experimentales.

4.2 Especificaciones de la biblioteca de módulos de control

En la presente sección se reseñan las consideraciones principales para la implementación de la biblioteca de módulos de control, aspectos como los hazards o el layout son abordados para ambos casos ya sea en forma general o para el caso que se oriente a una plataforma de FPGA. Posteriormente se detallan los requisitos para el estilo de diseño micropipeline, se definen los parámetros de su rendimiento y se concreta para el caso de la implementación FPGA. Finalmente, para formalizar la descripción de la biblioteca de módulos de control se desarrolla mediante STGs la correspondiente especificación de los principales módulos.

4.2.1 Requisitos generales

Al nivel de la implementación de la biblioteca de módulos de control se debe considerar la problemática de los hazards y el layout necesario para evitar y/o minimizar los problemas derivados de los hazards. Los requisitos y las recomendaciones se discuten en forma general y en particular para los FPGAs.

A) Hazards.

En los circuitos asíncronos deben ser diseñados cuidadosamente para evitar los hazards los cuales pueden causar que se activen falsas transiciones en las subsiguientes etapas. De forma general, el circuito debe considerar cualquier estado intermedio producido por cambios múltiples de las variables de entrada ya que no existe una señal de reloj para sincronizar las llegadas de las entradas.

Para los hazards se tienen definiciones de acuerdo a su naturaleza, los principales tipos son:

Hazard estático-0: Se define cuando en una salida de valor estable lógico 0 se presenta un valor momentáneo de un valor lógico 1. De igual manera se define un hazard estático-1.

Hazard dinámico: Se presenta cuando una salida en lugar de tener una única transición de su valor lógico (0 => 1), se tiene 3 o más transiciones lógicas (0 => 1 => 0 => 1).

Hazard funcional.

Pueden ser estáticos o dinámicos. Este tipo de hazard es inherente a la función, solo puede ser eliminado mediante una apropiada ubicación (placement) de retardos en el circuito. Las causas principales son las entradas que pueden ser imprevisibles en cuanto a su comportamiento y los retardos de las puertas que pueden diferir de los valores especificados.

Hazard lógico.

Es una consecuencia de la implementación. Es causado por retardos de las puertas lógicas, se puede presentar para un cambio en las entradas y puede ser independiente de la existencia de un hazard funcional. También pueden ser estáticos o dinámicos.

Hazard esencial.

Los hazard de los circuitos secuenciales que ocurren independientemente de la implementación y los que son evidentes en la Tabla de Flujo (FT) se denominan esenciales. Un hazard esencial, es causado por un cambio en entrada que alcanza diferentes partes del circuito a diferentes tiempos e inclusive algunas veces después que una variable de realimentación ha sido producida, lo que puede resultar en que la red alcance un estado errado.

Estos pueden ser eliminados añadiendo retardos en la red. Estos retardos aseguran que la lógica combinacional complete su respuesta a un cambio de la entrada antes que ésta

sea afectada por el cambio de las variables de estado. Los elementos de retardo serán necesarios solo para variables de estado que cambien durante las transiciones que involucran hazards esenciales.

Métodos de minimización libre de hazards han sido extensamente estudiados [12][14], en [14] se desarrolla un algoritmo de minimización de dos niveles libre de hazards, en [38] se revisa la problemática de los hazards enfocado hacia FPGAs. Se puede resumir que hazards debidos al cambio de una sola entrada pueden ser evitados añadiendo términos de suma de productos redundantes y transformaciones lógicas adecuadas. Para el caso de cambios simultáneos de varias entradas es difícil controlar los hazards que se pueden producir. Por lo cual la solución adoptada en el presente trabajo es restringir a un solo cambio a la vez de las entradas del circuito. Esto es posible en los módulos de control ya que son circuitos de pocas entradas y salidas y, además, considerando un diseño global adecuado.

Recomendaciones.

Se pueden tener en cuenta algunas restricciones y observaciones para la construcción de los bloques de control básicos que permitan su implementación de forma sencilla y evitar los hazards, por ejemplo:

- Se restringe el transmitir dos señales consecutivas sobre una misma entrada sin que intervenga una salida (comportamiento monótono). Implica un modo de operación serie donde los eventos de entrada y salida en los módulos de control deben alternarse estrictamente. Esto se garantiza en caso de micropipelines por el uso de protocolos handshaking que son utilizados entre etapas.

- En el data-path de los micropipelines asíncronos, cuando los retrasos de las etapas son dependientes de los datos de entrada, el control puede adaptarse a dicho retraso en cada etapa. Cuando los retrasos son distintos en cada etapa la latencia está limitada por la suma de los retrasos de las etapas individuales.

- Los hazards debidos a cambios de una sola entrada son relativamente fáciles de evitar pero en el caso de múltiples cambios simultáneos en las entradas son muy complicadas de detectar. En caso de los FPGAs, las propiedades de las LUTs (look-up table) ofrecen mayor control sobre los hazards.

B) Layout.

Las conexiones internas tienen un retardo intrínseco, su efecto es más notorio en el caso de las nuevas tecnologías submicrónicas y para el caso de conexiones programables. Por lo cual son necesarios cuidadosos supuestos en dichos retardos, por lo cual para facilitar el layout se definen las propiedades de región equipotencial y derivación isócrona en el diseño de circuitos asíncronos, los que se definen a continuación.

Región equipotencial.

Son áreas de pequeños circuitos donde los retardos de las líneas son despreciables, tal que se puede considerar una señal como idéntica en todos los puntos de la línea. Por

ejemplo el retardo de las líneas de realimentación que deben ser instantáneas para el correcto funcionamiento de dicho circuito. Se puede restringir un circuito a una pequeña área y las facilidades en el layout de las herramientas CAD pueden ser usadas para lograr regiones equipotenciales. En los compiladores de FPGAs se tienen facilidades de "cliques" (Altera) o RLOC (Xilinx) que permiten definir regiones equipotenciales restringiendo la ubicación de un grupo de celdas lógicas elementales, LEs (Altera) o CLBs (Xilinx), a una localización dada por las filas y columnas de la matriz del FPGA.

Derivación isócrona.

Es una derivación múltiple de líneas de conexión donde se tienen restricciones de los retardos acumulados hasta los puntos terminales incluyendo las cargas que son impulsadas. La distribución de una señal hacia varias puertas puede ser realizada con seguridad mediante una sola línea que es extendida hacia el conjunto de puertas que forman una región equipotencial, de esta forma se evitan derivaciones largas múltiples. En los FPGAs se tienen líneas de conexión de baja latencia de extremo a extremo que permiten minimizar los retardos.

Recomendaciones.

- En la práctica, las derivaciones isócronas pueden ser consideradas como pequeñas regiones equipotenciales, pero derivaciones largas deben ser realizadas mediante restricciones del "routing" en vez de restricciones de área.

- Aunque las herramientas de place&route usualmente están separadas de la especificación HDL de alto nivel y el proceso de síntesis, algunas herramientas permiten que los componentes puedan ubicados adyacentes unos a otros de forma relativa.

- Los FPGAs tienen facilidades que permiten que circuitos sean agrupados de forma relativa uno a otro. También ofrecen líneas para conexiones de puntos distantes de baja latencia que puede ser útiles para distribuir conexiones con muchas derivaciones cortas.

4.2.2 Requisitos para el diseño micropipeline

En forma general un pipeline puede definirse de varias maneras y se pueden varias formas de implementación. En presente trabajo el concepto de pipeline esta basado en la definición de micropipeline de Sutherland [6]. Para un diseño micropipeline se deben observar ciertos requisitos que a continuación se detallan. Las recomendaciones generales planteadas se concretan para el caso de los FPGAs.

A) Concepto de diseño micropipeline.

En un diseño micropipeline se intenta determinar qué operaciones son susceptibles a ser realizadas de manera concurrente sin que una afecte a otra. De esta forma se incrementa el número de operaciones en un periodo de tiempo a pesar de que el tiempo necesario para realizar las demás operaciones se mantenga igual.

El primer paso es determinar si el circuito puede ser dividido en operaciones independientes, luego es encontrar un número óptimo de etapas tal que la complejidad sea tratable. Teóricamente si se tiene mayor número de etapas, un mayor número de operaciones puede ser realizada concurrentemente y obtener mayor throughput. Sin embargo, se necesita más componentes en registros para las interfaces entre las etapas. Esto último para implementar micropipelines asíncronos en FPGAs puede llegar a ser prohibitivo debido a que son limitados los recursos de componentes.

Un registro FIFO micropipeline tiene dos parámetros importantes los cuales define su rendimiento:

Latencia: Es la velocidad con la cual nuevos datos se desplazan a través de un registro FIFO vacío. Este parámetro no es importante cuando se tiene datos que fluyen de manera constante a través del FIFO. Puede ser importante cuando el FIFO debe ser vaciar sus datos de forma regular y empezar un nuevo procesamiento.

Throughput: Es el valor máximo de tasa constante de datos a la cual puede operar el FIFO.

B) Implementación en FPGAs.

Los FPGAs ofrecen prototipaje rápido e implementación de circuitos digitales con mínimos costos agregados. Actualmente los FPGAs están orientados para los circuitos que basan su operación en una o varias señales de reloj, pero pueden ser útiles para investigar nuevas arquitecturas asíncronas, por lo cual un aspecto a considerar son los hazards en los FPGAs empleados y a fin de evitar su aparición es necesario aplicar algunas recomendaciones.

Por otro lado, los micropipelines asíncronos requieren circuitos adicionales para realizar el protocolo de señalización que coordinan los tiempos entre las etapas, además de los registros para mantener los estados en el bus de datos. También requieren dos líneas de Req y Ack para conectar las etapas consecutivas. Esto implica recursos adicionales en el trazado (routing) de las conexiones, también los elementos de retardo necesitan puertas y las conexiones necesarias. Todas estas consideraciones limitan el tamaño del circuito que puede ser implementado en un FPGA determinado.

De lo anterior y también basado en el trabajo existente del análisis de hazards para implementaciones al nivel de puertas [38], se puede llegar a establecer las condiciones que deben cumplirse en las implementaciones basados en LUTs (look-up table) y lograr que tengan un comportamiento libre de hazards. Los estudios anteriores permiten considerar valido dichos resultados para ambos FPGAs: Altera FLEX10K, APEX20KE y Xilinx XC4010, en la sección 6.2 se detallan las implementaciones en dichas plataformas.

En general los diseños implementados en FPGAs pueden presentar hazards debido a dos razones:

1) No aseguran una predicción exacta del retardo de las conexiones (routing).
2) No es posible restringir los cambios de los valores lógicos de las variables de entrada.

Como una consecuencia, una implementación en LUT, puede estar libre de todo hazard lógico pero tener algún hazard funcional. Una LUT produce la salida y se mantiene estable durante la transición, eliminando hazards lógicos esto es debido a que son casi iguales los retardos de propagación desde las entradas hacia las salidas. También si se tienen cambios simultáneos de las entradas y se tiene un hazard funcional, entonces se puede producir un glicht en la salida del LUT. Una forma de eliminar este glicht es cambiando a la vez solo una entrada de selección del LUT, este concepto es una consecuencia de las recomendaciones generales para la implementación de los módulos de control dados anteriormente.

Por otro lado, todas las funciones que son implementadas usando LUTs son libres de hazards esenciales. Esto es debido a que un cambio en la entrada es detectado por el generador de función y al mismo tiempo la correspondiente salida es seleccionada de la configuración de bits. Por lo tanto la nueva salida no es realimentada hasta que todo el circuito ha detectado el cambio en la entrada, esto es válido en los módulos de control ya que son circuitos pequeños e implementados con un número mínimo de LUTs. Por ejemplo el elemento-C ocupa un solo LE (Logic Cell). En las secciones 6.3 y 6.4 se evalúan estos módulos de control.

Recomendaciones.

Para los FPGAs, la implementación en LUTs, donde los resultados de la función son generados mediante un lookup table en lugar de puertas lógicas, tiene algunas propiedades a considerar:

1) El retardo de propagación desde cada entrada hacia la salida se mantiene constante y es igual al tiempo retardo de la memoria lookup table. Esta propiedad elimina las líneas críticas y los circuitos están libres de hazards esenciales y estáticos.

2) El número de variables de una función es un indicador del número de LUTs requeridos para implementar dicha función, la salida mantiene el valor lógico durante las transiciones eliminando hazards lógicos. A diferencia de una implementación con puertas lógicas donde es necesario introducir minterms redundantes para mantener la salida durante dichas transiciones.

3) Una función que se implementa en LUTs está libre de hazards para cualquier transición múltiple en sus entradas, pero un hazard funcional podría producir un glitch. Las implementaciones no contienen hazards esenciales, pero si las entradas cambian de una en una, si están libres de todo tipo de hazards.

4) Cumplir las restricciones en los retardos de las líneas de realimentación para conseguir un comportamiento libre de hazards. Estas restricciones deben ser observadas en la ubicación y conexionado de cada módulo.

Está sobre entendido que el esquema empleado es valido cuando los circuitos son usados observando las restricciones de operación ambientales y dentro de condiciones de operación típica del dispositivo.

Para el data-path de circuitos asíncronos se tiene una implementación bundled-data que permite significativo ahorro en la lógica y conexiones del circuito, esto es

particularmente conveniente cuando la implementación en un FPGA donde las líneas de conexión (routing channels) son limitadas. Una desventaja de este diseño bundled-data es que los elementos de retardo introducidos tienen que compensar el retardo de peor caso en cada bloque a fin de permitir el correcto funcionamiento del micropipeline. Aunque por la naturaleza asíncrona se puede variar la velocidad de operación de acuerdo a los requerimientos en un instante determinado, por lo cual al considerar el rendimiento global del sistema se mantiene la aproximación al comportamiento típico en lugar del peor caso.

4.2.3 Especificación mediante STGs

Para la metodología de diseño, la descripción de la biblioteca de módulos debe ser realizada de forma clara y sin ambigüedades, por lo cual el empleo de los STGs permite especificar correctamente cada módulo de control. A continuación, a fin de mostrar el procedimiento, se desarrolla dicha especificación para los principales módulos de control y de forma similar se considera para los demás módulos de control.

Elemento-C.

La Figura 4.1A muestra la especificación STG. El elemento-C realiza una función dirigida para eventos, la salida antes de anunciar un evento, espera hasta que se ha recibido un evento en ambas entradas. Responde a la siguiente especificación: si los estados de las entradas son iguales, entonces se transfiere dicho estado a la salida, si no son iguales se mantiene el estado anterior. En Figura 4.1B se tiene el símbolo y el esquema del circuito que implementa el elemento-C. Se realiza con un simple sumador completo de un bit. Esto permite una reducida área y menor retraso.

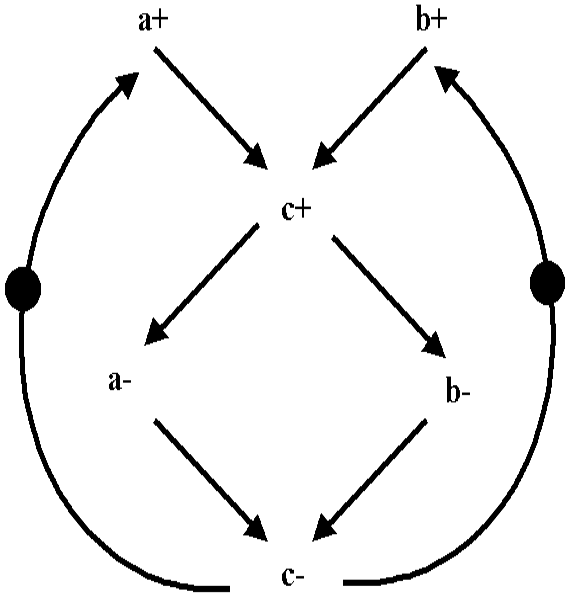

Fig. 4.1A: Especificación STG del elemento-C

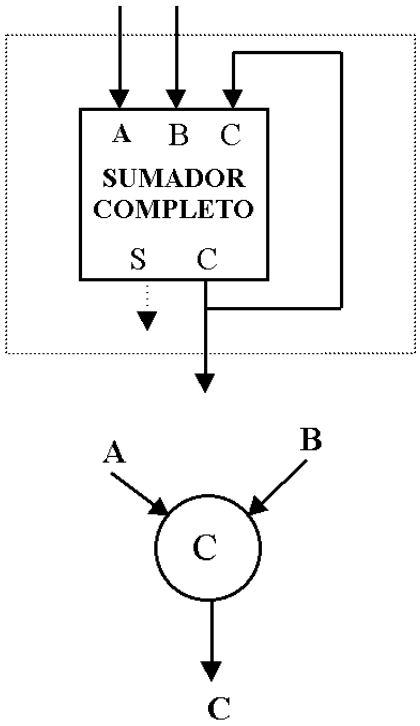

Fig. 4.1B: Símbolo y circuito del elemento-C.

La ventaja radica en que las herramientas CAD comerciales, tienen una implementación de la celda sumador completo eficiente en todo sentido ya que es el elemento básico para la construcción de otros componentes aritméticos fundamentales.

Para la implementación libre de hazards de una puerta elemento-C, las entradas deben monótonas, es decir, que no se permite dos transiciones consecutivas sobre una misma línea de entrada. Una transición en una línea de entrada puede ser seguida por otra transición en la otra entrada solo si la salida ha cambiado luego de la transición en la primera entrada o los estados internos han cambiado. La puerta elemento-C podría tener múltiples cambios en sus entradas por lo tanto podría tener hazards funcionales. Sin embargo, está libre de hazards funcionales debido a que no tiene transiciones de valores intermedios que producen diferentes resultados. Pero una línea larga de realimentación o un rápido cambio en una entrada mientras aún se respete el modo de comportamiento monótono, podrían dar como resultado un hazard. Por lo cual la línea de realimentación de la puerta elemento-C, debe ser trazada tal que su retardo debe ser menor que la suma del retardo en detectar la salida y el cambio de la entrada.

La descripción VHDL para una puerta elemento-C de dos entradas a, b y salida c se adjunta en la biblioteca de módulos, para su empleo de control en el micropipeline se añade una señal de reset y un inversor a la entrada b. La implementación mediante VHDL-LPMs permite un retraso mínimo y operación libre de hazards. En el caso de los FPGAs, el elemento-C es implementado en un solo LUT y ocupa el área lógica mínima (1 Logic Cell), la línea de realimentación es interna y de retardo mínimo, por cual está libre de hazards. En el capítulo VI se detalla la implementación en los FPGAs disponibles.

Elemento SINCT.

El módulo de control SINCT es un diseño ad-hoc de comportamiento monótono para la metodología de diseño propuesta. La especificación STG se muestra en la Figura 4.2. Este circuito de control permite actualizar la carga del los latches de las entradas al bloque de comparación dependiendo de la condición de la otra señal de entrada, según la siguiente especificación:

- Si la señal CE está en nivel lógico 1 y ocurre una transición cualquiera en la señal CTRL entonces ocurre una transición en la salida Lout. Si CE está en nivel lógico 0 se mantiene el estado actual en la salida Lout sin importar las transiciones de CTRL.

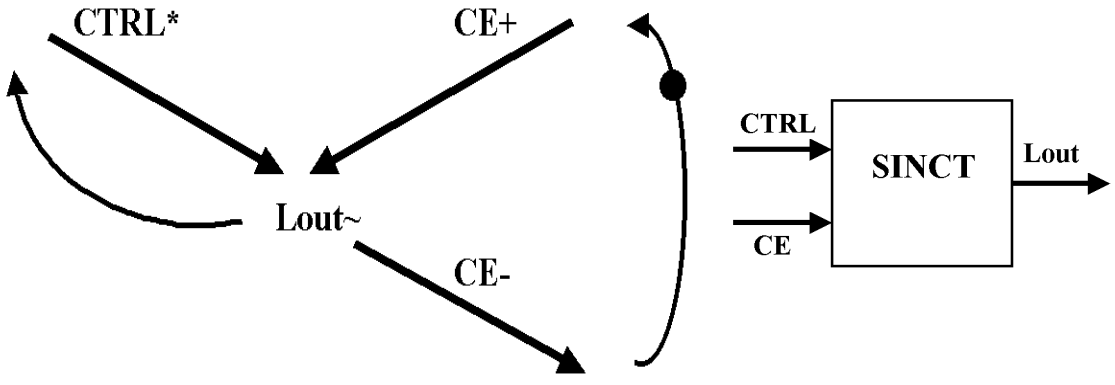

Fig. 4.2: Especificación STG del bloque SINCT

- La transición positiva de señal CE siempre se presenta antes de la señal CTRL y dicha transición positiva habilita el cambio de estado en la salida Lout.
- Las entradas CTRL y CE cumplen que cambian consecutivamente y la salida Lout cambia como respuesta a dichos cambios de las entradas.

La implementación en los FPGA realiza mediante dos FF tipo T que se activan para cada flanco de subida o bajada respectivamente y una puerta XOR produce las transiciones respectivas. No presenta hazards por su modo de operación monótono, no tiene ninguna línea de realimentación y se controla la señal "enable" de los FF. Dicha implementación tiene la ventaja de ser portable a todas las plataformas de CAD comerciales. Otro tipo de implementación puede ser derivada mediante métodos desde el diagrama de estados, aunque el coste de área y análisis de hazards es un factor a tomar en cuenta. La descripción VHDL del elemento SINCT adjunta en la biblioteca de módulos y en el capítulo VI se detalla la implementación en los FPGAs considerados.

Otros componentes.

Los demás componentes de los bloques de control, incluyendo el bloque de retraso y los bloques de procesamiento. Son circuitos bastante conocidos y su implementación es sencilla. Los detalles de dichas implementaciones se muestran en las secciones 4.3, 4.4 y en el capítulo VI se muestra los resultados de la implementación en los FPGAs.

Es importante señalar que el diseño de los demás bloques de control es realizado asumiendo las condiciones de comportamiento monótono, se asume el protocolo de señalización de transiciones de dos fases basado en handshakes y que los datos son transmitidos en una característica bundled-data. Los latches también tienen un orden establecido para los eventos de sus entradas, además, se considera las restricciones en las líneas de realimentación mediante las recomendaciones dadas para el layout. Por lo cual todos los bloques de control se pueden considerar libres de hazards. Los bloques procesamiento son implementados optimizando el coste de área/velocidad. Se asume que los circuitos de control permiten el establecimiento de las salidas de los bloques de procesamiento antes que sean ingresadas en los latches del micropipeline.

4.3 Clasificación de los Módulos

Para el estilo de diseño micropipeline se han propuesto en los últimos años varios circuitos de control, en esta sección se presentan de forma unificada los principales módulos que se pueden emplear para las funciones de control, se tienen los circuitos de control clásicos y los de control Tangram. Dentro de metodología propuesta, se establecen las restricciones que permiten usar circuitos simplificados de dichos módulos de control. A continuación se aborda el módulo elemento de retardo, se revisan los principales aspectos de su problemática y se establece el método empleado para su implementación.

4.3.1 Elementos de circuitos de control

Bloques de control clásicos.

En la Figura 4.3 se muestra los principales de bloques de control clásicos. A continuación se describe dichos bloques de control y algunas definiciones adicionales. Para evitar los hazards se debe cumplir el comportamiento monótono, es decir, en cada bloque de control se asume la restricción que no es posible transmitir dos eventos consecutivos sobre una misma entrada sin que se haya tenido un evento en la salida o un cambio de estados internos. Esto significa que los eventos de entrada y de salida se alternan. Para diseñar circuitos basados en micropipelines, se requieren unos pocos bloques de control que han sido bien estudiados, su comportamiento es especificado en función de eventos abstractos [38], [41]. Inicialmente fueron propuestos por Sutherland [6] y posteriormente se han ido añadiendo algunos más:

Wire.

Propaga eventos desde su entrada a su salida. Es una línea de conexión simple

Fork.

Propaga un evento sobre cada una de sus salidas para cada evento recibido sobre su entrada. Es una o varias derivaciones de la línea de conexión.

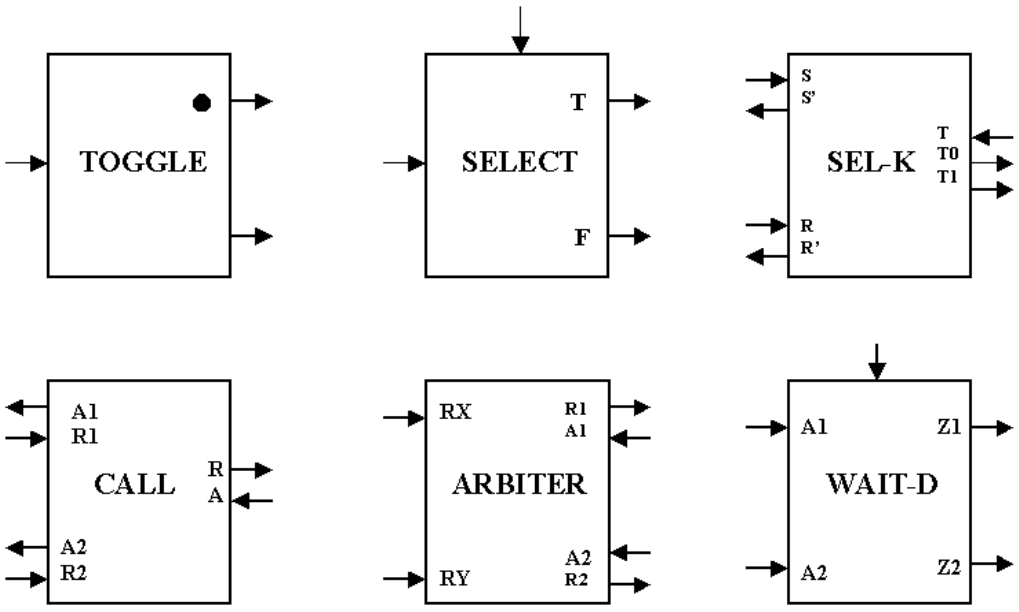

Fig. 4.3: Bloques de control clásicos

Elemento-C.

Es la conocida puerta Muller-C, también es denominado elemento-C simétrico. En la sección 4.2.3 se amplía su descripción.

Joint.

Produce un evento de salida cuando ha recibido eventos en ambas de sus entradas. Su implementación es una puerta elemento-C.

Merge.

Produce un evento en su salida siempre que reciba un evento en cualquiera de sus entradas. Su implementación es una puerta XOR (OR exclusivo) donde una transición en cualquiera de sus entradas produce una transición sobre la salida.

Toggle.

Se muestra en la figura 4.3. Permite transferir un evento desde su entrada hacia una de sus dos salidas de manera alternada. El primer evento en arribar es transmitido hacia la salida marcada con punto, el segundo evento es transmitido en la otra salida.

Select.

Se muestra en la figura 4.3. Permite que una variable dirija el evento de entrada hacia la una de las salidas T (true) o F (false).

Select Keller (Sel-K).

Se muestra en la figura 4.3. Permite recuperar estados lógicos, su estado interno lógico es puesto a 1 mediante un evento en la entrada S. Un evento en S´ confirma que su estado ha sido cambiado. De igual forma se pone a 0 mediante la entrada R y confirma en R`. Un evento en la entrada T permite saber su estado interno, es 0 si se obtiene un evento en la salida T0 y es 1 si se tiene un evento en T1.

Call.

Se muestra en la figura 4.3. Permite a dos procesos independientes compartir un subproceso común, mediante señales de petición (Req) y término (Ack). Los procesos que llaman deben mutuamente excluyentes, sino el acceso debe ser a través de un árbitro.

Arbiter.

Se muestra en la figura 4.3. Acepta peticiones asíncronas en sus dos entradas y transfiere solo una de ellas, en un instante determinado, hacia el recurso compartido. Cuando concluye dicha petición transfiere la otra petición sí aún está pendiente. Usualmente se construye basándose en el denominado elemento MUTEX (mutual exclusion). El diseño de árbitro debe considerar el problema de metaestabilidad interna del circuito para garantizar una correcta operación.

Decision-wait (Wait-D).

Se muestra en la figura 4.3. Es una generalización de la puerta C, permite que eventos puedan presentarse en cualquier orden en las entradas “fire” y en cualquiera de las entradas A1 y A2 (pero no en ambas al mismo instante), y produce un evento en una de las salida Z1 o Z2 de acuerdo a qué entrada A realizó una transición. Una vez que cualquiera de las salidas Z ha realizado la transición, el circuito está listo para su funcionamiento otra vez.

Elemento-C generalizado.

Un elemento-C generalizado es una puerta secuencial en la cual ambos eventos (transiciones de subida o bajada) de salida pueden ser escritas como una conjunción de un sub-grupo de sus entradas. El símbolo para dicho elemento es derivado del elemento-C simétrico. La entrada que contribuye en ambos eventos es conectada a la base del símbolo, la entrada que contribuye en un solo evento es conectada al terminal de extensión, puede ser +/- según la transición del evento.

Latch.

El latch propuesto por Sutherland [6] opera directamente con las señales de control y una señalización de dos fases, es un circuito sencillo pero es más apropiado para un diseño al nivel de transistores. Posteriormente se han propuesto diferentes tipos de latch [41] que operan para señalización de 4 fases y modos de operación normalmente transparente o normalmente cerrado. Una señalización de dos fases permite un mayor throughput y una operación tipo normalmente cerrado evita la propagación de estados transitorios que significan un consumo de potencia.

En el presente trabajo se diseña el modulo REG que es un latch activado por eventos, normalmente cerrado, opera con las señales de control del micropipeline de dos fases. Se implementa mediante dos latch simples, la habilitación de cada evento captura los datos y los mantiene hasta que se presente el siguiente evento. Para minimizar la sobre carga de la señal de activación de los latches, los módulos REG se agrupan según los datos que procesan y restringidos en un área de layout localizada. Basándose en el módulo REG se construye el registro FIFO (firts-in-first-out) del micropipeline para conformar la arquitectura de la Unidad de Clasificación.

Componentes handshake Tangram.

Se considera un conjunto de componentes básicos que puede usarse de manera sencilla y pueden ser implementados mediante los bloques clásicos. También por el comportamiento monótono, se asume la restricción que para cada bloque de control no es posible transmitir dos eventos consecutivos sobre una misma entrada sin que se haya tenido cambio de evento en la salida. Esto significa que los eventos de entrada y de salida se alternan.

En general se tienen uno o más procesos concurrentes que se comunican a través de canales sincronizados, los procesos se descomponen en una interconexión de componentes básicos que se comunican mediante un protocolo handshaking. Los canales tienen un puerto como interfaz de comunicación.

Cada puerto es una interfaz handshake que consiste en una línea de request y una línea de acknowledgment. El círculo vacío indica un puerto pasivo, la línea de request es la entrada y el acknowledgment es la salida. De la misma forma, el círculo lleno es un puerto activo, la línea de request es la salida y el acknowledgment es la entrada.

Una conexión directa entre puertos de dos componentes implica que un puerto es activo y el otro pasivo. Dos puertos del mismo tipo pueden ser conectados mediante un tercer

proceso. La Figura 4.4 muestra los componentes principales y a continuación se describen.

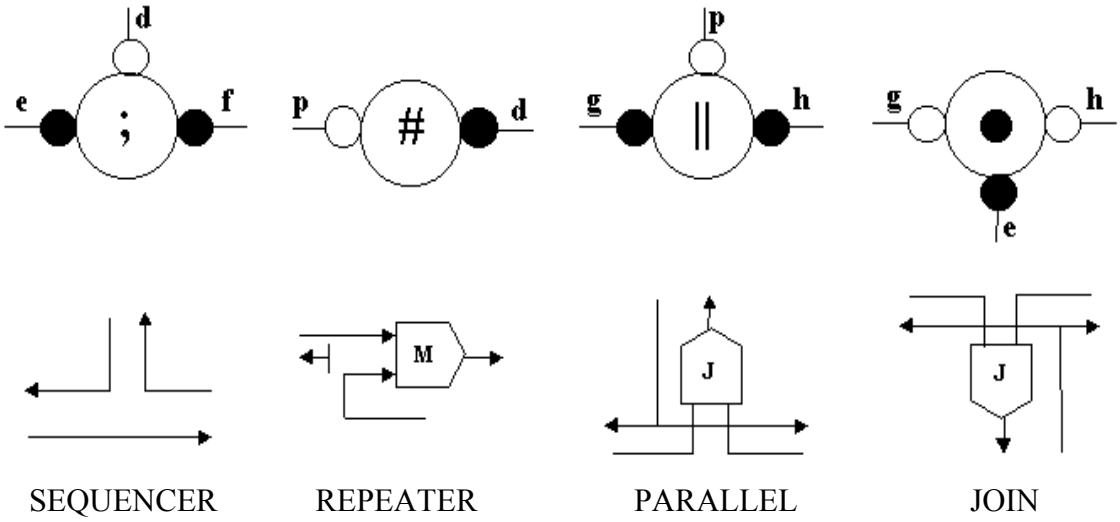

Fig. 4.4: Bloques de control Tangram

Sequencer (;).

Se muestra en la figura 4.4. Es usado para componer dos procesos handshake en secuencia y es implementado usando tres "wire" (definido anteriormente). Un Req activa el puerto **d** que a su vez realiza sincronizaciones (handshakes) a través de **e** y luego hacia **f**. Cuando ambos procesos han concluido, **d** responde con un Ack.

Repeater (#).

Ejecuta el proceso en su puerto activo **d** de manera indefinida. Permite realizar una infinita sucesión de sincronizaciones a través del puerto **d** una vez que ha sido activado por **p**. La línea de Ack del puerto pasivo **p** es puesta al terminal gnd ya que el proceso nunca termina.

Es implementado mediante un elemento Merge (M). La señal de Ack en el puerto activo **d** se convierte en una señal de Req, es necesario que exista un retardo en esta línea y no introducir un bucle combinacional.

Parallel (||).

Permite descomponer en paralelo hacia dos procesos handshake de los puertos **g** y **h** y cuando ambos procesos han terminado se produce un acknowledgment hacia el puerto **e**. Se implementa mediante elementos fork y joint (elemento-C) descritos antes.

Joint (·).

Permite sincronizar dos procesos, ejecutar alguna acción común y reasumir la ejecución cuando la acción común se completa. Cuando se reciben ambos Req en los puertos **g** y

h, se activa un Req en el puerto **e** para iniciar la acción común. Una vez concluida se transmite un Ack hacia **g** y **h**. Se implementa mediante elementos fork y joint (elemento-C) descritos antes. Es el dual de (||).

Circuitos de control ad-hoc.

Para el presente trabajo se han diseñado los siguientes circuitos de control propios:

SINCT.

Bloque control desarrollado en el presente trabajo permite actualizar la carga del los latches de las entradas al bloque de comparación dependiendo de la condición de una señal de control. Se describe en las secciones 4.2.3 y 5.4.1.

Interfaz de sincronización.

Circuito de control que permite sincronizar entre la salida del circuito clasificador serie y la entrada el circuito clasificador paralelo, una configuración simple es poner en cascada varios elemento-C de acuerdo al número de colas de clasificación. Se detalla en la sección 5.5.1.

4.3.2 Elemento de referencia de retardo

Para la implementación del modulo referencia de retardo y dentro de la problemática de los retardos en los circuitos asíncronos micropipeline se debe considerar simultáneamente dos aspectos:

- Estimación de los retardos de los diferentes bloques lógicos constitutivos incluyendo el retardo de las líneas de conexionado.

- Implementación de un elemento de referencia de retardo que permita compensar los retardos de los bloques lógicos.

A) Estimación de los retardos.

Se tienen numerosos factores que influyen en los retardos desde factores de fabricación tecnológica, estructura de las puertas lógicas y de condiciones de operación del dispositivo. Se pueden estimar los retardos desde un nivel de circuitos de transistores y modelos SPICE, pero en circuitos complejos puede ser prohibitivo por el tiempo y capacidad de computo necesarias.

Otros métodos de cálculo, por ejemplo el descrito en [42], se basan en la velocidad de los transistores, la relación salida/entrada de capacidades, la estructura de las puertas y capacidades parásitas. Este método de cálculo es general, se puede aplicar a circuitos síncronos y asíncronos. Aunque los cálculos son simples y pueden realizarse desde el esquema de puertas lógicas, no es apropiado cuando se tiene una descripción VHDL y debido a que algunos parámetros tecnológicos no siempre son conocidos con exactitud, como por ejemplo en las actuales tecnologías submicrónicas.

Cálculo práctico de los retrasos.

Una forma práctica de estimar los retardos es mediante la regla del 50/50 [66], esta regla referida a FPGAs permite una aproximación al análisis temporal. Considera el retardo de los bloques en cualquier conexión simple es aproximadamente el 50% del retardo total que presenta después que el diseño ha sido compilado. Por ejemplo para un conexionado de 10ns de retardo lógico de bloque se debe considerar 20ns de retardo total luego de que se ha realizado el place&route. En el caso de diseños extremadamente densos, el retardo de conexionado se debe considerarse como más del 50%.

Otro factor que debe añadirse en el cálculo es la variación del proceso tecnológico para el peor caso, usualmente es un parámetro dado por cada fundición de silicio en particular. Informes al respecto en aplicaciones similares [67] indican que para el peor caso la dicha variación puede ser del 30%.

Tomando en cuenta lo anterior se estructura un método práctico de para la evaluación de los retados, esto permite una aproximación que luego es corroborada mediante las simulaciones del circuito para la implementación en FPGAs. En el caso de una implementación ASIC, el método siendo general puede aplicarse y queda como un trabajo futuro a realizar.

El primer paso es el cálculo del número de niveles lógicos de los bloques. En el capítulo VI se muestra la evaluación respectiva de los niveles de lógica de los bloques lógicos constitutivos de la Unidad de Clasificación. El compilador da para cada bloque en cuantos niveles de lógica se implementa. El uso de LPMs (descritos en la sección 4.4.2) y que permiten independencia tecnológica, hace que sean implementaciones eficientes y se mantengan similar estructura en las diferentes plataformas tecnológicas.

Una vez que se ha calculado la profundidad lógica del mayor retardo, para una plataforma determinada, se toma como medida de referencia el retardo de un nivel lógico y se calcula el número de unidades de retardo para compensar dicha profundidad lógica. En el capítulo VI se muestra los detalles de los cálculos.

Las consideraciones en el layout permiten considerar que los retardos de conexionado son para la ubicación óptima de los bloques. Luego mediante el compilador y simulaciones respectivas se verifica dichos cálculos. El reporte de tiempos describe los retardos de los bloques lógicos y el retardo del conexionado para el peor caso, por lo cual el retardo de compensación representa con seguridad el mínimo necesario.

B) Implementación del elemento de referencia de retardo.

Un objetivo principal para el elemento de referencia de retardo es que sea portable a cualquier plataforma de implementación, por lo cual se diseña de forma ad-hoc. En la Figura 4.5 se muestra el circuito del elemento referencia de retardo. Se usa una puerta elemento-C, como se mostró implementada con un sumador completo de 1 bit, cuyas entradas son “a”, “b” y la salida es “C”. De la respectiva ecuación boolena se tiene:

$C= ab + c(a+b)$

Cuando a= b, se tiene:

C= a + c (a) = a (1+c) = a

Se muestra que la salida es igual a la entrada pero introduce un tiempo de retraso propio de la implementación y la tecnología.

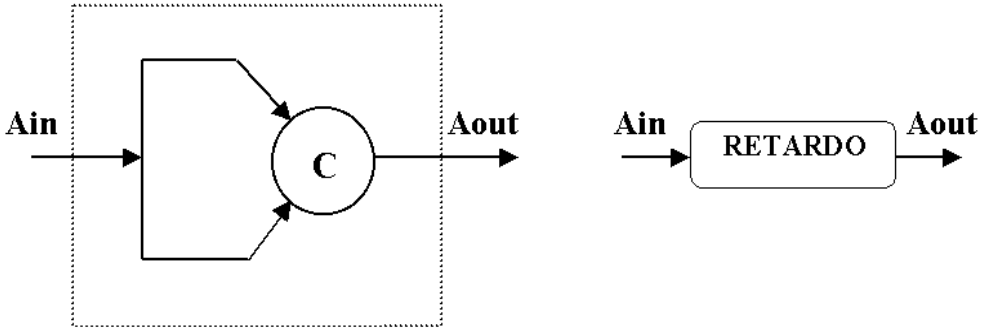

Fig. 4.5: Circuito referencia de retardo

Esta implementación tiene las siguientes ventajas:

- El uso de un elemento-C permite tener una estructura de referencia para estimar el costo en área de los bloques de retardo. Para los FPGAs ocupa la mínima área lógica (1 logic cell).
- La unidad de retardo permite un ajuste óptimo de la cadena de un bloque de retardo, cada paso incremento equivale al retardo de la unidad de referencia.
- La implementación en base un elemento-C, que es una estructura realimentada de área mínima, permite un fácil manejo de los retardos. Permite que el compilador reconozca cada unidad y no simplifique la cadena de retardo, usualmente los compiladores de FPGA simplifican si se usa una cadena simple de inversores o para otra configuración el compilador podría identificarlo como una estructura ilegal.

El bloque de compensación de retardo proporciona un retraso total que es equivalente a un número determinado de unidades del elemento de referencia de retardo. Luego es necesario calcular el retardo de un nivel lógico y del equivalente de una unidad de referencia de retardo.

El método más sencillo es considerar una cadena de varias unidades de retardo y que sea lo suficientemente larga para extraer un valor promedio del retardo, de esta forma se tiene un valor muy aproximado del retardo de cada unidad y dicho valor se usa para estimar los “matched delays”. De las simulaciones para cada plataforma, que incluye los pads de entrada y salida y retardo de conexiones, se estima el retraso que introduce una unidad de retardo.

En los FPGAs es posible usar otros elementos para implementar retardos, en el caso de Altera se tiene el buffer LCELL o directivas tipo WYSWYG (What You See Is What You Get) que permiten que el compilador no simplifique una cadena de retardos. Pero esta opción está limitada a este tipo de plataforma.

En la biblioteca de módulos se tiene la descripción VHDL de la implementación de la unidad de retardo y en el capítulo VI se muestra los cálculos para estimar los matched-delays para la implementación FPGA.

4.4 La descripción del circuito y sus módulos en un lenguaje de descripción de hardware

Dentro de la metodología de diseño establecida en el presente trabajo, la descripción del circuito se realiza mediante VHDL y LPMs. En la presente sección se revisan las ventajas y limitaciones del uso de dicho lenguaje, igualmente se define y considera las ventajas del uso de LPMs. Posteriormente se describe brevemente la implementación de los bloques constitutivos señalando los LPMs utilizados.

4.4.1 Utilización del lenguaje VHDL para circuitos asíncronos

El empleo del lenguaje VHDL como soporte del diseño asíncrono es materia de investigación. En [17] reportan varios desarrollos de donde se puede establecer algunas de las ventajas del lenguaje VHDL:

- VHDL es un estándar IEEE de uso industrial en el diseño de sistemas digitales.
- Los modelos escritos en VHDL son fácilmente simulados y verificados en cualquier entorno de diseño.
- Los módulos lógicos activados por eventos pueden ser modelados, compilados y puestos en la biblioteca VHDL de forma individual.
- VHDL sirve como interfaz hacia la integración con diversas herramientas comerciales, además de brindar facilidades de co-simulación, empleo de diferentes estilos de diseño y ofrecer alternativas de implementación sobre ASICs o FPGAs.

Las limitaciones de VHDL reportadas en [17] radican que en la descripción de alto nivel se pierde algunas características propias de los circuitos asíncronos. En el caso de modelar en VHDL el protocolo de comunicaciones asíncrono significa describir explícitamente de las señales de request, acknowledge y datos. Se intenta suplir las limitaciones añadiendo a VHDL común un conjunto de construcciones de lenguaje que soportan el diseño de circuitos asíncronos. Las sentencias VHDL son subdivididas en varios bloques, en cada bloque las sentencias son convertidas en circuitos ordinarios y estos son denotados mediante otro lenguaje como C++.

Una dificultad adicional es que las herramientas de place&route usualmente están separadas de la especificación VHDL de alto nivel y del proceso de síntesis. En presente trabajo se mantiene una descripción estructural que permite resolver el place&route, mediante las facilidades de compilador.

En el presente trabajo y dentro del marco de aplicación, se desarrolla una técnica de síntesis basándose en los modelos planteados. De forma general, se puede tener una descripción de alto nivel en VHDL, los circuitos de control se sintetizan de forma manual o mediante herramientas como PETRIFY. Considerando las especificaciones de la aplicación concreta se realiza la verificación de su correcto funcionamiento a lo largo de las diferentes etapas del diseño. La metodología de síntesis micropipeline para

circuitos asíncronos, permite explotar el paralelismo y concurrencia en el diseño para la aplicación concreta. A un bajo nivel se puede evaluar el consumo de potencia, área y velocidad de las posibles arquitecturas.

Específicamente, el estilo de diseño asíncrono micropipeline permite tener un buen balance entre área, velocidad y test para la aplicación desarrollada. En el presente trabajo a fin de tener una aproximación real a la implementación, el prototipo se plasma sobre FPGAs comerciales lo cual permite realizar una evaluación comparativa de las posibles arquitecturas asíncronas y verificar la correcta funcionalidad de la Unidad de Clasificación. También la implementación en los FPGAs se puede considerar como una aproximación útil antes de una implementación en un ASIC, lo cual es facilitado por el uso del lenguaje VHDL que permite el "mapping" sobre diferentes plataformas y evaluar las prestaciones de diferentes arquitecturas.

4.4.2 Implementación mediante Módulos Parametrizables (LPMs)

Los LPMs (Library of Parameterized Modules) fueron propuestos en 1990 y es una extensión del formato estándar EDIF [44], la descripción VHDL mediante LPMs de los diseños asíncronos otorga una característica fundamental de la metodología del presente trabajo. Permite lograr el objetivo de tener independencia tecnológica sin perder, o por lo menos minimizar, la pérdida de eficiencia en la relación área/velocidad.

Altera introdujo los LPMs en 1993 y ha mantenido un desarrollo constante junto con sus FPGAs. Aunque sin un desarrollo similar al de Altera, desde 1995 los LPMs son soportados por los mayores proveedores de herramientas EDA como Cadence, Mentor Graphics y Viewlogic. Una facilidad equivalente a LPMs en Xilinx son los LogiBLOX y Core Generator tools. Donde se tiene una GUI (graphical user interface) para los LogicBLOXs que genera un modelo VHDL de comportamiento para cada módulo LogiBLOX creado, por lo cual la simulación puede ser realizada de forma independiente de la implementación.

Por otro lado, la tendencia actual de diseño re-usable exige minimizar la pérdida de eficiencia. En [43] se muestra que los diseños re-usables a través de VHDL son aproximadamente hasta un 41% mayores en lógica adicional y tienen hasta un 17% mayor de retardo que los diseños no-reusables orientados a una tecnología concreta. En este sentido, en la medida que las compañías de herramientas CAD comerciales desarrollen LPMs, estos pueden considerarse como una interfaz para un diseño independiente de la tecnología ya sea ASICs o FPGAs.

En relación con la presente aplicación, en las implementaciones reportadas en la sección 2.4 se observa una deficiencia general de los diseños asíncronos respecto a los diseños síncronos, la cual es de necesitar una mayor área y dependiendo del estilo de diseño presentan una mayor o menor degradación de la velocidad, además, tienden a emplear recursos normalmente escasos en un CI (doble número de líneas de conexiones, mayor número de pines I/O, etc.).

De todo lo anterior, como las ventajas de los LPMs se pueden resumir en:

- Permite una entrada de diseño independiente de la tecnología, ya que no es necesario especificar de antemano la tecnología de implementación.
- Eficiente diseño físico debido a que el "mapping" tecnológico es especificado por el suministrador de la tecnología.
- Dota de una entrada de diseño independiente de las herramientas EDA, los LPMs permiten migrar los diseños entre diferentes proveedores EDA y mantener la descripción de alto nivel de las funciones.
- Permite especificar completamente el diseño, el conjunto módulos de LPMs es suficiente para especificar cualquier diseño y también para cualquier función no incluida puede ser creada en forma aparte de los módulos.

Descripción VHDL-LPMs de los bloques.

En el presente trabajo se implementan todos los bloques constitutivos mediante LPMs. Se usa una estructura jerárquica, usando funciones parametrizadas (LPMs) para los circuitos de control y para los bloques de procesamiento. Los LPMs para los circuitos de control permiten implementaciones con un número mínimo de LEs, por ejemplo el elemento-C ocupa un solo LE. Para mantener un diseño flexible para los bloques de procesamiento se tienen sentencias tipo GENERIC para definir el número de bits de los vectores que se procesan y definir parámetros de las funciones LPM.

Como una muestra de anterior, en las descripciones se emplean funciones LPMs sencillas como por ejemplo:

- Elemento-C: Se implementa mediante un lpm_add_sub.
- Latch: Se implementa mediante dos lpm_latch.
- Bloques de Comparación: Se implementa mediante un lpm_compare y un lpm_ mux.

En la sección 4.2.3, se detalla la implementación de los circuitos de control. En las secciones 5.3 y 5.4 se muestran los detalles de la implementación para los circuitos de clasificación. En la biblioteca de módulos se tienen los listados de las descripciones VHDL-LPM.

4.5 Verificación funcional del diseño micropipeline

Una deficiencia de los diseños asíncronos en general es el poco desarrollo de las técnicas de test. En la sección 2.2.3 se describe las dificultades y también las características de los micropipelines que pueden facilitar dicho test.

En el presente trabajo la verificación de funcionamiento de los módulos de control y bloques de procesamiento se realiza mediante simulaciones. Se simula cada módulo de forma independiente luego también integrado en el circuito de clasificación serie y paralelo. Para bloques de procesamiento, en el capitulo V se describen posibles estrategias de test.

A continuación se establece el método para la verificación funcional del diseño micropipeline, tomando en cuenta las principales especificaciones de la aplicación como el tiempo de retardo total, los niveles de prioridad y el número de células por cola. En la

implementación sobre FPGAs se demuestra, mediante las mediciones experimentales, que las simulaciones respectivas permiten diseñar correctamente los circuitos del micropipeline.

Metodología de verificación.

El concepto de verificación funcional implica demostrar que el circuito implementado cumple correctamente la función de la Unidad de Clasificación para algoritmo de QoS expuesto en la sección 3.4.3.

El criterio para la verificación funcional se puede enunciar de forma general que de acuerdo con el área de aplicación, los parámetros de los circuitos de comunicaciones pueden ser caracterizados dentro de ciertos rangos y modelos funcionales típicos, ya sea mediante especificaciones, funciones de transferencia o algoritmos. Una vez establecido el modelo funcional puede ser asociado con arquitecturas ya sea de tipo comportamiento o de tipo estructural y en los cuales sea posible confirmar su equivalencia. La equivalencia entre ambas arquitecturas del subsistema, es decir, de comportamiento y estructural se confirma a través de las simulaciones.

En este caso concretamente se necesita corroborar que la Unidad de Clasificación implementada satisface las especificaciones dadas en la sección 3.4.4, y de las cuales las especificaciones estrechamente ligadas a la Unidad de Clasificación son:

- El tiempo de retardo total.
- Niveles de prioridad.
- Células por cola.

Para estas especificaciones en la Tabla 6.7 se resume los valores teóricos y los resultados de la implementación en la plataforma FPGA indicada.

Las otras especificaciones como el índice de utilización, la capacidad de células, probabilidad de pérdida de células, el número de puertos, están relacionadas a un nivel del conmutador y con el modelo de tráfico ATM. Por lo cual implica una verificación a escala global que no es el objetivo del presente trabajo y es un trabajo a considerar en el futuro, en todo caso se tiene la garantía que el tipo de conmutador y el modelo de tráfico han sido lo suficientemente estudiados en las referencias citadas en la sección 3.4.3.

Verificación del retardo total.

Implica verificar que el circuito Unidad de Clasificación tiene un retardo que está de dentro del rango del retardo teórico dado por el modelo de tráfico y la norma considerada. En la sección 3.4.3 se derivan los valores teóricos de retardo total para las normas OC-48 y OC-12. Obviamente el retardo total depende de la tecnología de implementación y la verificación se establece a través de simulaciones. En el presente trabajo se usa como plataforma de implementación los FPGAs disponibles de Altera y Xilinx.

Estos retardos son corroborados mediante simulación y posteriormente se realizan las mediciones experimentales sobre los prototipos FPGAs, los resultados permiten demostrar que la simulación es lo suficientemente aproximada a la implementación

física. En los capítulos V y VI se detallan dichos resultados de las simulaciones y las mediciones de laboratorio sobre los FPGAs, donde se destaca que para un diseño micropipeline asíncrono se puede verificar de forma sencilla el throughput de operación y que su valor depende de la tecnología particular.

Verificación de los niveles de prioridad.

Los niveles de prioridad son establecidos por la estrategia de administración de la QoS, depende de los tipos de clases de tráfico y también puede considerarse que se incluye el puerto de destino. El número de niveles de prioridad depende del número de bits de los vectores de prioridad. Un mayor número de bits de prioridad implica un mayor tiempo de procesamiento y por lo tanto aumenta el tiempo de retardo total de la Unidad de Clasificación.

En la sección 3.4.4 se fijan los niveles de prioridad para las normas OC-48 y OC-12. Igualmente en los capítulos V y VI se detallan dichos resultados de las simulaciones que verifican que se cumplen dichas especificaciones de prioridad sobre los FPGAs disponibles de Altera y Xilinx.

Células por cola.

Esta especificación está ligada a la capacidad total de células y depende del número de puertos del conmutador. Este número de células por cola debe permitir satisfacer ambos requisitos de capacidad total de células y de retardo total de la Unidad de Clasificación. En la sección se fijan el número de células por cola para las normas OC-48 y OC-12.

Como se explica en la sección 3.4.3, en el circuito Unidad de Clasificación el número de células por cola implica el uso de un buffer que mantiene la dirección de memoria donde se guarda la célula. Por lo cual no se realiza ningún procesamiento con dicho vector y el retardo de un buffer siempre será menor que el retardo de procesamiento del vector de prioridades. Luego es claro que no añade ningún retardo adicional.

Igualmente en los capítulos V y VI se detallan dichos resultados de las simulaciones sobre los FPGAs disponibles de Altera y Xilinx que verifican que se cumplen dichas especificaciones del número de células por cola.

CAPÍTULO V

ANÁLISIS DE LAS ARQUITECTURAS ASÍNCRONAS QoS-ATM

5.1 Introducción

El presente capítulo revisa los principales tipos de circuitos de clasificación candidatos para la implementación de la Unidad de Clasificación, se selecciona los circuitos adecuados y se establecen los requisitos para una implementación asíncrona. Se continúa con la especificación las arquitecturas paralela y serie basados respectivamente en el clasificador bitónico y el clasificador por bloques, adicionalmente se esboza el diseño para testabilidad (DFT) de dichos circuitos constitutivos.

A continuación de lo mencionado arriba, se aborda en detalle el diseño de la Unidad de Clasificación y de acuerdo a las especificaciones de las normas ATM se tratan aspectos de las interfaces externas y diseño global dentro de un conmutador ATM. Posteriormente se formaliza el análisis de rendimiento de la arquitectura micropipeline y se revisan las ventajas y limitaciones respecto a otras arquitecturas clásicas.

5.2 Tipos de arquitecturas posibles

En la presente sección se revisan brevemente los principales tipos de circuitos de clasificación que pueden servir para la implementación de la Unidad de Clasificación, por las ventajas evidentes se hace hincapié en los tipos de circuitos denominados clasificador por bloques y el clasificador bitónico. A continuación, considerando la aplicación se establecen los requisitos de implementación asíncrona de la Unidad de Clasificación.

5.2.1 Conceptos generales de circuitos de clasificación

En el campo de los sistemas digitales clásicos síncronos, la clasificación de una secuencia de vectores (sorting) es uno de los problemas mejor estudiados, entre otras razones debido a muchos algoritmos requieren clasificar datos y de esta manera procesar datos que son generalmente de naturaleza aleatoria.

Se han desarrollado diversas técnicas de clasificación por hardware que intentan reducir el tiempo y los circuitos requeridos, por ejemplo se tiene la red de clasificación de Batcher [45] y el clasificador de matriz sistólica “heap” [46]. Existen soluciones no escalables, por ejemplo para un número de pequeño prioridades se puede mantener un

FIFO para cada prioridad de los vectores [47], de forma que la clasificación es solo una operación de múltiplex. Si se tiene un amplio rango de prioridades pero pequeño número de vectores, se puede usar una estructura de búsqueda paralela mediante una CAM (content addressable memory). Cuando se requiere rápidas inserciones pero extracciones lentas, puede ser suficiente una cola de prioridad tipo bloques [48], los vectores insertados y son comparados con el valor mínimo actual y el resultado es almacenado en un buffer. La extracción selecciona el mínimo actual y se realiza una búsqueda del buffer para encontrar el nuevo mínimo, puede ser de forma serie o con algún grado de concurrencia de las operaciones.

Se tienen algoritmos más elaborados como el clasificador denominado "rebound" [49], se necesita (N-1) comparadores para clasificar N vectores. El vector de entrada tiene un campo de datos y un campo de prioridad, aunque es necesario que primero ingresen los bits de prioridad y en un subsiguiente ciclo ingresen los datos. En el algoritmo up/down [50], se requiere N/2 comparadores para clasificar N vectores y el tiempo de clasificación es una función O(N), los elementos de clasificación son más complejos y el rendimiento decrece cuando aumenta la longitud de los comparadores.

Por otro lado, la clasificación en paralelo permite un alto rendimiento mediante la utilización de múltiples bloques funcionales operando concurrentemente. Los circuitos de clasificación denominados "odd-even" y "bitonic" [53], tienen la capacidad de clasificación clasificar N vectores en un tiempo que es una función $O(\log_2 N)$ y necesita un número de comparadores que es una función $O(N \log_2 N)$, por su simplicidad se consideran muy prácticos de implementar.

Adicionalmente se tiene soluciones sofisticadas como los sistemas de bus configurable denominados "shift switches" [51], que permiten resolver algoritmos de clasificación. Se han realizado implementaciones de circuitos de clasificación donde el diseño se organiza como una malla reconfigurable de tipo "shift switching", la dimensión de la malla se puede adaptar a cualquier dimensión dependiendo de los recursos de hardware y de la velocidad de clasificación requerida. Una desventaja es la mayor área necesaria y que para dimensiones mayores se degrada rápidamente debido a que requiere $O(N^2)$ comparadores.

Otro tipo de solución propuesta es el clasificador periódico, se considera como una red de comparadores que son usados repetidamente mientras que la salida no este ordenada y donde la salida es realimentada en la red. Por ejemplo en [52] se propone tiene red con periodo (log N) que clasifica N vectores en un tiempo $(\log^2 N)$. El problema radica en el diseño del comparador periódico que debe tener una alta velocidad independiente aún para el caso de un periodo pequeño, igualmente el diseño necesita una gran área de layout.

5.2.2 Circuito de clasificación paralelo tipo Bitónico

La red de clasificación tipo bitónico se basa en un modelo de red de comparación, en el cual se tiene operaciones de comparación y permutación que se realizan en paralelo. El elemento principal es el denominado comparador 2x2, las dos entradas son comparadas y las salidas tienen un orden ascendente o un orden descendente. Un "bitonic merger" es una red de comparación que acepta una secuencia bitónica como entrada y produce de

salida una secuencia clasificada monotónica. Se puede considerar el comparador 2x2 como el menor "bitonic merger".

Un comparador bitónico de clasificación para una secuencia arbitraria, es construido mediante varios "bitonic merger" usando una regla iterativa [53]. El tiempo de clasificación sigue la función $O(\log_2 N)$ y número de comparadores la función $O(N\log_2 N)$. En general, para N vectores de M bits cada vector y para el clasificador tipo bitónico se observa:

- Se tiene una concatenación de secuencias bitónicas de $\log_2 N$ etapas, donde cada etapa i requiere i pasos.
- Se tiene una profundidad de niveles o pasos de $(\log^2 N + \log N)/2$ que es una medida de la concurrencia pipeline.
- Cada nivel consiste de N/2 módulos 2x2 de comparación y de permutación, cada uno procesa dos vectores de M bits.
- Se necesita $(\log^2 N + \log N)\ N/2$ comparadores de M bits.
- El retraso total T es proporcional a $M\ \delta\ (\log^2 N + \log N)/2$, donde M es número de bits de los vectores, δ representa el ciclo de pipeline que se calcula como el retraso de la comparación/permutación de un bit añadido a los retardos de interconexión.

5.2.3 Circuito de clasificación serie tipo Bloques

Un clasificador de bloques (block sorter) [54] [55], tiene un solo puerto de entrada que acepta en serie un conjunto de vectores y tiene un solo puerto de salida de los vectores de la misma forma. Se asume que el flujo de vectores de entrada es dividido en bloques de longitud fija. Por lo cual los vectores se presentan agrupados en bloques de longitud N, N>1, y la salida es el mismo bloque clasificado en orden descendente. El circuito de ordenación tipo bloques tiene un tiempo de clasificación constante y el máximo throughput se tiene cuando la operación de cada bloque básico tiene la máxima operación de tasa constante y por lo cual la latencia depende del número de bloques básicos.

La tasa de clasificación puede mantenerse constante por lo cual es apropiado para una solución micropipeline, el bloque de clasificación es realizado mediante una matriz de (N-1) bloques básicos. La matriz clasifica, cada bloque de vectores con valores, mediante la transferencia de los valores menores hacia la salida de la matriz. Cada bloque básico amplia la matriz hacia la salida en una unidad adicional. Una vez que el bloque ha pasado la matriz, los (N-1) valores están en su posición final y se encuentran clasificados en orden descendente, lo cual significa que en primer lugar salen por el puerto de salida los vectores con los valores menores.

5.2.4 Especificaciones de circuito de clasificación

Los circuitos de clasificación reseñados anteriormente se han orientado a una implementación síncrona, por lo cual la equivalente implementación asíncrona debe satisfacer algunos requisitos adicionales o dar mayor peso a otros que resultan obvios en la implementación síncrona. Se puede establecer dos grupos de requisitos, aunque muy relacionados, un grupo debido a la implementación y otro grupo debido a la aplicación.

El diseño de Unidad de Clasificación toma en cuenta dichos requisitos y se muestran a continuación.

Requisitos debidos a la implementación circuital.

- El circuito debe mantener un balance entre la velocidad y costo de área, los circuitos resultantes deben minimizar el área y tener una estructura geométrica que facilite el layout micropipeline. Como figura de mérito de la implementación se puede considerar el producto del tiempo total de clasificación por número de comparadores del circuito. Usualmente el parámetro que toma en cuenta el grado de concurrencia pipeline y que indica el compromiso velocidad y área es el producto AT^2, donde se considera el área y el cuadrado del tiempo total de clasificación.

- Debe ser construido por repetición de un bloque básico. Para la implementación del circuito serie se debe repetir N veces el bloque básico y para el circuito paralelo basándose en una unidad básica. Esto facilita el layout y facilita la característica de poder ser ampliable.

- Un requisito del rendimiento para el circuito clasificador es que ambas tasas de operación de entrada y de salida deben ser independientes del número de vectores a clasificar, ya que el arribo de los N vectores es forma asíncrona.

- El circuito debe poder operar para igual número de extracciones y de inserciones de vectores, ya que para la administración de los vectores se consideran ventanas de tiempo de duración fija (tiempo de célula). Como una variante y ventaja adicional se requieren un máximo de extracciones de vectores en cada ventana de tiempo.

- Los vectores con prioridades iguales deben ser puestas en la salida en el mismo orden FIFO como fueron ingresadas. Esta especificación es útil para operaciones de QoS-ATM. Las células ATM de igual prioridad requieren mantener el mismo orden de arribo.

Requisitos debidos a la aplicación.

- La arquitectura debe ser ampliable desde el punto de vista de tamaño del buffer, de número de puertos de entrada y de ancho de banda. Además, debe poder administrar un gran rango de clases de servicio debido a que las células son conmutadas de acuerdo a su destino y prioridad. Cada célula de entrada es asociada a un puerto de salida, este proceso de asociación se efectúa en paralelo para cada puerto de salida. Las arquitecturas deben reducir el problema de sobrecarga de los buses de comparación.

- La clasificación debe hacerse en tiempo real, lo cual permita que la red pueda ser usada para aplicaciones en tiempo real y transmitir inmediatamente células de alta prioridad. Se pueden llegar a tener decenas de miles de prioridades para permitir una mayor flexibilidad en la planificación de la QoS en la red. Para ofrecer garantías de servicio se debe evitar reservar los recursos para las condiciones de peor caso, ya que normalmente resulta en una infra-utilización de recursos.

- Se debe tener un mecanismo de clasificación alta velocidad pero sin un alto costo de mantenimiento. Aunque para mejorar el rendimiento de la red es necesario un mayor número de entradas lo que significa establecer más rutas de conexión y mayor costo de mantenimiento. La tendencia de las redes de alta velocidad está marcada por necesitar un gran número de niveles de prioridad, una gran capacidad del buffer y un mayor número de puertos de entrada.

5.3 Arquitectura tipo Paralela

En la presente sección se especifica la arquitectura paralela basándose en el clasificador bitónico, se establecen las condiciones de operación para satisfacer las normas de tráfico ATM OC-12 y OC-48 y se detalla la implementación de los componentes. Para estimar el área de cada componente se considera el número de LEs (logic cell) que requiere dicha implementación. Adicionalmente se describe el posible diseño para testabilidad (DFT) del circuito mencionado.

5.3.1 Diseño del circuito

El circuito de clasificación paralelo es del tipo bitónico, en la Figura 5.1 se tiene el diagrama de bloques de una red bitónica de clasificación para N= 8 vectores y los módulos de comparación 2x2. Se indica los niveles de micropipeline los cuales coinciden con el número de los niveles o pasos del circuito.

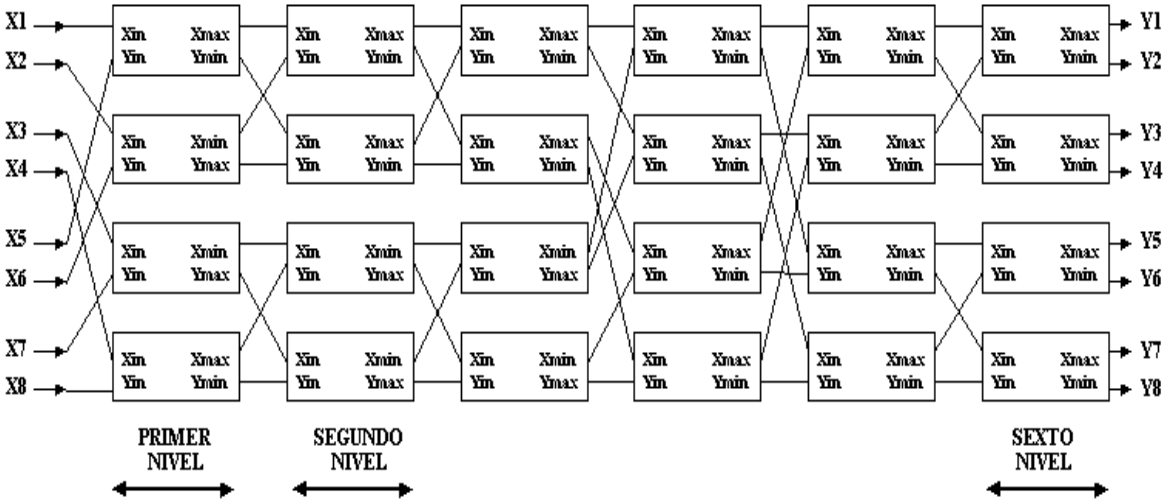

Fig. 5.1: Red bitónica de clasificación para N= 8 vectores

Se tiene las siguientes especificaciones de acuerdo a las relaciones dadas en la sección 5.2.2, para $N=8 = 2^3$ y cada vector de M bits:

- Profundidad de niveles o pasos: $(\log^2 N + \log N)/2 = 6$
- Número de módulos de comparación 2x2 en cada nivel: $N/2 = 4$
- Número de comparadores: $N/2 \times (\log^2 N + \log N) = 24$
- El retraso total: $T \sim M \times \delta \times (\log^2 N + \log N)/2 = 6 (M \times \delta)$

Consta de 6 niveles de micropipeline, 8 vectores de entrada en paralelo, el número M bits en cada vector depende de la norma. Para OC-12 los vectores tienen 25= (16 + 9) bits, 16 son para las prioridades y 9 bits para las direcciones de memoria y para la norma OC-48 los vectores tienen 16= (8 + 8) bits, 8 bits para las prioridades y 8 bits para las direcciones de memoria.

Los circuito-base son los módulos de comparación 2x2, son similares en cuanto a su función, comparan los bits de prioridad de cada nuevo par de vectores de entrada en cada etapa de micropipeline. Los pares de vectores son permutados y comparados a lo largo del micropipeline siguiendo el algoritmo de clasificación bitónico. Finalmente, los 8 vectores son clasificados en paralelo en orden descendente, donde se define que a la salida se tienen en las posiciones superiores los vectores con la menor prioridad (cuyos valores binarios son mayores) así sucesivamente hasta los vectores de mayor prioridad (cuyos valores binarios son menores).

En la Figura 5.2 se tiene el diagrama de bloques del circuito-base de comparación 2x2 y a continuación se describe su operación.

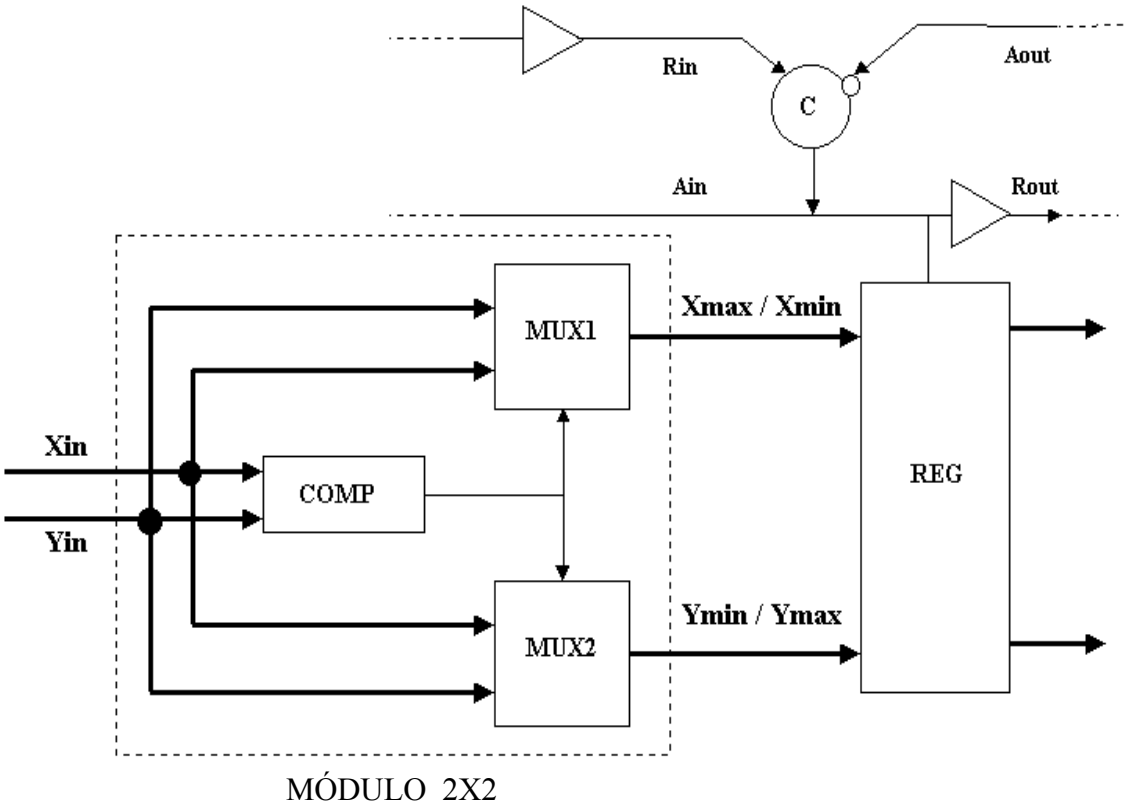

Fig. 5.2: Diagrama del circuito-base de comparación 2x2

Cuando un nuevo par de vectores ingresa son comparados en COMP y dependiendo de sus valores son transferidos mediante los MUX1 y MUX2 a la posición superior o inferior respectivamente. Se tienen dos tipos de circuito-base que difieren solo en dicha posición final de Xmax y Xmin. Cada circuito-base tiene la misma complejidad y profundidad lógica, luego de un tiempo suficiente para finalizar el procesamiento, los resultados son capturados en los bancos REG del micropipeline, en cada nivel tiene 4

bloques circuito-base y en las salidas el conexionado es modificado según el algoritmo. Este proceso se repite en los 6 niveles de micropipeline hasta la salida final donde se tienen los 8 vectores clasificados según su prioridad.

Para el bloque circuito-base se tienen los siguientes componentes principales que se describen a continuación. En capítulo VI se detallan las características de cada componente en cuanto a los niveles de lógica y número elementos lógicos ocupadas para la implementación en Altera y Xilinx. En la biblioteca de módulos se dan los listados VHDL respectivos.

Comparador (COMP).

Se implementa mediante un LPM_COMPARE, que permite establecer en los puertos el número de bits de cada vector (16 bits ó 8 bits) y la función de salida de comparación (ageb, CMP1= 1 si dataa[ ] > = datab[ ]). La salida de COMP controla los multiplexores MUX1 y MUX2. Área ocupada por bit en cada comparador: 0.5 LE.

Multiplexor (MUX1, MUX2).

Se implementan mediante LPM_MUX, igualmente se programa los puertos de entrada y salida necesarios de acuerdo la longitud de los vectores. La salida es vector que satisface la operación descrita anteriormente para cada bloque circuito-base. Área ocupada por bit en cada MUX: 0.5 LE.

Registros (REG).

Se implementa mediante dos LPM_LATCH, operan de manera asíncrona y se activan en cada flanco de la señal de control. Se programan los bits de cada longitud de vector y una configuración que permite la correcta operación. Área ocupada por bit en cada registro: 3 LEs.

Retraso (Δ).

Es un bloque que retarda la señal Rin para permitir la operación correcta de etapa de micropipeline, considerando el retraso del bloque circuito-base y retraso de REG. Su cálculo se realiza basándose en los retrasos requeridos en la implementación de los componentes. El método de cálculo se da en la sección 4.3.2. Área ocupada por cada unidad de retraso: 1 LE

Elemento-C.

Sirven para el control de la transferencia de los datos entre las sucesivas etapas del micropipeline. Se tienen igual número de puertas elemento-C que el número de etapas de micropipeline. En la sección 4.2.3 se detalla su especificación e implementación. Área ocupada por cada elemento-C: 1 LE.

Buffers.

Los bits de dirección de memoria obviamente no son comparados y son transferidos mediante buffers junto con los bits de prioridad. Los buffers son implementados igual

que los registros REG, mediante dos LPM_LATCH, operan de manera asíncrona y se activan en cada flanco de la señal de control. No se muestran en las figuras anteriores a fin de una mejor claridad. Área ocupada por bit en cada registro: 3 LEs.

Se programa la longitud de los vectores, para la norma OC-12 se tiene 9 bits para las direcciones de memoria y para la norma OC-48 se tiene 8 bits para las direcciones de memoria. Los buffers no afectan el throughput del circuito de clasificación pero implican un área adicional.

5.3.2 Consideraciones del test

El diseño para testabilidad (DFT) de los circuitos bitónicos clásicos ha sido ampliamente estudiado [56] [57]. Por otro lado, el test de la arquitectura micropipeline y como se explico en la sección 2.3.3, implica el uso de técnicas scan que igualmente es ampliamente conocida. El DFT es imprescindible en una implementación ASIC, para los FPGAs se pueden adaptar las técnicas estándar sugeridas por los propios fabricantes.

Para los bloques de procesamiento del circuito de clasificación paralelo bitónico micropipeline se puede adaptar y aplicar el método desarrollado en [57]. Es un método probabilístico para estimar la cobertura del test, donde se propone un esquema denominado detección de error concurrente (CED) para circuitos de clasificación paralelos. Dicho esquema, dependiendo del nivel de cobertura, tiene un área adicional de un 10% al 30%, se logra prácticamente el 100% de cobertura sin degradar el throughput.

5.4 Arquitectura tipo Serie

En la presente sección se especifica la arquitectura serie basándose en el clasificador de tipo bloques, se establecen las condiciones de operación para satisfacer las normas de tráfico ATM OC-12 y OC-48 y se detalla la implementación de los componentes. Para estimar el área de cada componente se considera el número de LEs (logic cell) que requiere dicha implementación. Adicionalmente se esboza el diseño para testabilidad (DFT) del circuito mencionado.

5.4.1 Diseño del circuito

En la Figura 5.3 se muestra la trama de vectores a clasificar, el cual tiene un formato que consiste de una cabecera o inicio, N vectores consecutivos y una cola o fin.

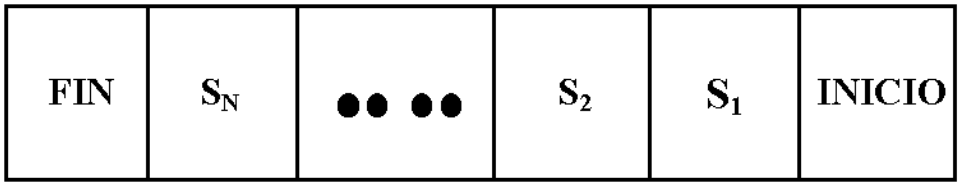

Fig. 5.3: Formato de entrada al clasificador serie

La cabecera es un vector fijo cuyos bits son todos ceros y significa el inicio del bloque, la cola es otro vector fijo cuyos bits son todos unos e indica el fin del bloque.

Todos los N vectores tienen el mismo número de bits. Aunque se pueden tener otros valores de acuerdo a los requerimientos, se considera que para la norma OC-12 los vectores tienen 25= (16 + 9) bits, 16 son para las prioridades y 9 bits para las direcciones de memoria y para la norma OC-48 los vectores tienen 16= (8 + 8) bits, 8 bits para las prioridades y 8 bits para las direcciones de memoria.

La cabecera permite iniciar un nuevo bloque de clasificación, la longitud máxima N del bloque a clasificar depende del número los bloques de circuito base. Los N bloques de circuito-base son iguales, están conectados en cadena, funcionan de manera asíncrona y operan de forma autónoma. Para la norma OC-12 se tienen N=30 vectores y para la norma OC-48 se tienen N=14 vectores. El esquema asíncrono permite que se inicie un nuevo bloque de clasificación en cuanto se requiera, los bloques de circuitos básicos no ocupados se consideran con valores de la más baja prioridad y pueden ser descartados al final del proceso de clasificación.

Cada bloque de circuito-base compara los bits de prioridad de cada nueva entrada con un valor almacenado en un registro dentro del bloque, luego cada bloque transfiere el vector menor a la salida y guarda el vector mayor en el registro interno. Finalmente, los N vectores son clasificados en orden descendente en los N bloques de circuito-base, a la salida se tienen en primer lugar los vectores con la más alta prioridad, cuyo valor binario es el menor, así sucesivamente hasta el vector de la menor prioridad, cuyo valor binario es el mayor.

En la Figura 5.4 se tiene el diagrama de bloques del circuito-base serie y a continuación se detalla su operación.

Cuando un nuevo bloque ingresa, la cabecera es detectada por el circuito DETEC, el vector en REG1 es puesto a la salida mediante el MUX, luego la cabecera es guardada en REG2. Para los siguientes vectores de entrada se comparan los bits de prioridad entre el vector en REG1y el vector que arriba, cuando el vector que llega tiene un valor menor (mayor prioridad) el comparador es CMP1= 1 y dicho vector es transferido a la salida mediante el MUX. Si el vector que arriba tiene un valor mayor (menor prioridad) entonces CMP= 0, el vector guardado en REG1 es enviado a la salida y el vector recién llegado es guardado en REG2, donde permanece hasta el siguiente ciclo que es copiado a REG1 para ser comparado. Cuando arriba la cola CMP= 0, se transfiere el vector en REG1 a la salida y la cola es guardada en REG2 hasta el inicio de un nuevo bloque de vectores a clasificar. Este proceso se repite para todos los bloques circuito-base.

A continuación se describen los componentes principales para el bloque circuito-base. En el capítulo VI se dan las características de cada componente en cuanto a los niveles de lógica y número células ocupadas para la implementación en Altera y Xilinx. En la biblioteca de módulos se dan los listados VHDL respectivos.

Comparador (CMP).

Se implementa mediante un LPM_COMPARE, que permite establecer en los puertos el número de bits de cada vector (16 bits ó 8 bits) y la función de salida de comparación

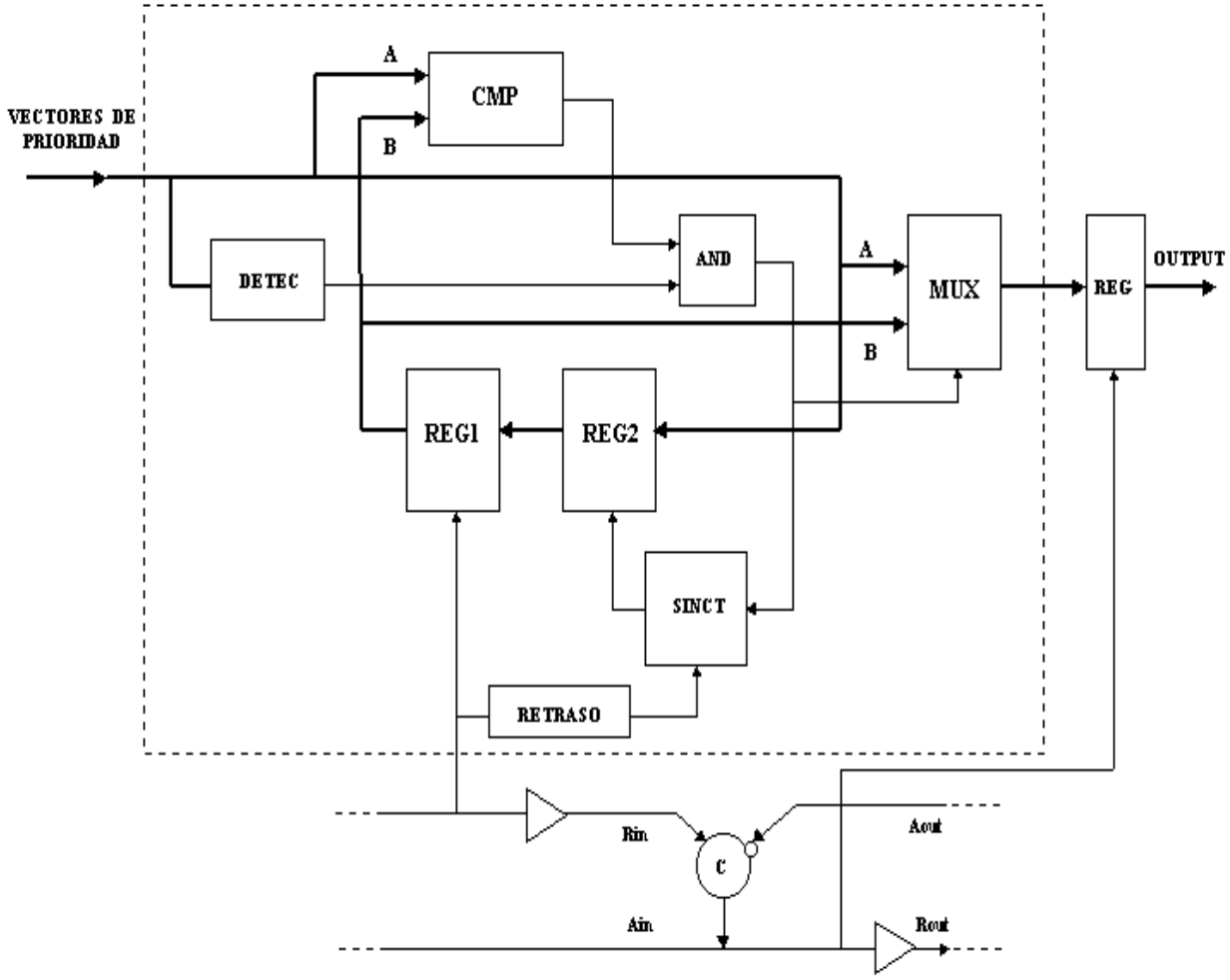

Fig. 5.4: Diagrama del circuito-base de comparación serie

(aleb, CMP1= 1 si dataa[..] <= datab[..]). La salida de CMP1 controla el multiplexor MUX1. Área ocupada por bit en cada comparador: 0.5 LE.

Multiplexor (MUX).

Se implementa mediante un LPM_MUX, igualmente se programa los puertos de entrada y salida necesarios de acuerdo la longitud de los vectores. La salida es vector que satisface la operación descrita anteriormente para cada bloque circuito-base. Área ocupada por bit en cada MUX: 0.5 LE.

Registros (REG, REG1 y REG2).

Se implementan mediante dos LPM_LATCH, operan de manera asíncrona y se activan en cada flanco de la señal de control. Se programan los bits de cada longitud de vector y una configuración que permite la correcta operación. Área ocupada por bit en cada registro: 3 LEs.

Detector (DETEC).

Permite detectar cuando ingresa la cabecera, su implementación puede ser una simple puerta OR si la cabecera es un vector el cual todos sus bits son ceros. Se implementa mediante un LPM_OR. Área ocupada: 3 LEs para 8 bits y 8 LEs para 16 bits.

Controlador (SINCT).

Permite el control carga de REG2, de acuerdo a las condiciones de los vectores que se comparan. Su implementación es mediante circuitos activados por flancos, es un circuito de control y su síntesis se detalla en la sección 4.2.3. Área ocupada: 3 LEs.

Retraso (Δ).

Es un bloque que retarda la señal Rin para permitir la correcta operación de cada etapa del micropipeline, considera el retraso del bloque circuito-base y el retardo de REG. Su cálculo se realiza basándose en los retrasos requeridos en la implementación de los componentes. El método de cálculo se da en la sección 4.3.2. Área ocupada por cada unidad de retraso: 1 LE.

Elemento-C.

Sirven para el control de la transferencia de los datos entre las sucesivas etapas del micropipeline. Se tienen igual número de puertas elemento-C que el número de etapas de micropipeline. En la sección 4.2.3 se detalla su especificación e implementación. Área ocupada por cada elemento-C: 1 LE.

Buffers.

Los bits de dirección de memoria obviamente no son comparados y son transferidos mediante buffers junto con los bits de prioridad. Los buffers son implementados igual que los registros REG, mediante dos LPM_LATCH, operan de manera asíncrona y se activan en cada flanco de la señal de control. No se muestran en las figuras anteriores a fin de una mejor claridad. Área ocupada por bit en cada registro: 3 LEs.

Igualmente se programa la longitud de los vectores, para la norma OC-12 se tiene 9 bits para las direcciones de memoria y para la norma OC-48 se tiene 8 bits para las direcciones de memoria. Los buffers no afectan el throughput del circuito de clasificación pero implican un área adicional.

5.4.2 Consideraciones del test

El diseño para testabilidad (DFT) de circuitos de clasificación de entrada y salida serie como el tipo de bloques ha sido bien estudiado [58], [59]. Igualmente son válidas para la arquitectura micropipeline las técnicas scan descrita en la sección 2.3.3. Para una implementación en FPGAs se tienen técnicas estándar y en el caso de una implementación ASIC se puede considerar un diseño DFT. Para el clasificador serie micropipeline tipo bloques se puede adaptar y aplicar el método desarrollado en [59] que a continuación se describe.

Para el test del path de datos, la testabilidad puede ser mejorada si hace que los estados de los registros dentro de los bloques de clasificación sean controlables. El test se implementa mediante una técnica BIST (built-in self test), el modo de operación es cambiado de forma asíncrona mediante un canal de dos fases que consiste de dos líneas una de Req y otra de Ack. El generador de los patrones de test aleatorios es implementado mediante un LFSR (linear feedback shift register). También por la propiedad de auto-test del micropipeline los fallos se detectan debido a que los bloques de clasificación detienen su operación

Dicho método reporta que el área adicional del BIST es 15.7%, se tienen 5 pines adicionales y se necesita aplicar alrededor de 1000 pares de vectores de test para detectar fallos en los bloques de clasificación.

5.5 Arquitectura de la Unidad de Clasificación QoS

En la presente sección se aborda el diseño global de la Unidad de Clasificación, dentro de la metodología de diseñó se configura la arquitectura micropipeline, se especifican los circuitos de control y de procesado de datos. A continuación se detallan las etapas de place&route y verificación de tiempos, a partir de lo cual se obtiene una óptima implementación de acuerdo a las especificaciones respectivas.

Un otro aspecto de la presente sección es considerar puntos adicionales importantes de la Unidad de Clasificación como las interfaces externas, sincronización y facilidades añadidas de operación. Se establece las normas observadas y que permitan una comparación con algunas arquitecturas clásicas.

5.5.1 Diseño del circuito

Una vez definidos y diseñados los bloques de procesamiento constitutivos de la Unidad de clasificación como se muestra en las secciones 5.3 y 5.4, se puede continuar con la metodología planteada en la sección 2.6.2 para los aspectos de síntesis del diseño micropipeline y el diseño físico (place&route). Los aspectos de simulación y verificación se consideran en la sección 5.6 y la implementación sobre las plataformas FPGAs se detallan en el capítulo VI.

Síntesis micropipeline.

En la Figura 5.5 se muestra el esquema de bloques del diseño micropipeline de la Unidad de Clasificación. Considerando los fundamentos del estilo de diseño micropipeline revisados en la sección 2.3.2 se tienen dos objetivos: 1) Definir los circuitos de control del micropipeline y 2) Definir el data-path del micropipeline.

1) Circuitos de control.

Se tienen puertas elemento-C como circuitos de control para cada etapa del circuito de clasificación serie y para cada etapa del circuito de clasificación paralelo. El número de circuitos de control elemento-C y los latches REG que controlan el micropipeline, depende de las especificaciones de las normas OC-12 y OC-48 dadas en la Tabla 3.2.

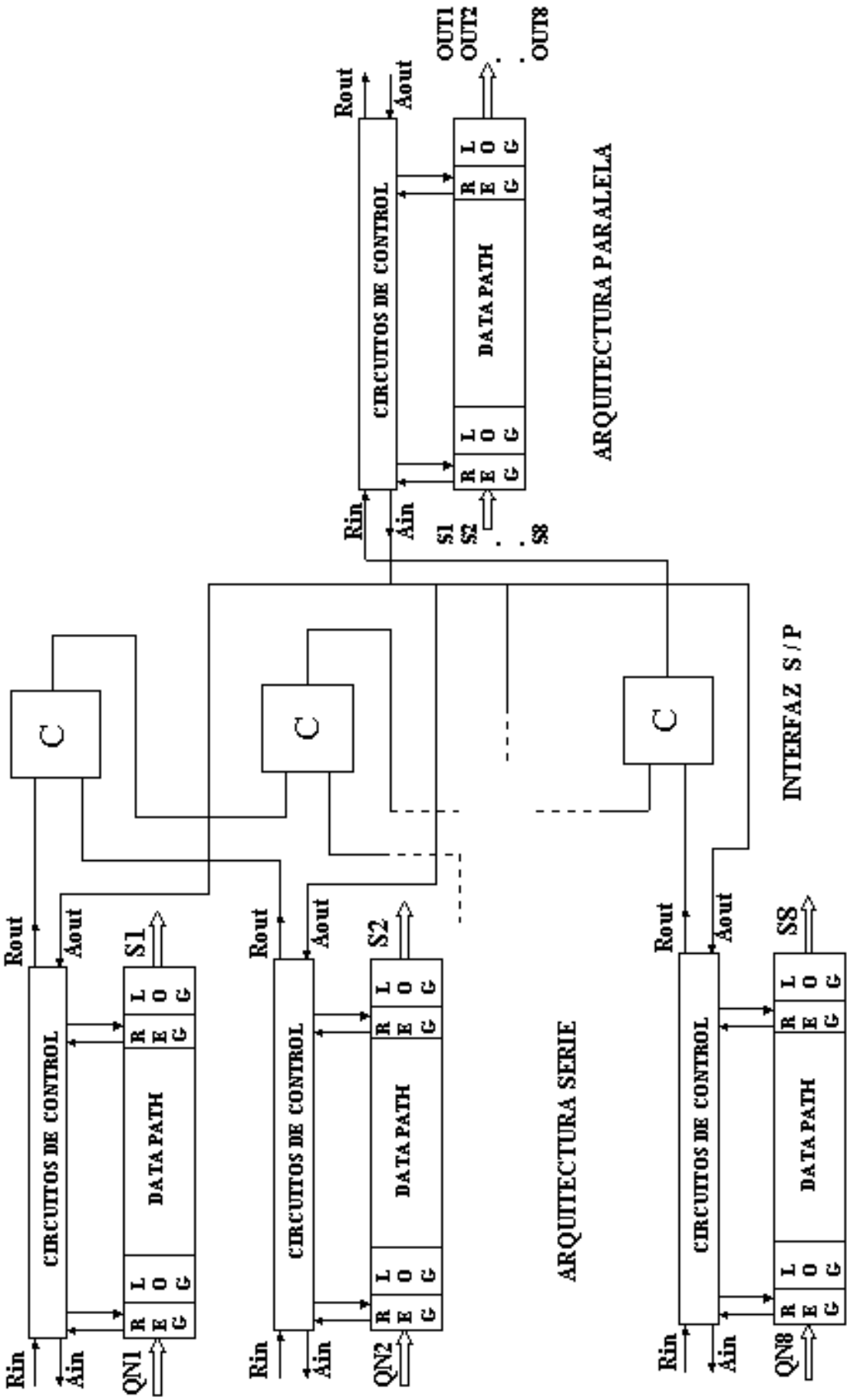

Fig. 5.5: Diseño micropipeline de la Unidad de Clasificación

Por otro lado, se tiene el circuito de control de la interfaz serie/paralelo, como se muestra en la Figura 5.5, tiene la finalidad principal es de enlazar el micropipeline del circuito serie con el micropipeline del circuito paralelo.

Se pueden tener diferentes circuitos de interfaz dependiendo del algoritmo QoS específico, además de las políticas de admisión ó rechazo de las células. También como se indicó en la sección 3.4.3, opcionalmente y dependiendo de la gestión QoS se puede tener un buffer de reingreso para células multicast o células de baja prioridad, dicho buffer no se muestra para una mejor claridad de la figura.

La mostrada es una interfaz simple que se puede basar en simples elemento-C o puertas elemento-C generalizado. Se construye como una cascada de dichas puertas. Se tiene 8 entradas que son las señales Rout del último bloque serie de cada cola, una vez que todas las señales Rout han sido activadas a la salida se activa la señal Rin para la primera etapa del circuito paralelo. De esta forma los 8 vectores de cada circuito serie son ingresados en el circuito paralelo para ser clasificados de acuerdo a sus prioridades. Una vez que se activa la señal Ain del circuito paralelo es transmitido a cada una de las entradas de la señal Aout de los circuitos serie y de esta forma continuar la operación micropipeline de la Unidad de Clasificación.

2) Data path.

Como se muestra en la Figura 5.5, el data-path del micropipeline está definido por bloques de procesamiento, tanto para el circuito serie y el circuito paralelo. Los latches REG permiten el desplazamiento de los datos a lo largo del data-path, la longitud de etapas del data-path y la longitud de bits de los latches REG dependen de las especificaciones de las normas OC-12 y OC-48 dadas en la Tabla 3.2.

Diseño físico (place&route).

Se mostraron respectivamente en las Figuras 5.2 y 5.4 una etapa cualquiera del micropipeline para ambas arquitecturas paralela y serie, en las cuales el diseño físico tiene por objetivo optimizar el área y la prestación del micropipeline. Se basa en simples recomendaciones que han sido definidas en la sección 4.2, además de los siguientes puntos adicionales:

- Para el data-path, es suficiente fijar la ubicación para etapa de los latches REG a lo largo del layout del diseño. El compilador intenta agrupar los demás componentes en un área alrededor del latch respectivo. Esto permite definir un área equipotencial correspondiente para dicha etapa.

- Para los bloques de retardos (matched-delays), es suficiente que estén agrupados formando una cadena lo más compacta posible y permite optimizar el retardo que introducen. Mediante directivas de cliques, RLOC se definen estas áreas y el compilador encuentra una ubicación cerca de la etapa que corresponden.

- Elección adecuada de los pines de entrada y salida de las señales globales de control del micropipeline (Rin, Ain, Aout, Rout). Esto permite minimizar el retardo de la señal Rin-Ain, Ain indica que los datos son capturados por el latch de entrada. También esta elección permite dirigir implícitamente al compilador, la ubicación

adecuada de los pines de entrada de las señales de datos. La elección de los pines Aout y Rout, igualmente tiene los mismos objetivos, minimizar el retraso Rout–Aout, y orientar al compilador la ubicación de los pines de salida de los datos. La señal global de reset puede ser un pin dedicado que permite una mejor distribución en todo el diseño, aproximándose a una derivación isócrona.

5.5.2 Consideraciones del diseño global e interfaces

Debido a que la conmutación ATM no forma parte de las normas ATM, las compañías de telecomunicaciones usan diferentes técnicas para construir sus conmutadores por lo cual existe una gran variedad de arquitecturas tanto en software y hardware, pero en un objetivo común es incrementar la velocidad y la capacidad eficiente de administrar la QoS.

En el presente trabajo, el diseño asíncrono de la Unidad de Clasificación para QoS utiliza el protocolo de señalización de handshake de 2 fases y tiene una arquitectura modular. Su integración e interfaces específicas con los otros subsistemas dependerán de la organización global del conmutador.

A continuación se revisan aspectos acerca de la organización del diseño global e interfaces estándar y otros puntos comunes en los conmutadores ATM.

Concepción global.

La incorporación de subsistemas asíncronos en el diseño global implica un sistema heterogéneo síncrono/asíncrono [60][61], donde la señal de reloj solo subsiste en aquellos módulos en los cuales sea imprescindible. En este escenario, el diseño global del conmutador ATM y de los diferentes subsistemas puede concebirse de diferentes formas y es un tema de investigación futuro.

Un sistema heterogéneo se basa en un esquema de comunicación de alta velocidad entre los diferentes módulos síncronos o asíncronos los cuales operan de forma independiente. El esquema de comunicación debe minimizar las fallas de sincronización sin sacrificar el throughput. Dentro de esta problemática, la sincronización es una función importante en un sistema digital heterogéneo ya que permite que dos unidades diferentes del sistema puedan trabajar en conjunto.

El problema de metaestabilidad [61][62], ocurre en la interacción de un sistema síncrono con una señal asíncrona externa o con otro sistema con un reloj diferente. La señal externa debe ser sincronizada al reloj del sistema por un circuito de sincronización. Una falla de sincronización en la interfaz de los módulos ocurre cuando los tiempos de llegada de una señal de transición externa y del flanco de activación del reloj no tienen la temporización correcta (tiempos de setup y hold). En este sentido un sistema asíncrono es apropiado para la interfaz con señales del mundo real ya que puede esperar hasta que se resuelva la metaestabilidad o en cualquier caso presenta mayor seguridad y eficiencia que un circuito síncrono.

Interfaces al nivel de red.

En la Figura 5.6 se muestra el concepto de interfaces al nivel de red de un conmutador. Un conmutador ATM puede ser usado por redes portadoras públicas y también dentro de redes privadas, se tiene dos formas distintas de ATM-UNI (User-Network Interface):

- UNI Pública. Puede ser usada para la interconexión un usuario ATM con un conmutador ATM asociado a un proveedor de red de servicio público. Tiene la misma especificación de capas ATM pero las facilidades de conexión de los usuarios deben ser capaces de abarcar grandes distancias.

- UNI Privada. Puede ser usada para la interconexión un usuario ATM con un conmutador ATM que es administrado como parte de la red privada. Tiene la misma especificación de capas ATM pero pueden estar localizadas en el mismo espacio donde esta ubicado el equipo del usuario (ATM user).

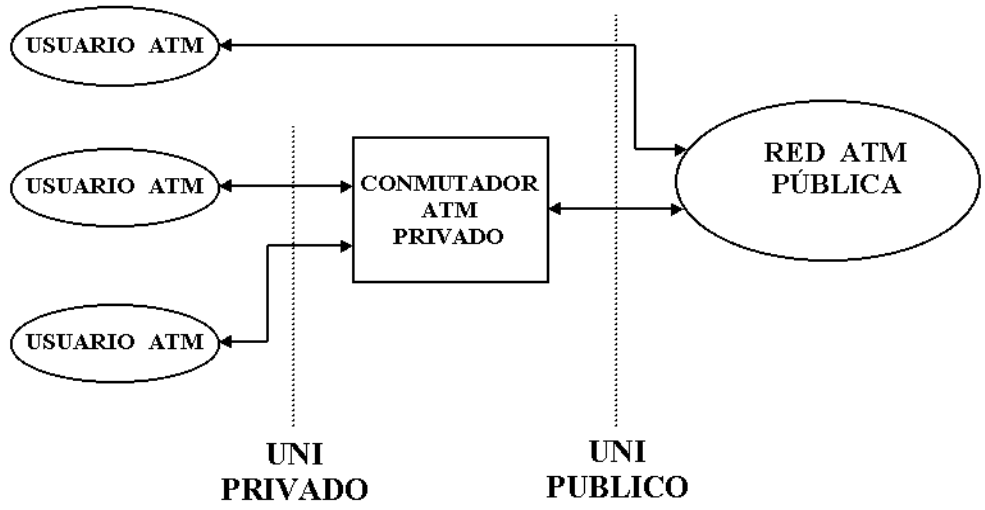

Fig. 5.6: Interfaces al nivel de red de un conmutador ATM

Un sistema ATM puede ser un conmutador o un adaptador de sistema final de usuario (end-system) que es un sistema de red que ejecuta AAL-5 o también capas altas de protocolo por ejemplo bridges, routers o interfaces de estaciones de trabajo. El equipo de usuario podría ser un sistema intermedio (IS) que compacta datos en células ATM y las envía, a través de un ATM UNI a un conmutador ya sea público o privado a su vez transfiere las células a otros dispositivos de usuario ATM mediante una red pública.

Las especificaciones UNI [30], involucran aquellos protocolos los cuales finalizan u operan en las interfaces del usuario de la red. Involucra a la capa física, a la capa ATM y a los protocolos de capas altas que se requieren para la administración de la UNI. Las especificaciones de la capa física consideran bit-ratios de 44.736 Mbps, 100 Mbps, 155.52 Mbps (OC-3), 622.08 Mbps (OC-12) y 2.5 Gbps (OC-48).

Arquitectura global del conmutador.

Las frecuencias de ATM exceden a la de cualquier procesador por lo cual es imposible un control en tiempo real, la potencia de procesamiento necesaria puede lograrse

mediante procesadores on-chip, por ejemplo ATMmizerII+ [63], que son usados para funciones especiales. Aparte de dicha consideración, en la Figura 5.7 se muestra un sistema general de conmutación ATM, el cual está formado por los principales bloques:

1) El bloque ALS (Atm Local switching Subsystem), está conformado por:

- SIM (Subcriber Interface Modules)
- LIM (Link Interface Modules)
- SCP (Subcriber Call Processor)
- ASNM (Access Switch Network Module). Consiste de elementos de conmutación tipo buffer compartido.

2) El bloque ACS (Atm Central Switching), realiza las interconexiones entre los ALS y está conformado por:

- LIM (Link Interface Modules)
- OMP (Operation and Maintenance Processor)
- ISNM (Interconnection Switch Network Module). Funciona también como un elemento de conmutación tipo buffer compartido.

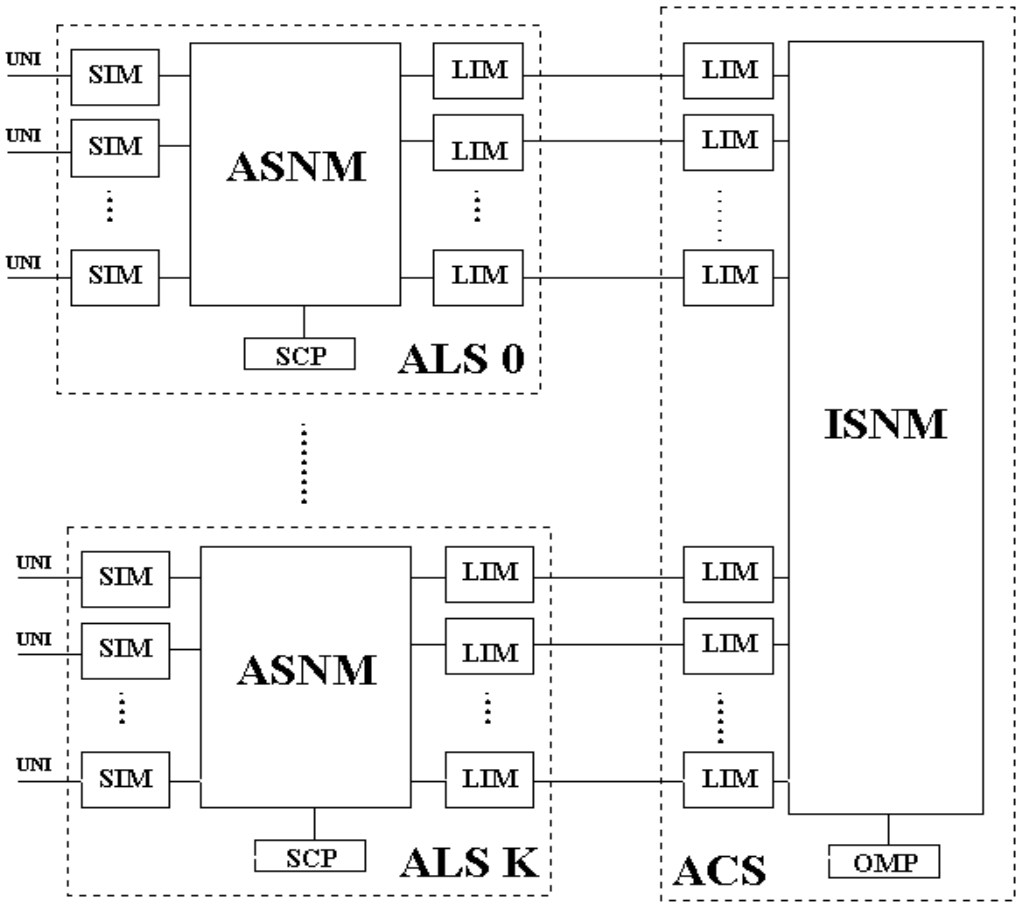

Fig. 5.7: Sistema general de conmutación ATM

Si una célula entrante en un ALS es destinada a otro puerto de salida en otro ALS, esta célula es conmutada a través del ACS hacia la correspondiente salida. Las células entrantes tipo multicast son replicadas mediante un mecanismo de partición y copia.

El bloque de conmutación ALS es equivalente al que se mostró en la Figura 3.4, tiene una arquitectura buffer compartido. Las células entrantes luego de un proceso de detección y alineación, son convertidas en bits de datos en paralelo para reducir la velocidad interna del bloque de conmutación. Luego las células son multiplexadas secuencialmente por el crossbar de entrada y cada célula es almacenada en la memoria buffer compartida tipo bit-slices (SBM).

La extracción de la cabecera delimita la célula que luego es enviada al bloque de Administración de QoS. Se ingresa la dirección SBM de la célula entrante en la Unidad de Clasificación junto con el valor de prioridad asociado a dicha célula suministrado por el procesador de prioridades.

El bloque Administración QoS tiene el circuito IAP (Idle Address Pool) que guarda las direcciones vacías del SBM y se encarga de proporcionar la dirección vacía (idle) para cada célula entrante. Las direcciones mantenidas en la Unidad de Clasificación son usadas para recuperar las células en la SBM y enviarlas al puerto de salida mediante el crossbar y el bloque de transmisión se encarga de la conversión paralelo/serie los bits de datos.

El sistema de conmutación ATM general, cuya estructura lógica fue descrita anteriormente, puede ser implementado como una arquitectura modular ampliable y re-usable, esto se muestra en la Figura 5.8, la cual puede ser usada como un conmutador o un "end-system" que ejecuta funciones de la capa AAL-5 o también funciones de capas más altas.

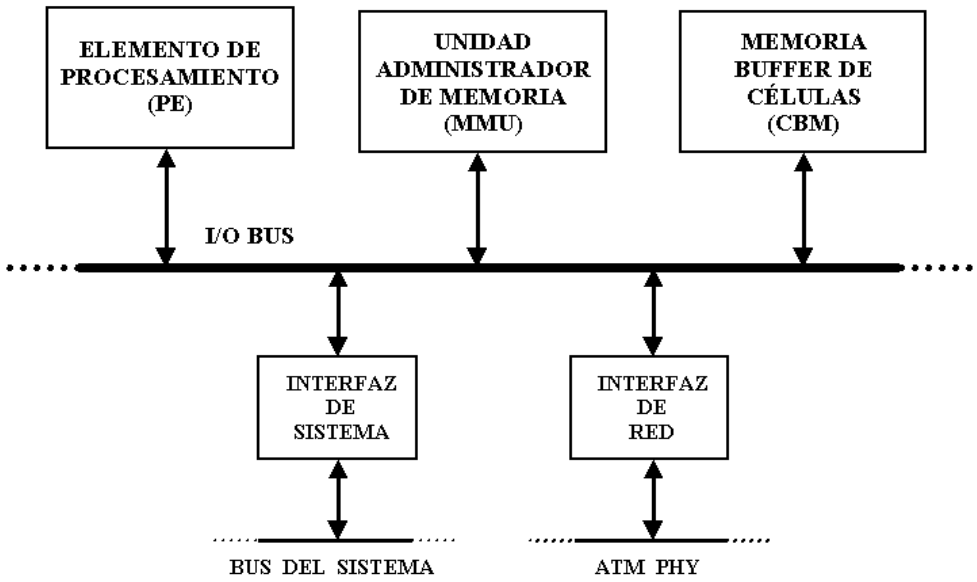

Fig. 5.8: Arquitectura modular de un conmutador ATM

En dicha arquitectura se agrupan los bloques de la arquitectura lógica y se tienen los siguientes bloques funcionales:

- Un subsistema de memoria compuesto por el subsistema MMU (memory manager unit) que administra las estructuras lógicas de datos que componen las células. Estas estructuras lógicas pueden ser colas relacionadas estructuralmente por pertenecer a un mismo paquete o asociadas operacionalmente ya que comparten un mismo recurso.

- Otro bloque es el CBM (cell buffer memory) que almacena las células ATM en el sistema y el bloque de administración de QoS (QoS-M) se encarga de la administración de colas y prioridades. Dentro de este bloque se enmarca el diseño del presente trabajo.

- El elemento de procesamiento (PE) que ejecuta para el sistema el apropiado protocolo de las capas superiores.

- Interfaces del sistema o enlaces son responsables de la recepción y transmisión células ATM sobre los enlaces físicos conectados al sistema.

Usualmente un puerto I/O puede ser configurado para operar como un bus PCI. En esta configuración permite que en lugar de un puerto normal, pueda ser un dispositivo "slave" y las células son escritas desde el ordenador hacia el conmutador. Como un dispositivo "master", el conmutador lee las células desde la memoria del ordenador mediante DMA (Direct Memory Access).

El principal protocolo de las normas ATM para la capa física es SONET/SDH (Synchronous Optical Network / Synchronous Digital Hierarchy) es un conjunto de normas aplicables a una interfaz de transmisión óptica, nació en los laboratorios BellCore. Fue la primera norma adoptada por ATMforum [30]. Adicionalmente se tienen otros protocolos para la capa física como la norma IEEE Std 1355 "HIC/HS" o UTOPÍA-2.

Otros aspectos relacionados a un conmutador ATM como las normas de interfaz física son ampliados en el anexo A.

5.6 Análisis de las prestaciones

En esta sección se establecen los conceptos del modelo de tiempos que permite formalizar el análisis de rendimiento de la arquitectura micropipeline, posteriormente se detallan las ventajas y limitaciones tomando como referencia algunas arquitecturas clásicas.

5.6.1 Formalización de las figuras de mérito

Para el diseño de la Unidad de Clasificación, la cual tiene una arquitectura micropipeline asíncrona, se establece la formalización del análisis del rendimiento en cuanto al throughput que se puede considerar como el principal parámetro.

La Figura 5.9 muestra el esquema general de la etapa (i) y la etapa (i +1) cualesquiera del micropipeline. Los retardos indicados se hacen referencia para formalizar las ecuaciones respectivas, en la notación t_z y T_z se refieren respectivamente al retardo mínimo y máximo del parámetro z.

Considerando el apropiado matched delay t_d, para una etapa (i) el tiempo de ciclo mínimo (P) en el micropipeline es el intervalo de tiempo mínimo entre las sucesivas transiciones locales que provienen de la señal de request (Rin):

$P= T_Q + T_{logic} + t_{su} + t_{AA}$

Donde T_Q es el retardo de salida de los latches REG, T_{logic} es el retardo del bloque de procesamiento, t_{su} es el set-up time de REG, t_{AA} es el retardo del elemento-C.

El retardo ($T_Q + T_{logic} + t_{su}$) es el mínimo periodo que equivale a un pipeline síncrono, y en cada etapa (i), el tiempo adicional (overhead) debido al micropipeline asíncrono es el retardo de un simple elemento-C (t_{AA}).

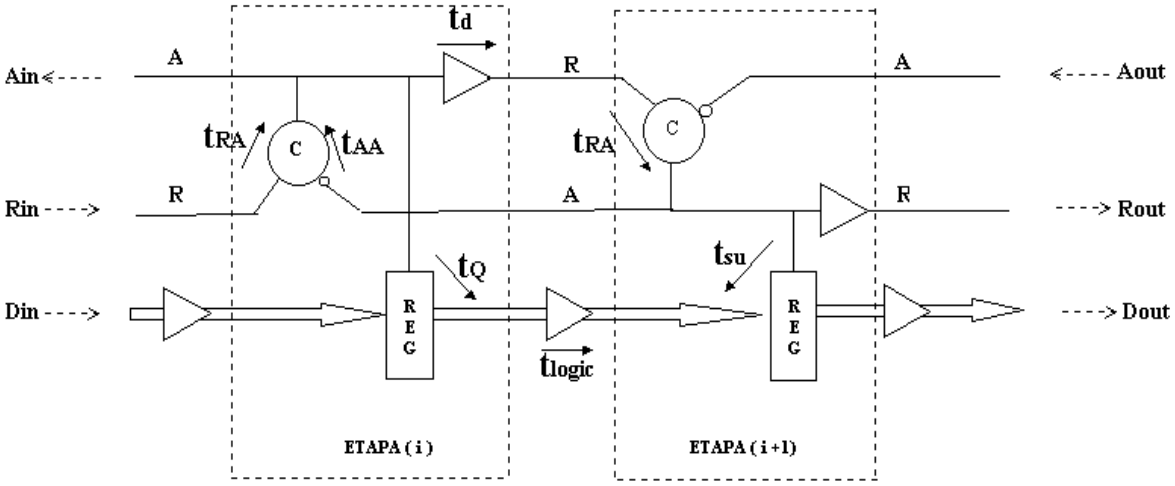

Fig. 5.9: Análisis de rendimiento del micropipeline

También se puede interpretar que para una etapa del micropipeline el tiempo del ciclo mínimo es la suma de las latencias directa e inversa, es decir:

$P= L_f + L_r$
donde:
$L_f = T_Q + T_{logic} + t_{su}$
$L_r = t_{AA}$

La latencia directa (L_f) corresponde al retraso mínimo de la señal que se propaga desde Rin hacia Rout. La latencia inversa (L_r) es el retraso mínimo de la señal que se propaga desde Aout hacia Ain. Basándose en esta formalización se pueden realizar las mediciones de laboratorio que corroboren el dicho análisis. En el Capitulo VI se muestran los cálculos de retardos en los FPGAs utilizados.

En forma general el diseño de conmutadores desde un nivel de sistema, se basa en la optimización de ciertos criterios relacionados a parámetros como el throughput, la tasa de pérdida y el retardo de células, además de otros como área, potencia e interfaces. En el presente trabajo, a fin de facilitar el diseño del conmutador se considera principalmente las características del tráfico y los parámetros requeridos de calidad de servicio QoS. En la sección 5.2 se establecen las figuras de mérito para el área y latencia. Los parámetros de tasa de pérdidas y retardo de células se detallan en la sección 3.4.4, las mediciones de laboratorio y las estimaciones para las plataformas de FPGAs en cuanto de área y componentes lógicos se tratan en el capítulo VI.

5.6.2 Ventajas comparativas y limitaciones

A continuación se describe algunos circuitos síncronos de conmutación ATM similares en nivel de complejidad y plataforma de implementación, se hace hincapié en el método utilizado para administrar la QoS. De lo cual se puede establecer las ventajas y limitaciones principales de la Unidad de Clasificación del presente trabajo.

MuqPro I.

- MuqPro I [64], considera enlaces para la norma OC-3, consiste de 3 FPGAs y 8 chips SRAM. Se proporciona 4 enlaces UTOPÍA-1 y N tarjetas pueden conformar un conmutador 4xN. La implementación se realiza sobre Altera FLEX 10K50.
- La administración de colas puede ser implementada en software o en hardware. Usa una memoria externa para manejo de las colas y el buffer es implementado usando SRAM. El módulo de data-path proporciona las conexiones físicas entre la interfaz de enlaces y la memoria CB (cell bodies), donde todas las células ATM son almacenadas.
- El módulo QM (queue manager) organiza las células en colas lógicas conforme se necesiten, a través de punteros para la memoria CB, el OS (output scheduler) administra las células para su transmisión utilizando un algoritmo HPF (high priority first).

Atlas I.

- Atlas I [65], diseñado para enlaces OC-12, tiene 16x16 enlaces serie a 622 Mbits/seg cada uno, puede configurarse a 8x8 y 1.25 Gbps, a 4x4 y 2.5 Gbps. Las SRAM y CAMs son implementadas en full-custom, consiste de 3 memorias, 2 decodificadores, 1 mantenedor de prioridad, 1 codificador, y circuitos periféricos. Los enlaces tienen como capa física la norma IEEE Std 1355 HIC/HS.
- Internamente opera como un “crossbar” con un buffer compartido de 256 células. Utiliza la técnica denominada “backpressure” que permite, en caso de congestión del buffer, enviar señales a los puertos de entrada para restringir servicios. Las conexiones o grupos de conexiones son puestos en colas con un control de flujo independiente para cada uno. Permitiendo un directo ingreso de células de acuerdo a sus prioridades y re-ingreso en el caso congestión de tráfico. Esto permite que las células no sean descartadas nunca, debido a que son transmitidas cuando se conoce que existe espacio en el buffer del receptor.
- La operación de controlador es tipo pipeline para administrar estructuras de datos a altas velocidades. El bloque QM (Queue Management) se encarga mantener las

células que esperan mediante un esquema de créditos, las células son tratadas de acuerdo si su conexión VC está sujeta a backpressure. La clase de alta prioridad, que no está sujeta a backpressure, es asignada a tráfico de tiempo real por ejemplo voz, para el cual descartar células es preferible a incrementar el retardo.

MuqPro II.

- MuqPro II [64], es una versión mejorada del predecesor ya que está diseñado para la norma OC-12. Tiene 2 puertos bi-direccionales y soporta tráfico ATM bi-direccional entre esos puertos con una adicional capacidad de buffer y colas múltiples de células en ambas direcciones. Cada puerto puede ser configurado para operar de acuerdo a diferentes normas: HIC/HS o UTOPÍA-2.
- Proporciona un gran espacio de buffer y unas grandes tablas de traslación VP/VC, tablas de colas, tablas de créditos, punteros de enlace de listas y datos. En cualquier tiempo varios VCs, que pertenece a diferentes clases de servicio, pueden estar listos para transmitir, elegir uno para ser el siguiente a ser transmitido es una decisión compleja que se realiza sobre la base de varios factores.
- Incorpora memorias DRAM para almacenar células. La información de administración usa SRAM debido a que las estructuras de datos dinámicas requieren punteros rápidos y es necesario cortos tiempos de acceso. El data-path de las células es simple. Las células entrantes son, en general, almacenadas en DRAM y las células salientes se originan desde el mismo buffer de memoria.

Ventajas y limitaciones.

- La Unidad de Clasificación debido a su operación micropipeline puede alcanzar velocidades compatibles con la norma OC-48 e inclusive tasas mayores de tráfico, lo cual depende principalmente de los ciclos de tiempos de acceso de la memoria compartida además de especificaciones de throughput y latencia.

- La Unidad de Clasificación puede administrar un gran número de niveles de prioridad, para la norma OC-12 se considera 2^{16} hasta prioridades sin ninguna degradación del rendimiento. La velocidad de operación no es afectada por la capacidad de la memoria ni por los niveles de prioridades, debido al mecanismo de procesamiento distribuido.

- La arquitectura asíncrona garantiza características óptimas de parámetros como ancho de banda, latencia, tasa de pérdidas y jitter que normalmente no proveen las arquitecturas best-effort actuales.

- La consideración en el diseño del modelo estadístico de tráfico ATM, permite un óptimo uso de recursos sin tener una sobre-utlización de componentes. Facilita el control centralizado ya que libera de la señal de reloj para Unidad de Clasificación-

- Es una arquitectura modular ampliable tanto para otras normas y especificaciones. El diseño asíncrono permite un adaptarse a los requerimientos de tráfico en un instante determinado.

- La arquitectura asíncrona tiene bajo consumo de potencia, bajo EMI como propiedades intrínsecas.

- Una limitación se encuentra en la velocidad y capacidad de la memoria compartida y que necesita una administración global dentro de un sistema heterogéneo síncrono/asíncrono.

- La arquitectura asíncrona necesita de nuevos desarrollos de los diferentes subsistemas. En las interfaces para etapas de entrada y salida, especialmente en los puertos de entrada deben desarrollarse circuitos ad-hoc a fin de explotar las cualidades de los circuitos asíncronos y mejorar las prestaciones respecto a un diseño síncrono.

CAPÍTULO VI

RESULTADOS

6.1 Introducción

El presente capítulo trata de la implementación en FPGAs de ambas arquitecturas paralela y serie los cuales constituyen la Unidad de Clasificación. Requisitos del diseño como portabilidad o recomendaciones para las diferentes etapas de la metodología de diseño son detallados para cada tipo de los FPGAs considerados. Con la finalidad de una mejor exposición se trata en forma separada la implementación de las arquitecturas serie y paralela, los resultados reflejan el efecto de la tecnología de los dispositivos en el rendimiento de la Unidad de Clasificación.

Por otro lado, dentro verificación funcional del diseño micropipeline, se detalla el procedimiento de las mediciones de laboratorio sobre dichas plataformas FPGA, los resultados observados permiten validar la metodología de diseño y comprobar las especificaciones de la Unidad de Clasificación. Finalmente dentro de la aplicación considerada, se establece comparativamente las ventajas y limitaciones de las implementaciones.

6.2 Consideraciones de la implementación

En la presente sección se abordan aspectos de las implementaciones particulares sobre los FPGAs, de acuerdo a la metodología de diseño planteada y los requisitos iniciales, se detallan las recomendaciones seguidas en la implementación en cada plataforma FPGA y el procedimiento de las mediciones. Las implementaciones se realizan sobre las plataformas FLEX 10K, XC4010 y APEX 20KE. Se emplean los compiladores de XILINX Foundation (versión 1.5), de ALTERA MAX+PLUS II (versión 9.6) y QUARTUS II (versión 1.1). Dichos FPGAs representan tecnologías maduras y se dispone toda la información de datos técnicos. Familias recientes de Xilinx (Virtex II) o de Altera (Cyclone y Stratix) pueden ofrecen mayor velocidad y son una alternativa considerar en el futuro, en el anexo B se adjuntan estimados del posible incremento de velocidad.

6.2.1 Portabilidad del diseño micropipeline

Como se mencionó anteriormente, dentro del los objetivos de la presente tesis se considera la portabilidad del diseño como un requisito fundamental, por lo cual en la implementación se considera dos plataformas de FPGAs comerciales disponibles. El

flujo de la metodología de diseño, mostrada en la Figura 2.6, considera la captura del diseño desde VHDL y luego la biblioteca de módulos asíncronos es implantada en la estructura física de celda básica de los FPGAs. La incorporación de LPMs permite minimizar la pérdida de eficiencia en área de silicio.

Un aspecto importante para la portabilidad del diseño micropipeline es la implementación del elemento de referencia de retardo o denominado también bloque "matched delay", en la sección 4.3.2 se muestra la especificación de este módulo.

Para una particular plataforma, en este caso FPGAs, la implementación del bloque matched delay se realiza tomando en cuenta el retardo de cada nivel lógico y el atraso que introduce una unidad retardo (ur). Esta unidad de retardo (ur) se implementa en un equivalente a 3 niveles de lógica, incluye el retardo de lógica, retardo de líneas de conexión y variaciones del proceso tecnológico. Un nivel de lógica está delimitado dentro un solo LAB (Altera) o un solo CLB (Xilinx), de manera práctica se tiene:

T(ur)= 3t, donde t representa el retardo de un nivel de lógica interno.

Este retardo de puede calcularse de la hoja de datos de tiempos del fabricante por ejemplo, con referencia a los parámetros de la Figura 6.1, y para la plataforma FLEX EPF10K20RC240-4:

t_D = 0.6ns (retardo dentro un mismo LAB)

$t_{LE}= t_{LUT} + t_{COMB}$ = 1.7 + 0.6 = 2.3ns.

$t_{LE} + t_D$ = 2.3ns + 0.6ns = 2.9ns

Por lo cual T(ur)= 2.9ns, luego t= (2.9)1/3= 0.97ns

De la misma forma para el FPGA APEX 20K100EQC240-2

$t_{LE} = t_{LUT} + t_{COMB}$ = 0.95ns

t_{F1-4} = 0.27ns

$t_{LE} + t_{F1-4}$ = 0.95ns + 0.27ns = 1.22ns

T(ur)= 1.22ns, luego t= (1.22)1/3 = 0.41ns

Cabe mencionar que a diferencia de la plataforma anterior, la hoja de datos técnicos no menciona el retardo t_D dentro de un mismo LAB, por lo cual se considera t_{F1-4} que es el retardo de una interconexión local.

Para la plataforma Xilinx XC4010XLPC84-3:

La especificación de retardos dentro del CLB considera el retardo combinacional total, que depende de las entradas y salidas utilizadas, el retardo de la lógica de acarreo rápido dentro del CLB es:

Tascy= 3.3ns (Retardo desde la entrada F3 a Cout de un sumador/restador)

Se considera este parámetro debido a que la unidad de retardo se implementa mediante un sumador completo de 1 bit y la salida se realimenta a la entrada, entonces:

T(ur)= 3.3ns, luego t= (3.3)1/3= 1.3ns

De acuerdo con el procedimiento descrito en la sección 4.3.2, para una mejor aproximación en el cálculo de la unidad de retraso se sigue un método sencillo, en el cual se considera una cadena de 20 unidades de retardo, que es lo bastante larga y obtener un valor promedio. Para este número de 20 unidades se toma en cuenta que el layout este localizado en un área que equivalente al área conformado por los LEs o CLBs necesarios, también que los pines I/O se ubiquen próximos a dicha área. De esta forma se tiene un valor muy aproximado al retardo de cada unidad y dicho valor se usa para estimar los matched delays.

De las simulaciones de cada plataforma, que incluyen los pads de entrada y salida y retardo de conexiones, se estima el retardo de una unidad T(ur) y se comprueba que estos valores son muy cercanos a los valores derivados de las hojas técnicas de los fabricantes, las diferencias son menores a 0.1ns. Se tienen los valores siguientes para cada plataforma disponible.

FLEX EPF10K20RC240-4:
T(ur)= 2.9ns, luego t= (2.9)1/3= 0.97ns

APEX 20K100EQC240-2:
T(ur)= 1.12ns, luego t= (1.12)1/3 = 0.37ns

Xilinx XC4010XLPC84-3:
T(ur)= 3.2ns, luego t= (3.2)1/3= 1.1ns

En el cálculo de bloque matched delay, adicionalmente se debe considerar el retardo del routing y también un factor de seguridad para las variaciones del proceso. El retardo de las conexiones se estima por la regla práctica del 50%. Las variaciones del proceso dependen de la tecnología y son menores del 30%, se puede considerar estas variaciones incluyen un factor de seguridad.

Con estas consideraciones, a fin de validar la metodología, la implementación del prototipo se aborda en dos fases en la primera fase ambas arquitecturas serie y paralela se implantan en las plataformas FLEX 10K y XC4010, debido a la limitación de capacidad de puertas lógicas se implementan prototipos pequeños. Esta primera fase permite validar las etapas de síntesis, layout y verificación de tiempos, las mediciones de laboratorio corroboran los resultados previstos en dichas plataformas.

Para la segunda fase de la implementación ambas arquitecturas se amplían para cumplir las especificaciones de la Unidad de Clasificación, esto implica usar un FPGA de mayor capacidad y también de mayor rendimiento. La implementación en la plataforma APEX 20KE muestra que la Unidad de Clasificación satisface los requerimientos para las normas OC-12 y OC-48 en cuanto a la tasa de pérdida de células y el retardo de células.

6.2.2 Implementación en ALTERA FLEX 10K y XILINX XC4010

En esta primera fase tiene por objetivo poner en práctica todos los aspectos involucrados en las diferentes etapas de la metodología de diseño mostrada la sección 2.6.2. Se aplican los conceptos, criterio, y recomendaciones expuestas en los capítulos anteriores, especialmente lo tratado en la sección 4.2, la implementación, mediante FPGAs, de Unidad de Clasificación sirve para validar dicha metodología y sobre la base de mediciones de laboratorio corroborar los resultados.

En la Figura 6.1 se tiene un modelo general para la estimación de los retardos que posteriormente se particulariza para cada plataforma. Se tiene dos registros REG, uno como registro fuente y otro como registro destino, también bloques lógicos combinacionales intermedios. Las indicaciones entre paréntesis dan los retardos equivalentes a los mostrados en la Figura 5.9.

El cálculo del retardo total es la suma de los retardos en el camino crítico (critical path):

$$T_{TOTAL} = t_{CO} + t_{D1} + t_{D2} + ...t_{D(n+1)} + (n)t_{LE} + t_{SU}$$

Donde:
t_{CO}: es el retardo capture-to-output del registro fuente (equivale a t_Q).

t_{Dn}: es el retardo del path, n es el número de niveles lógicos y depende de la ubicación relativa de los elementos lógicos.

t_{LE}: es el retardo de los elementos lógicos ($t_{LUT} + t_{COMB}$). Se denomina LEs en el caso de Altera FLEX 10K y CLBs en el caso de Xilinx XC4010 (equivale a t_{logic}).

t_{SU}: es el retardo setup del registro destino (equivale a t_{su}).

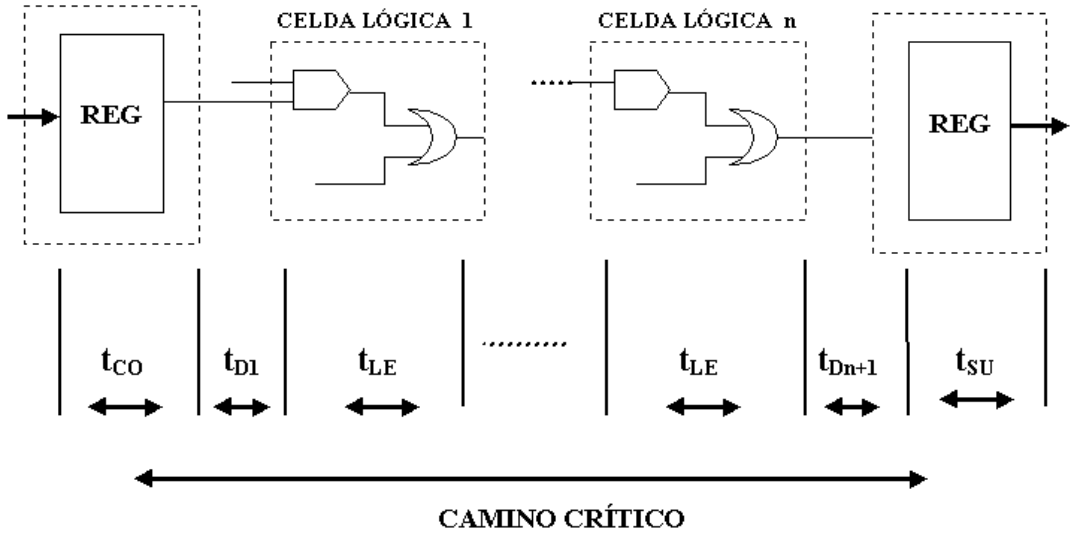

Fig. 6.1: Parámetros de retardo en FPGAs

Para los FPGAs los niveles de lógica se definen basándose en el máximo número de LUTs que contiene el "critical path" de una señal. En ambas plataformas se tiene una

estructura similar basada en LUTs, el uso de LPM o LogicBloxs permite que sean implementaciones eficientes y se mantengan similares en las diferentes plataformas. El compilador de Xilinx permite evaluar directamente para cada bloque constitutivo en cuantos niveles de lógica se implementa dicho bloque.

A continuación se desarrollan los puntos importantes para la implementación en cada plataforma, un aspecto es la estimación de los retardos y otro los aspectos del diseño físico (layout) particular para cada plataforma. En el anexo B se adjuntan aspectos adicionales de los parámetros para cada plataforma considerada.

Implementación en ALTERA FLEX 10K.

- Estructura relativa a los retardos.

En referencia al modelo general para la estimación de los retardos de la Figura 6.1, a continuación se muestran los valores típicos de retardo para el dispositivo FLEX EPF10K20RC240-4:

t_{CO}= 1.5ns
t_D = 0.6ns (mismo LAB)
$t_D = t_{SAMEROW}$ = 3.9ns (LAB en la misma fila)
$t_D = t_{DIFFROW}$ = 5.5ns (LAB en diferentes filas)
$t_{LE}= t_{LUT} + t_{COMB}$ = 1.7 + 0.6 = 2.3ns.
t_{SU}= 2.5ns

La familia FLEX10K tiene una estructura de routing continua que proporciona una temporización predecible y permite determinar los retardos involucrados de un diseño. Para 2 LEs se tienen 3 posibilidades: 1) que estén en el mismo LAB, 2) que estén en diferentes LABs pero en la misma fila, 3) que estén en diferentes filas. Para los dos últimos casos, el compilador considera un fan-out y la distancia.

En el compilador MAX+PLUS II y QUARTUS II, se puede habilitar la opción “carry chain” que permite un menor retardo entre LEs, los LPMs automáticamente habilitan dicha opción. También se puede usar “cliques” para agrupar toda la lógica del “critical path” para optimizar el retardo. En cualquier caso el compilador calcula automáticamente los parámetros de retardo y permite valorar el rendimiento del diseño.

- Recomendaciones para el layout.

El empleo de las funciones pertenecientes a la Biblioteca de Módulos Parametrizables (LPM) [44], proporciona una entrada de diseño independiente de la arquitectura y de la tecnología para todos los dispositivos soportados por MAX+PLUS II y QUARTUS II.

Se tiene la posibilidad de fijar la fila y columna de los LABs en donde se desea hacer la implementación de partes críticas de micropipeline como los latches o los circuitos de control. Igualmente se puede fijar la distribución adecuada de los pines de entrada/salida. Esta información de “placement” puede ser suministrada desde un esquemático o desde un archivo de configuración.

La estructura de routing continua mediante canales interconectados fast-track, proporciona una temporización acotada y permite estimar el rendimiento de un diseño mediante los parámetros de retardo del dispositivo. La herramienta "Matrix Delay" da los valores máximos de los retrasos y permite corroborar los retardos.

Se puede usar la opción cliques para agrupar circuitos independientemente de su ubicación, esto permite optimizar los retardos, esta opción es útil para implementar los bloques matched delay. También se puede ajustar la opción "carry chain" para la implementación de funciones aritméticas como los bloques "comparators", los LPMs automáticamente habilitan dicha opción. En el anexo B se muestran especificaciones adicionales.

Implementación en XILINX XC4010.

- Estructura relativa a los retardos.

Para el dispositivo XC4010XLPC84-3, igualmente se tienen los siguientes valores de retardo:

t_{CO}= 1.5ns - 2.1ns
t_D = 0.1ns (dentro del mismo CLB ó 20ns para varios "switching points")
t_{LC}= t_{LE}= t_{LUT} + t_{COMB} = 1.6ns - 3.3ns (depende de las entradas/salidas que se usa)
t_{SU}= 0.6ns - 1.1ns

- Recomendaciones para el layout.

Una facilidad equivalente a los LPMs son los LogiBLOX y Core Generator [40]. Los circuitos generados pueden ser transportados a otras plataformas permitiendo que el diseño sea independiente de la arquitectura y de la tecnología.

Tanto el número de bloques CLB y el retardo son minimizados implementando funciones que ocupen un solo bloque CLB, por lo cual se incrementa la densidad y la velocidad. En la familia XC4010 los flip-flops pueden ser configurados como simples latches y permite que los latches sean agrupados de forma relativa uno a otro, igualmente los bloques matched delays ocupan un menor número de CLBs.

Se tiene la posibilidad, mediante la facilidad RLOC, de fijar los CLBs en los cuales se desea hacer la implementación de partes críticas de micropipeline como los latches o los circuitos de control. Igualmente se puede fijar la distribución adecuada de los pines de entrada/salida. Esta información de "placement" puede ser suministrada desde un esquemático o desde un archivo de configuración.

La estructura segmentada de routing minimiza la capacidad de interconexión y proporciona unos retardos acotados que permiten estimar el rendimiento de un diseño mediante los parámetros de retardo del dispositivo. La herramienta de simulación del compilador da los valores máximos de los retrasos a través de los diferentes "switching points" y permite corroborar los retardos. En el anexo B se muestran especificaciones adicionales.

6.2.3 Implementación en ALTERA APEX 20KE

Esta familia de FPGAs, en la tecnología CMOS de 0.15 micras, representa una mejora significativa respecto a las dos anteriores y es el mejor candidato que la familia FLEX para servir como plataforma de implementación de la Unidad de Clasificación. APEX 20KE se puede configurar en 3 modos: normal, aritmético, contador y para cada uno los LEs usan los recursos de forma apropiada. El software QUARTUS II y el empleo de LPMs permite habilitar el modo de operación adecuado automáticamente.

A continuación se considera los aspectos principales para la implementación y se compara para las versiones de "speed grade" –2 y –1. En el anexo B se muestran especificaciones adicionales.

- Estructura relativa a los retardos.

Para el dispositivo APEX EP20K100E–2, el retardo de path (t_D) considera la distancia entre filas, se tiene los siguientes valores típicos:

t_{CO} = 0.28ns
t_{LE} = t_{LUT} + t_{COMB} = 0.95ns
t_{SU} = 0.25ns
$t_{F1\text{-}4}$ = 0.27ns
T(ur)= 1.12ns (Valor práctico estimado de una unidad de retardo)

Igualmente para el dispositivo APEX EP20K100E–1:

t_{CO} = 0.28ns
t_{LE} = t_{LUT} + t_{COMB} = 0.80ns
t_{SU} = 0.25ns
$t_{F1\text{-}4}$ = 0.24ns
T(ur)= 0.96ns (Valor práctico estimado de una unidad de retardo)

La comparación de los valores de T(ur) de los "speed grade" -1 y -2 muestran que se puede obtener alrededor de un 15% de mejora de throughput en el diseño micropipeline de Unidad de Clasificación, lo cual es corroborado mediante simulaciones.

- Recomendaciones del layout.

La estructura de los LEs, LABs y agrupaciones de LABs denominadas MEGALABs, se mantiene similares al de FLEX 10K por lo cual el layout se realiza según las recomendaciones anteriores del mencionado FPGA.

6.2.4 Metodología de las mediciones

En la Figura 6.2 se muestra el esquema circuital utilizado para las mediciones en ambas arquitecturas paralela y serie. El objetivo principal es corroborar el funcionamiento correcto de las implementaciones en las plataformas FPGAs arriba mencionadas y la medición del throughput en cada caso. En la Unidad de Clasificación se tienen algunas características que se han mencionado en los capítulos anteriores:

- Todas las etapas del micropipeline tienen la misma profundidad lógica, esto significa que el tiempo de retardo de cada etapa es el mismo. Las consideraciones de layout permiten que las variaciones debido al retardo de las conexiones sean minimizadas y no influyen en las mediciones.

- El throughput y la latencia se mantienen iguales para cada etapa y también para el caso de considerar todo el conjunto del micropipeline. Esto permite que las mediciones se realicen directamente sobre todo el micropipeline. Independientemente del número de etapas, se tendrá un valor de medición que se puede considerar el promedio para cada etapa.

En la Figura 6.3 sirve para explicar el cálculo del throughput y la latencia en ambas arquitecturas paralela y serie basándose en las señales de control del micropipeline.

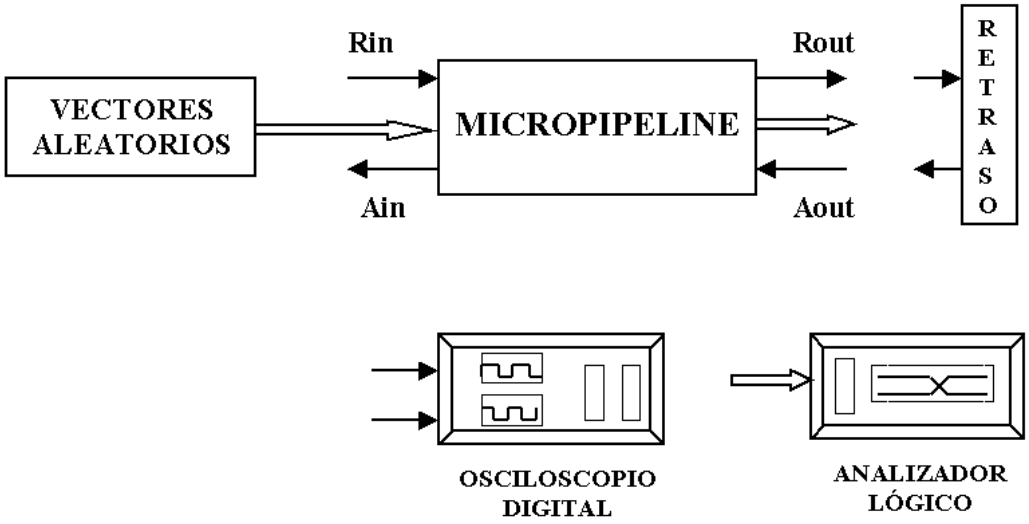

Fig. 6.2: Esquema de mediciones para el micropipeline

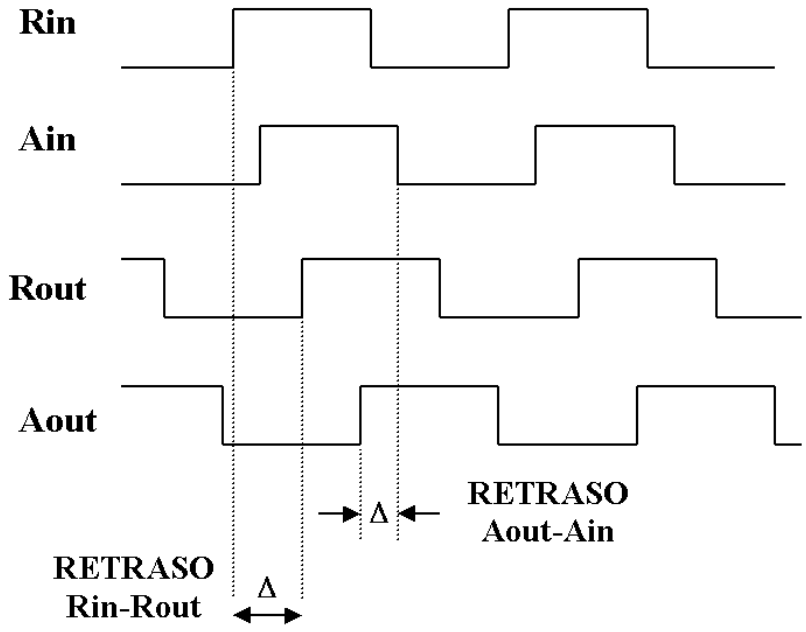

Fig. 6.3: Diagrama para el cálculo del throughput

Tomando en cuenta las características anteriores se realiza las mediciones con los siguientes criterios:

- La observación mediante el analizador lógico de los vectores de salida de ambas arquitecturas serie y paralela, los cuales son inyectados a la entrada de forma pseudo-aleatoria (el generador es de 8 bits con secuencias de 2^8-1 bits), a la salida se tienen los vectores clasificados según su valor binario (prioridad). Esto permite corroborar el funcionamiento de su operación de clasificación. En las figuras 6.4A y 6.5A se tienen vistas del analizador lógico.

- La medición, mediante el osciloscopio digital, de la latencia y el throughput del micropipeline se realiza observando las señales Rin, Ain en la entrada y las señales Rout, Aout en la salida. La condición necesaria es que se tenga la más alta tasa de transferencia de datos en la entrada y en la salida simultáneamente.

- La más alta tasa de transferencia constante de datos en el micropipeline se obtiene cuando todos los bloques del micropipeline operan de forma sostenida a dicha tasa de transferencia y no se tienen bloques detenidos (idle) esperando sus correspondientes señales locales de Rin o Aout.

- Debido en algunos casos, por ejemplo del sistema de desarrollo FLEX UP1 utilizado, la señal de reloj externa disponible en la tarjeta del FPGA es fija o solo se puede dividir entre un número entero (caso del sistema XS40), es una limitación para llevar al circuito a la cadencia máxima de operación. Pero es suficiente llevar el micropipeline a una tasa de transferencia de los datos lo más cercana al máximo para la cual se cumpla las condiciones anteriores y observar que no exista traslape de las señales, es decir, se tenga la condición mostrada en la figura 6.3. Donde a la señal Rin responde Ain, luego Rout. La señal Aout se activa antes del siguiente Rin. En estas condiciones, mediante el osciloscopio digital, las mediciones de retardo (Δ) entre las señales Rin-Rout y Aout-Ain dan el máximo throughput del micropipeline. Es decir, el throughput es la inversa de la latencia (P).

  $P = \Delta(\text{Rin-Rout}) + \Delta(\text{Aout-Ain})$ y el throughput $= 1/P$

- A fin de asegurar que todos los bloques del micropipeline operan a tasa constante y no se tienen bloques detenidos (idle), en la etapa final del micropipeline, se aplica Rout hacia la señal de entrada Aout luego de un retraso adecuado (figura 6.2) que permita operar los bloques del micropipeline en las condiciones señaladas. Este retraso se ajusta de acuerdo al retraso que tienen los bloques de procesamiento del micropipeline.

- Se observa que dependiendo de la transición de la señal (de subida o bajada) que se tome como referencia en las mediciones, puede variar la lectura en los instrumentos. Esto es debido a dichas transiciones tienen retardos distintos pero fijos, por lo cual se toma el valor medio.

- Todas las mediciones se realizan bajo condiciones de operación normales en cuanto al voltaje de alimentación y temperatura ambiente para cada dispositivo de FPGA. En las figuras 6.4B y 6.5B se tienen vistas de las señales de control en el osciloscopio digital.

6.3 Implementación de la arquitectura paralela

Para la implementación de la arquitectura paralela se toma en cuenta los bloques constitutivos y las características tecnológicas de los FPGAs. Para dicha implementación se muestra los cálculos de los parámetros de rendimiento como el throughput y el área, se concluye con los resultados de las mediciones de laboratorio.

6.3.1 Análisis de sus prestaciones

En el análisis de las prestaciones de la implementación en las plataformas FPGAs de la arquitectura paralela se toma en cuenta dos principales parámetros: el throughput y el área ocupada. En la implementación de la Unidad de Clasificación se puede considerar como principal parámetro el throughput, el cual en cada etapa del micropipeline está directamente relacionado al bloque de retardo (matched delay). . El análisis detallado se muestra en la sección 5.6.1. En esta sección se detalla el procedimiento del cálculo del retardo necesario a ser introducido en los bloques matched delay de la arquitectura paralela.

En la Tabla 6.1 se tienen los niveles de lógica de los bloques constitutivos de la arquitectura paralela.

BLOQUES PRINCIPALES	ELEMENTO-C	LATCH	COMP	MUX
NIVELES DE LOGICA (16 BITS / VECTOR)	3	3	11	3

Tabla 6.1: Niveles de lógica en la arquitectura paralela

A fin de mostrar el método de los cálculos para la arquitectura paralela, se considera 8 vectores, cada vector de 16 bits, y el FPGA APEX 20KE. Para el bloque latch se considera que la suma de los tiempos de setup, hold y output es equivalente al tiempo de retardo del equivalente en niveles de lógica, en este caso a 3 niveles de lógica.

- Cálculo referido a los niveles de lógica.

Para la arquitectura paralela de la Figura 5.2, en el bloque de lógica de procesamiento, el mayor retardo se tiene en la trayectoria:

T(total)= T(comp) + T(mux) + T(latch)

Si cada nivel lógico tiene un retraso “t” y considerando los niveles de cada bloque:

T(total)= 11t + 3t + 3t= 17t

- Cálculo del bloque matched delay.

Una vez que se ha calculado la profundidad lógica del mayor retardo, referido al retardo de un nivel lógico, se calcula el número de unidades de retardo para compensar dicha profundidad lógica.

Tomando como referencia la plataforma FPGA APEX 20K100EQC240-2 se tiene:

T(ur)= 1.12ns, luego t= 1/3 (1.12)= 0.37ns

Luego el retardo total: T(total)= 17t= 17(0.37ns)= 6.3ns.

El retardo debido al routing es similar: T(routing)= 6.3ns

El factor de seguridad debido a las variaciones del proceso:
T(process)= 30% (6.3ns + 6.3ns)= 3.8ns

El bloque matched delay debe tener un retraso:
T(matched)= 6.3ns + 6.3ns + 3.8ns = 16.4ns.

Esto equivale a 16.4ns/1.12ns=14.6 y se consideran 15 unidades de retardo. Con estos resultados se realizan las simulaciones en QUARTUS II [39] para verificar los tiempos y el correcto funcionamiento.

Para la arquitectura paralela, el simulador da los valores los tiempos de propagación de las señales de control Rin y Rout, igualmente entre Aout y Ain. Con estos valores y tomando en cuenta el número de etapas del micropipeline se estima el throughput que es verificado mediante simulación.

Para 8 vectores de entrada, cada vector de 16 bits, se necesita una mayor capacidad y mayor número de pines I/O, por lo cual se usa el FPGA de la misma familia: EP20K160EBC356-2. Para este FPGA es posible ajustar el bloque matched delay a 14 unidades de retardo y se simula a una tasa de transferencia de datos de 25ns.

Luego la latencia de cada etapa es: 25ns
El throughput máximo: 1/25ns = 40 M outputs/seg.

Estos valores de consideran como “operación segura”, indicando que el throughput puede ser mayor y cuyo máximo se determina mediante las mediciones experimentales, dicho throughput medido es alrededor de un 5% mayor que el dado por la simulación.

6.3.2 Mediciones experimentales

La metodología de las mediciones, que se detalla en la sección 6.2.4, se aplica para ambas arquitecturas paralela y serie. Como se reseño anteriormente en una primera fase y de acuerdo a los objetivos planteados en la presente tesis, se realiza la implementación en plataformas de FPGAs FLEX 10K y Xilinx XC4010. En la segunda fase se implementa en APEX 20KE.

En las Tablas 6.2A y 6.2B, se muestran algunas comparaciones del área ocupada en cada FPGA de los bloques que conforman la arquitectura paralela, se toma en cuenta el número de LEs (logic element) para Altera y el número de CLBs (configurable logic block) para Xilinx.

BLOQUES PRINCIPALES (4 BITS / VECTOR)	ELEMENTO-C	LATCH	COMP	MUX
NÚMERO DE CLBs XC 4010XLPC84	1/2	6	4	2
NÚMERO DE LEs FLEX 10K20RC240	1	12	3	4

Tabla 6.2A

BLOQUES PRINCIPALES APEX EP20K100EQC240	ELEMENTO-C	LATCH	COMP	MUX
NÚMERO DE LEs (4 BITS / VECTOR)	1	12	4	4
NÚMERO DE LEs (16 BITS / VECTOR)	1	24	10	8

Tabla 6.2B

Tablas 6.2A - 6.2B: Comparaciones de área en la arquitectura paralela

En la Tabla 6.2A se observa que para el bloque COMP la implementación en el FPGA FLEX 10K es más eficiente que su implementación en el FPGA XC4010. Esto se deduce porque en las implementaciones de los otros bloques, el área es el doble en número de LEs que el número de CLBs y es la misma relación de área para la implementación del elemento-C que es el bloque de referencia. El bloque COMP es un circuito aritmético por lo cual su implementación mediante LPMs habilita automáticamente opciones para una implementación eficiente y necesita solo 3 LEs.

En la Tabla 6.2B se observa para el FPGA APEX 20KE que cuando se incrementa el número de bits de 4 a 16, el bloque COMP necesita un mayor número de LEs en comparación de los otros bloques los cuales mantienen la misma relación de 2 a 1. Por estas observaciones de deduce que el bloque COMP es un componente crítico dentro del diseño.

A continuación se presentan los resultados de la verificación funcional y de las mediciones del throughput para la arquitectura paralela, con el fin de ilustrar mejor los cálculos en los cuales se basa el método descrito en la sección 6.2.4. En las Figuras 6.4A y 6.4B se muestran las principales vistas del analizador lógico TEKTRONIX TLA704 y del osciloscopio digital TEKTRONIX TDS430A.

En la Figura 6.4A se tiene un ejemplo de la vista del analizador lógico para los vectores de salida de la arquitectura paralela, en la entrada se tienen vectores pseudo-aleatorios, cada uno de los 8 vectores tiene 4 bits. Se observa que los 8 vectores son clasificados correctamente de acuerdo a su valor de prioridad. En la parte superior se muestra las señales de control del micropipeline y mediante las cuales se realiza las mediciones para el cálculo del throughput.

En la Figura 6.4B se tiene las vistas en el osciloscopio digital de las señales de control para la arquitectura paralela, siguiendo el método descrito anteriormente, mediante los retardos entre Rin-Rout y Aout-Ain se realizan las mediciones del throughput. Las

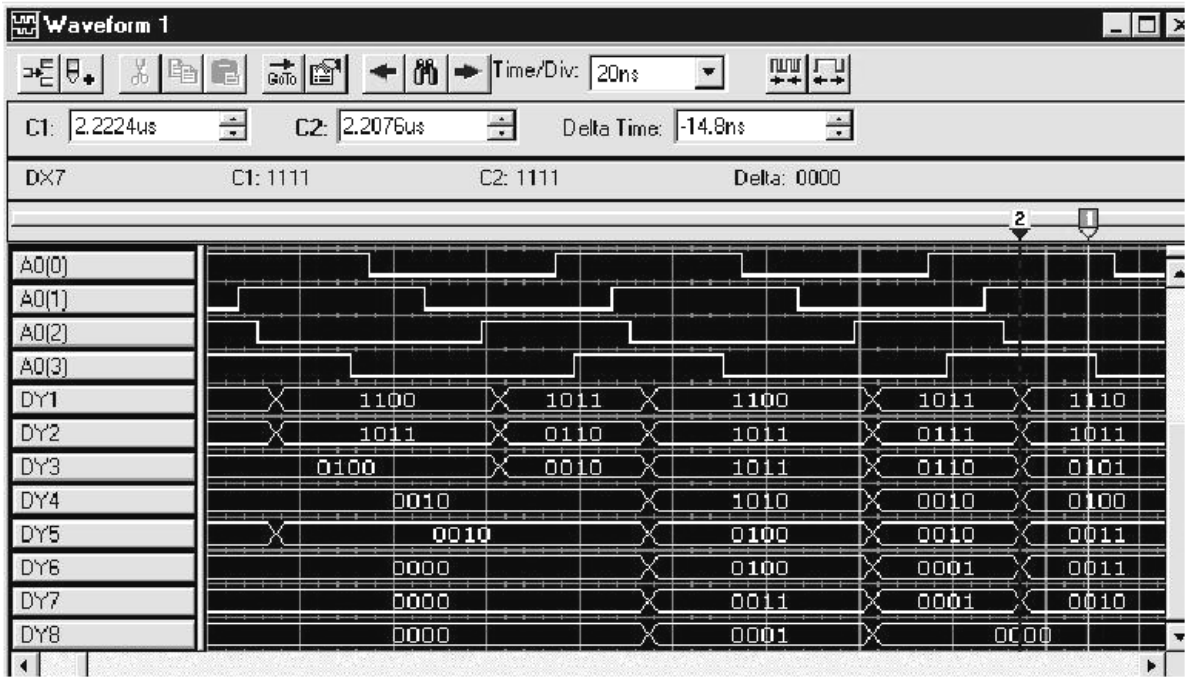

Fig. 6.4A: Vista del analizador lógico para la arquitectura paralela

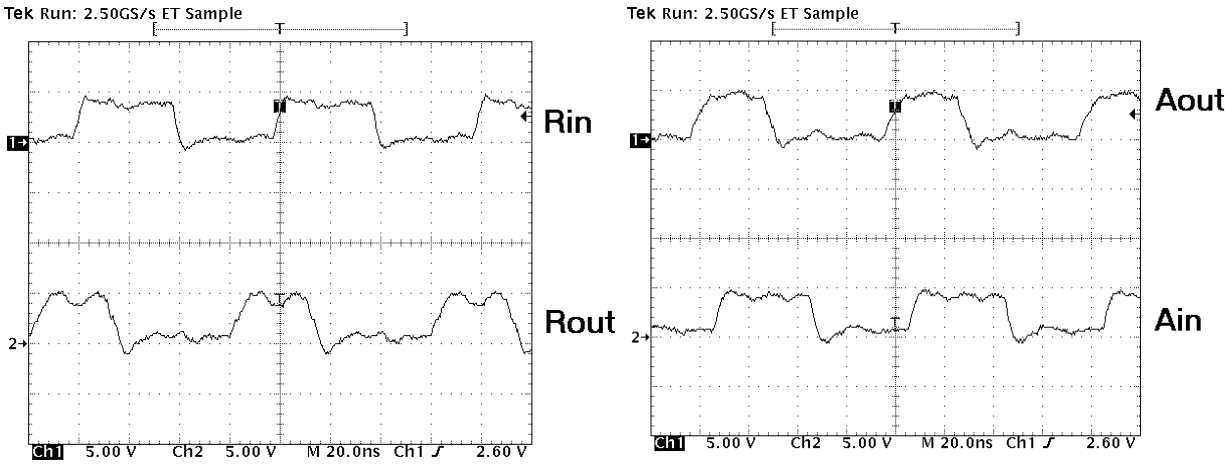

Fig. 6.4B: Vistas de las señales de control en la arquitectura paralela

vistas corresponden a la implementación en el FPGA FLEX 10K20RC240-4 del sistema de desarrollo UP1 y se observa una buena definición de las formas de onda que permiten una correcta medición.

Las Tablas 6.3A, 6.3B, 6.3C, resume los principales resultados de la implementación para arquitectura paralela considerando las normas OC-48 y OC-12, se incluye el circuito de clasificación bitónico y el buffer correspondiente. En todos los casos, las mediciones basadas en dichos instrumentos, permiten medir los valores los tiempos de propagación de las señales de control Rin y Rout, igualmente entre Aout y Ain. Con estos valores y tomando en cuenta el número de etapas del micropipeline se estima el throughput y latencia, los cuales se resumen en las tablas siguientes.

La Tabla 6.3A muestra el área y el throughput para el FPGA FLEX 10K, un resultado importante es que aún para un número reducido de bits de prioridad el valor del throughput no satisface la especificación de retardo de células de la norma OC-48. También sirve para mostrar el impacto de las correspondientes tecnologías de fabricación comparado con el FPGA APEX 20KE.

ARQUITECTURA PARALELA	ÁREA	THROUGHPUT MEDIDO
FLEX 10K20RC240-2 8 VECTORES 4 BITS / VECTOR	1021 LEs (88%)	28.9M outputs/seg (34.60 ns/output)

Tabla 6.3A: Implementación paralela en FLEX 10K

ARQUITECTURA PARALELA	ÁREA	THROUGHPUT
APEX 20K100EQC240-1 8 VECTORES 8 BITS /VECTOR	1808 LEs (44%)	50 M outputs/sec (20 ns/output)
APEX 20K160EBC356-2 8 VECTORES 16 BITS/VECTOR	3560 LEs (56%)	40 M outputs/sec (25 ns/output)

Tabla 6.3B: Implementación paralela en APEX 20KE

BUFFER PARALELO	ÁREA	THROUGHPUT
APEX 20K100EQC240-1 8 BITS /VECTOR	192 LEs/ETAPA	71 M outputs/sec (14 ns/output)
APEX 20K160EBC356-2 16 BITS/VECTOR	384 LEs/ETAPA	47 M outputs/sec (21 ns/output)

Tabla 6.3C: Buffer paralelo en APEX 20KE

La Tabla 6.3B muestra el área y el throughput para el FPGA APEX 20KE, considerando lo descrito en la sección 3.4.4 sobre las normas OC-48, OC-12 y el throughput necesario para el retardo de células, se observa que la Unidad de Clasificación si puede cumplir dicha especificación y que dicho FPGA es la mejor opción para una implementación final.

En la Tabla 6.3C muestra el área y el throughput del buffer paralelo para el FPGA APEX 20KE, como se describe en la sección3.4.3, este buffer sirve para transportar los vectores con las direcciones de memoria. Se observa que para ambas normas consideradas OC-48 y OC-12 se pueden tener respectivamente la capacidad suficiente de direcciones de memoria para cumplir con el requisito de CLP (probabilidad de pérdida de células) respetando el throughput necesario.

6.4 Implementación de la arquitectura serie

En forma similar, en este caso para la implementación de la arquitectura serie se toma en cuenta los bloques constitutivos y las características tecnológicas de los FPGAs. Para dicha implementación se muestra los cálculos de los parámetros de rendimiento como el throughput y el área, se concluye con los resultados de las mediciones de laboratorio.

6.4.1 Análisis de sus prestaciones

Con las mismas consideraciones mencionadas en la sección 6.2.4 y como en el caso de la arquitectura paralela, se hace hincapié en los parámetros de throughput y el área ocupada, el análisis detallado se muestra en la sección 5.6.1. En esta sección se detalla el procedimiento del cálculo del retardo necesario a ser introducido en los bloques matched delay para la arquitectura serie.

También en referencia a la Tabla 6.4, donde se tienen los niveles de lógica de los bloques constitutivos de la arquitectura serie y a fin de mostrar el método de los cálculos para la arquitectura serie, se considera 8 vectores cada vector de 16 bits, para el FPGA APEX 20KE.

BLOQUES PRINCIPALES	ELEMENTO-C	LATCH	CMP	MUX	DETEC	AND	SINCT
NIVELES DE LOGICA **(16 BITS/VECTOR)**	**3**	**3**	**11**	**3**	**4**	**3**	**4**

Tabla 6.4: Niveles de lógica en la arquitectura serie

- Cálculo referido a los niveles de lógica.

Para la arquitectura serie Figura 5.4, en el bloque de lógica de procesamiento, el mayor retardo se tiene en la trayectoria:

T(cmp) + T(and) + T(mux)

Si cada nivel lógico tiene un retraso “t” y considerando los niveles de cada bloque:

T(total)= 11t + 3t + 3t= 17t

- Cálculo del bloque interno matched delay.

Una vez que se ha calculado la profundidad lógica del mayor retardo, referido al retardo de un nivel lógico, se calcula el número de unidades de retardo para compensar dicha profundidad lógica. En la arquitectura serie, el bloque de procesamiento es un circuito secuencial, por lo cual se tiene un bloque matched delay interno y otro bloque matched delay externo para el control del micropipeline. A continuación se detalla el cálculo del primero.

Para el FPGA APEX 20K100EQC240-2 se tiene:

T(ur)= 1.12ns, luego t= 1/3 (1.12)= 0.37ns

Luego el retardo total: T(total)= 17t= 17(0.37ns)= 6.3ns.
El retardo debido al routing es similar: T(routing)= 6.3ns

El factor de seguridad debido a las variaciones del proceso:
T(process)= 30% (6.3ns + 6.3ns)= 3.8ns

El bloque matched delay debe tener un retraso:
T(matched)= 6.3ns + 6.3ns + 3.8ns = 16.4ns.

Esto equivale a 16.4ns/1.12ns= 14.6 y se consideran 15 unidades de retardo.

- Cálculo del bloque externo matched delay.

Para el bloque externo se debe añadir al bloque interno el retraso de latch micropipeline que es equivalente a 3 unidades, además, se debe añadir el retraso del circuito de control SINCT que es equivalente a 4 unidades.

T(ext)= T(matched) + T(latch) + T(SINCT)

T(add)= T(latch) + T(SINCT)= 3t + 4t= 7t

Siguiendo el caso para la plataforma APEX 20K100EQC240-2:

T(add)= 7t= 7(0.37ns)= 2.59ns.

El retardo debido al routing es similar, T(routing)= 2.59ns
Debido a las variaciones del proceso T(process)= 30% (2.59ns + 2.59ns)= 1.55ns

T(matched)= 2.59ns + 2.59ns + 1.55ns = 6.73ns

Equivale a 6.73ns/1.12= 6.00, se toma 7 unidades de retardo.

Luego:

T(ext)= T(matched) + T(latch) + T(SINCT) = 15T(ur) + 7T(ur)= 22T(ur).

El T(ext) equivale a 22 unidades de retraso.

En este caso, el compilador de Altera automáticamente permite que de las 22 unidades externas, se deriven 15 unidades para bloque interno y las 7 adicionales son para el retardo externo. Para la arquitectura serie, el simulador da los valores los tiempos de propagación de las señales de control Rin y Rout, igualmente entre Aout y Ain. Con estos valores y tomando en cuenta el número de etapas del micropipeline se estima el throughput que es verificado mediante simulación.

Para 8 etapas de la arquitectura serie, se comprueba su correcto funcionamiento mediante simulaciones en QUARTUS II [39] se verifican los tiempos y se ajustan los retardos, en este caso se confirma el usar 22 unidades para el bloque de retraso externo y 15 unidades para el bloque retraso interno. Se simula a una tasa de 40ns.

Luego la latencia de cada etapa es: 40ns
El throughput máximo: 1/40ns = 25 M outputs/seg.

De la misma forma que el en caso paralelo, estos valores de consideran como "operación segura", indicando que el throughput puede ser mayor y cuyo máximo se determina mediante las mediciones experimentales, dicho throughput medido es alrededor de un 5% mayor que el dado por la simulación.

6.4.2 Mediciones experimentales

La metodología de las mediciones, que se detalla en la sección 6.2.4, se aplica para ambas arquitecturas paralela y serie. Como se indicó anteriormente, en una primera fase y de acuerdo a los objetivos planteados en la presente tesis, se realiza la implementación en plataformas de FPGAs FLEX 10K y Xilinx XC4010. En la segunda fase se implementa en el FPGA APEX 20KE.

En las Tablas 6.5A y 6.5B, se muestra la comparación del área ocupada en cada FPGA de los bloques que conforman la arquitectura serie, se toma en cuenta el número de LEs (logic element) para Altera y el número de CLBs (configurable logic block) para Xilinx.

En la Tabla 6.5A se observa que para el bloque CMP la implementación en el FPGA FLEX 10K es más eficiente que su implementación en el FPGA XC4010. Esto se deduce porque en las implementaciones de los otros bloques como LATCH o MUX, el área es el doble en número de LEs que el número de CLBs y es la misma relación de área para la implementación del elemento-C que es el bloque de referencia. El bloque COMP es un circuito aritmético por lo cual su implementación mediante LPMs habilita automáticamente opciones para una implementación eficiente y necesita solo 3 LEs.

En la Tabla 6.5B se observa para el FPGA APEX 20KE que cuando se incrementa el número de bits de 4 a 16, el bloque CMP necesita un mayor número de LEs en comparación de los otros bloques LATCH o MUX los cuales mantienen la misma

relación de 2 a 1. Por estas observaciones de deduce que el bloque CMP es un componente crítico dentro del diseño. Dentro de los otros circuitos de control el bloque DETEC depende del número de bits de los vectores de entrada y los otros bloques como AND o SINCT tienen el mismo número de LEs.

BLOQUES PRINCIPALES (4 BITS / VECTOR)	ELEMENTO-C	LATCH	CMP	MUX	DETEC	AND	SINCT
NÚMERO DE CLBs XC 4010XLPC84	1/2	6	4	2	1	1	2
NÚMERO DE LEs FLEX 10K20RC240	1	12	3	4	1	1	3

Tabla 6.5A

BLOQUES PRINCIPALES APEX EP20K100EQC240	ELEMENTO-C	LATCH	CMP	MUX	DETEC	AND	SINCT
NÚMERO DE LEs (4 BITS / VECTOR)	1	12	4	4	1	1	3
NÚMERO DE LEs (16 BITS/ VECTOR)	1	24	10	8	5	1	3

Tabla 6.5B

Tablas 6.5A- 6.5B: Comparaciones de área en la arquitectura serie

A continuación se presentan los resultados de la verificación funcional y de las mediciones del throughput para la arquitectura serie, con el fin de ilustrar mejor los cálculos en los cuales se basa el método descrito en la sección 6.2.4. En las Figuras 6.5A y 6.5B, se muestran las principales vistas del analizador lógico TEKTRONIX TLA704 y del osciloscopio digital TEKTRONIX TDS430A.

En la Figura 6.5A se tiene un ejemplo de la vista del analizador lógico para los vectores de salida de la arquitectura serie, en la entrada serie ingresan vectores pseudo-aleatorios en bloques de 8 vectores, cada uno de los 8 vectores tiene 4 bits. Se observa en la salida serie que los 8 vectores son clasificados correctamente de acuerdo a su valor de prioridad. En la parte superior se muestra las señales de control del micropipeline y mediante las cuales se realiza las mediciones para el cálculo del throughput.

En la Figura 6.5B, se tiene las vistas en el osciloscopio digital de las señales de control para la arquitectura serie, siguiendo el método descrito anteriormente, mediante los retardos entre Rin-Rout y Aout-Ain se realizan las mediciones del throughput. Para esta arquitectura serie, las vistas corresponden a la implementación en el FPGA

XC4010XLPC84-3 del sistema de desarrollo XS40 y se observa una buena definición de las formas de onda que permiten una correcta medición.

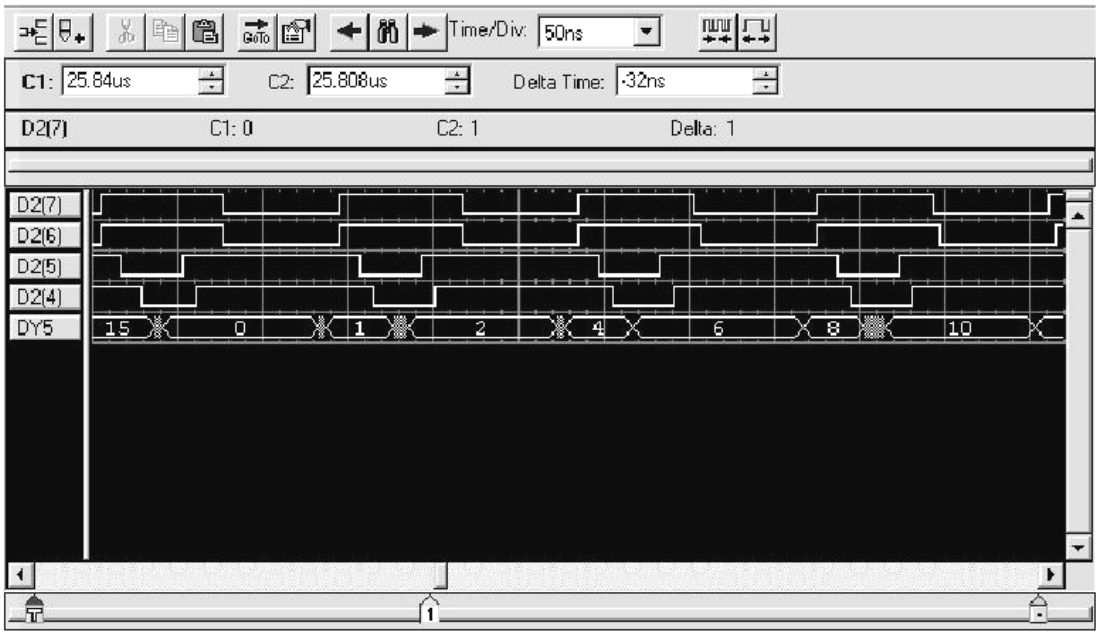

Fig. 6.5A: Vista del analizador lógico para la arquitectura serie

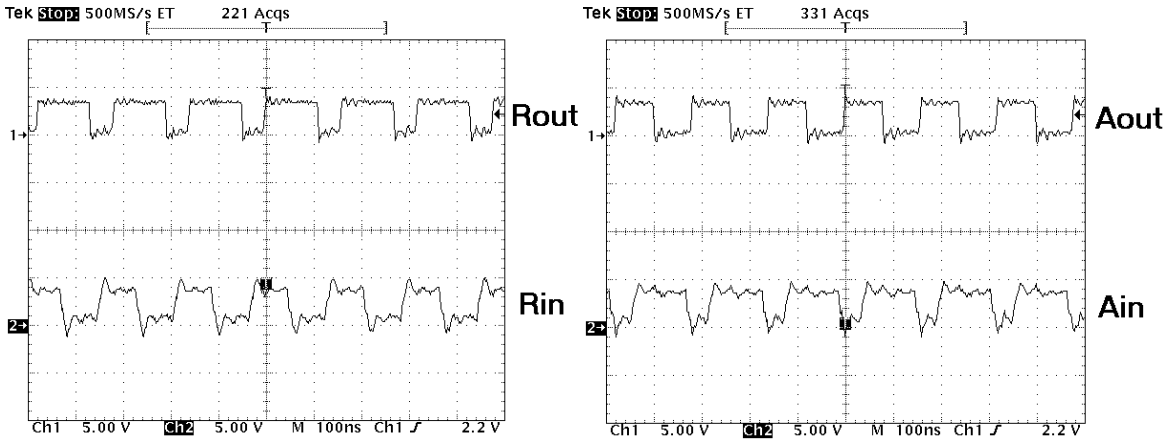

Fig. 6.5B: Vistas de las señales de control en la arquitectura serie

Las Tablas 6.6A, 6.6B, 6.6C, resumen los principales resultados de la implementación para arquitectura serie considerando las normas OC-48 y OC-12, se incluye el circuito de clasificación serie y el buffer correspondiente. En todos los casos, las mediciones basadas en dichos instrumentos, permiten medir los tiempos de propagación de las señales de control Rin y Rout, igualmente entre Aout y Ain. Con estos valores y tomando en cuenta el número de etapas del micropipeline se estima el throughput y latencia, los cuales se resumen en las tablas siguientes.

La Tabla 6.6A muestra el área y el throughput para el FPGA XC 4010 y FLEX 10K, un resultado importante es que aún para un número reducido de bits de prioridad los

valores del throughput no satisfacen la especificación de retardo de células de la norma OC-48. También sirve para mostrar el impacto de las correspondientes tecnologías de fabricación comparados ambos y con el FPGA APEX 20KE.

ARQUITECTURA SERIE	ÁREA	THROUGHPUT
XC 4010XLPC84-3 8 VECTORES 4 BITS / VECTOR	336 CLBs (84%)	14.04 M outputs/seg (71.23 ns/output) MEDIDO
FLEX 10K20RC240-4 8 VECTORES 4 BITS / VECTOR	504 LEs (43%)	19.8 M outputs/seg (50.5 ns/output)

Tabla 6.6A: Implementación serie en XC4010 y FLEX10K

ARQUITECTURA SERIE	ÁREA	THROUGHPUT
APEX 20K100EQC240-1 8 VECTORES 8 BITS / VECTOR	904 LEs (22%)	40 M outputs/seg (25 ns/output)
APEX 20K100EQC240-2 8 VECTORES 16 BITS / VECTOR	1688 LEs (41%)	25.3 M outputs/seg (39.5 ns/output)

Tabla 6.6B: Implementación serie en APEX 20KE

BUFFER SERIE	ÁREA	THROUGHPUT
APEX 20K100EQC240-1 8 BITS / VECTOR	24 LEs / ETAPA	71.4 M outputs/seg (14 ns/output)
APEX 20K100EQC240-2 16 BITS / VECTOR	48 LEs / ETAPA	54.9 M outputs/seg (18.2 ns/output)

Tabla 6.6C: Buffer serie en APEX 20KE

La Tabla 6.6B muestra el área y el throughput del circuito serie para el FPGA APEX 20KE, considerando lo descrito en la sección 3.4.3 sobre las normas OC-48, OC-12 y el throughput necesario para el retardo de células, se observa que la Unidad de Clasificación si puede cumplir dicha especificación y que dicho FPGA es la mejor opción para una implementación final.

En la Tabla 6.6C muestra el área y el throughput del buffer serie para el FPGA APEX 20KE, como se describe en la sección 3.4.3 este buffer sirve para transportar los vectores con las direcciones de memoria. Se observa que para ambas normas consideradas OC-48 y OC-12 se pueden tener respectivamente la capacidad suficiente de direcciones de memoria para cumplir con el requisito de CLP (probabilidad de pérdida de células) respetando el throughput necesario.

6.5 Evaluación de los resultados

En la evaluación para las implementaciones particulares consideradas, se toma en cuenta los procesos tecnológicos de fabricación y las características del dispositivo. Las ventajas y limitaciones se basan en el protocolo al cual está dirigida la aplicación y su posible extensión a otros tipos de tráfico.

6.5.1 Comparación de los resultados

En la Tabla 6.7, para la aplicación considerada y sus especificaciones según las normas OC-48 y OC-12 considerados en la sección 3.4.4, se resume los valores teóricos y los resultados de la implementación de la Unidad de Clasificación en la plataforma FPGA APEX 20KE.

UNIDAD DE CLASIFICACIÓN	TIEMPO DE CÉLULA	RETARDO PROMEDIO TEÓRICO	RETARDO REAL CONSTANTE	ÁREA TOTAL
NORMA OC-48	176.123 ns.	440.308 ns.	446.5 ns.	18311 LEs
NORMA OC-12	707.890 ns.	1892.836 ns.	1282.5 ns.	67785 LEs

Tabla 6.7: Implementación de la Unidad de Clasificación

La comparación de rendimiento de las arquitecturas serie y paralelo que conforman la implementación de la Unidad de Clasificación, en este caso en los FPGAs considerados, está directamente relacionado con los procesos tecnológicos de fabricación de los FPGAs, de las facilidades del compilador y también de la estructura interna de los dispositivos. En el anexo B se amplía las principales características de los dispositivos empleados.

Con dicha consideración, a continuación se describen los principales puntos de comparación de las implementaciones:

- Un parámetro que permite valorar la tecnología de implementación es el retardo de los componentes lógicos y el retardo de las líneas de conexión. Concretamente, como se vio en la sección 6.2.1, T(ur) es el atraso que introduce una unidad de retardo y es un buen parámetro para valorar la tecnología de las plataformas FPGA utilizadas. Estos valores muestran que la plataforma de Altera APEX 20KE en la tecnología CMOS de 0.15 micras, es la mejor candidata para la implementación de la Unidad de Clasificación:

  APEX 20K100EQC240-2: T(ur)= 1.12ns
  FLEX EPF10K20RC240-4: T(ur)= 2.9ns
  Xilinx XC4010XLPC84-3: T(ur)= 3.2ns

- En la familia APEX 20KE, el valor de "speed grade" permite mejorar el rendimiento, la comparación de estos valores (-2 y -1) muestran que se puede obtener alrededor de un 15% de mejora total en el throughput en el diseño micropipeline de Unidad de Clasificación. Para la norma OC-48 se considera el "speed grade" –1, para la norma OC-12 es suficiente un "speed grade" –2.

- El compilador utilizado de Altera MAX+PLUS II y QUARTUS II, incorporan una biblioteca completa de LPMs y automáticamente habilitan la opción "carry chain" que permite un menor retardo entre LEs. Se puede usar la opción de "cliques" para agrupar toda la lógica del "critical path" y optimizar el retardo de las líneas de conexión. También el compilador calcula automáticamente los parámetros de retardo y permite valorar el rendimiento del diseño con una suficiente aproximación.

- En la Tabla 6.7 se muestra el área para ambas normas OC-48 y OC-12 en la implementación de la Unidad de Clasificación, en la familia APEX 20KE se pueden tener dispositivos de hasta 51,840 LEs, en la familia STRATIX se tiene capacidades de hasta 114,140 LEs.

6.5.2 Ventajas y limitaciones

Dentro de las ventajas y limitaciones de la Unidad de Clasificación, la cual está compuesta de ambas arquitecturas serie y paralela, se pueden considerar las relacionadas con protocolo de comunicación ATM y por la implementación asíncrona micropipeline de la Unidad de Clasificación.

Dentro el ámbito del protocolo ATM se ha considerado las especificaciones de las normas OC-12 y OC-48, dentro de este marco se puede citar los siguientes puntos:

- La arquitectura asíncrona micropipeline de la Unidad de Clasificación como un subsistema QoS-ATM, permite implementar una gran variedad de esquemas de prioridad, el tiempo de clasificación es independiente del estado de las colas y satisface los requisitos de retardo de células y tasa de pérdidas de células para las normas OC-12 y OC-48. La arquitectura modular permite adaptar la arquitectura a diferentes requerimientos de throughput y latencia.

- El tiempo de procesamiento total para ambas normas, dado el tiempo de célula (slot) y el promedio de retardo está dentro del rango del tiempo real de clasificación constante, estos valores se muestran en la Tabla 6.7. Una observación adicional es que, dependiendo del entorno de la red de comunicación y la aplicación, dicho retardo constante se puede considerar despreciable comparado con los tiempos de máximo retardo admisibles para tener una buena interacción en tiempo real. Las normas consideran retardos máximos para voz de 200ms y para vídeo comprimido de 100ms.

- En la arquitectura de un conmutador de memoria compartida la mayor limitación se encuentra en la memoria, pero los avances en la tecnología de las memorias ofrecen altas frecuencias de acceso y mayores anchos de palabra. Salvo de una consideración de costo económico, se puede considerar resuelto el problema de acceso de memoria en este tipo de arquitectura.

- Aunque el presente trabajo se limita a un conmutador ATM, para el cual el modelo de tráfico es bien conocido, para otro tipo de tráfico por ejemplo IP/ATM, el diseño del conmutador es el mismo en lo que se refiere a la estructura lógica y física. La diferencia radica en la fuente del tráfico y en como sus parámetros estadísticos afectan el diseño. Por lo cual la extensión a otros tipos normas implica un buen conocimiento del modelo de tráfico considerado.

- Para la extensión a otras normas o aplicaciones en las redes de comunicaciones, debido a la complejidad de los distintos factores involucrados en el diseño de los conmutadores es deseable realizar un test de campo del conmutador ATM. Esto permitiría una verificación a escala global y basada en mediciones de campo reales. Las especificaciones como el índice de utilización, la capacidad de células, probabilidad de pérdida de células, el número de puertos, están relacionadas a un nivel del conmutador y con el modelo de tráfico. En forma general el diseño de conmutadores desde un nivel de sistema, se basa en la optimización de ciertos criterios relacionados a parámetros como el throughput, la tasa de pérdida y el retardo de células, además de otros como área, potencia e interfaces.

Dentro ámbito de la implementación asíncrona micropipeline se puede citar los siguientes puntos:

- En esta aplicación específica de un subsistema para la QoS-ATM, el uso de los circuitos asíncronos se presenta como una alternativa y se justifica cuando el diseño del equivalente circuito síncrono presenta severos obstáculos (skew, jitter, sincronización) o cuando los requisitos para el diseño de una arquitectura digital son casi imposibles de implementar mediante circuitos síncronos (velocidad de operación variable, ratios distintos entre los datos en las entradas y en las salidas, bajo EMI, bajo consumo de potencia).

- La metodología de diseño asíncrono micropipeline es independiente de la tecnología, se definen todas las etapas desde la inicial descripción VHDL, layout, verificación e implementación. El método de diseño se basa en herramientas de diseño comerciales lo cual facilita su incorporación dentro del flujo de diseño convencional. La implementación tiene por objetivo validar la metodología. Se

considera plataformas comerciales de FPGAs, para las cuales se aplica la metodología en la implementación QoS-ATM y se corrobora los resultados mediante mediciones de laboratorio.

- En forma general se puede considerar que siempre es difícil una correcta comparación entre circuitos síncronos y asíncronos debido a las diferencias existentes para la implementación óptima. Las diferencias no solo se encuentran en la metodología y las herramientas CAD empleadas, sino en también en la estructura y tipos de celda de las bibliotecas que soportan el proceso de diseño, por ejemplo celdas de control o elemento de retraso usualmente no son ofrecidas en las clásicas bibliotecas síncronas. Actualmente es muy difícil encontrar bibliotecas compatibles que permitan la comparación.

- Con la consideración anterior y como criterio para una comparación síncrono-asíncrono en el presente trabajo, se debe tomar en cuenta que en el diseño micropipeline el data-path es similar para un diseño síncrono pipeline. En un diseño síncrono la frecuencia de reloj se fija para el peor caso y realizada la implementación usualmente se puede operar a una frecuencia mayor que la prevista. Para la arquitectura micropipeline asíncrona el throughput está limitado solo de los parámetros tecnológicos de la plataforma de implementación y siempre se asegura una correcta operación. El área adicional es debido a los latches activados por flancos, los circuitos de control y los bloques retraso. Como se muestra en Capitulo V, estos costos adicionales de área son mínimos para la arquitectura propuesta en una implementación FPGA.

CAPÍTULO VII

CONCLUSIONES

En la presente tesis se ha investigado el estilo de diseño asíncrono micropipeline y su aplicación en el campo del protocolo de comunicaciones ATM. Mediante la metodología de diseño propuesta partiendo desde una especificación VHDL, considerando todas sus fases e incluyendo la biblioteca de módulos, ha permitido la implementación del prototipo demostrador de la Unidad de Clasificación para la QoS-ATM. En la actualidad el diseño de circuitos asíncronos es un área de constante investigación y a lo largo del desarrollo de la tesis, tanto de la metodología y su aplicación en la arquitectura asíncrona de la Unidad de clasificación, se ha realizado la publicación de artículos en diversos congresos [74] [75] [76] [77] [78].

En este capítulo final se resumen las principales conclusiones y perspectivas de la presente tesis, a fin de una mayor claridad se abordan en las tres secciones que a continuación se desarrollan.

7.1 Metodología del diseño micropipeline

Un aporte de la presente tesis es el desarrollo de una metodología de diseño de circuitos asíncronos, dentro de los objetivos y logros importantes alcanzados se puede mencionar:

- Independencia tecnológica: Se ha desarrollado una metodología de diseño asíncrono micropipeline de "matched delays" independiente de la tecnología, en la cual se definen todas las etapas desde la síntesis, layout, verificación e implementación. El método de diseño se basa en herramientas de diseño comerciales lo cual facilita su incorporación dentro del flujo de diseño convencional. La introducción de LPMs en la descripción VHDL permite minimizar la perdida de eficiencia en área de los diseños.

- Módulos de control: Para la metodología de diseño mencionada, se especifica una biblioteca de módulos sencillos de control de comportamiento monótono. El módulo SINCT es un diseño ad-hoc de comportamiento monótono. Los módulos de control elemento-C y RETARDO son diseñados de forma ad-hoc, el elemento-C es implementado mediante un simple LPM_add_sub y el módulo referencia de RETARDO es implementado mediante un elemento-C.

- Método de temporización: Para el cálculo de los matched delays se establece un método práctico para evaluar el retardo de los bloques de procesamiento de datos

basado en los niveles de lógica de cada bloque, y el correspondiente cálculo de un retardo total equivalente tomando como unidad de medida el retardo que introduce un módulo referencia de RETARDO.

- Prototipo demostrador: La metodología de diseño desarrollada es independiente de la plataforma de implementación pero con el objetivo validar la metodología se considera plataformas comerciales de FPGAs. En esta implementación particular se aplica la metodología de diseño y se corrobora los resultados mediante mediciones de laboratorio. Dentro de las mediciones, se desarrolla un procedimiento sencillo basado en instrumentos de laboratorio disponibles y que a su vez dichas mediciones sean lo suficientemente precisas para evaluar las prestaciones de la implementación.

7.2 Aplicación en comunicaciones

La presente tesis enfoca un aspecto importante del protocolo de comunicaciones ATM, cual es la QoS, donde el problema de garantía de QoS es una aplicación aún no resuelta satisfactoriamente. El propósito es explorar la aplicación de los circuitos asíncronos en dicho campo, por lo cual el prototipo implementado intenta mostrar que los circuitos asíncronos son una alternativa frente a los problemas que presentan a los circuitos síncronos. El empleo de arquitecturas asíncronas se justifica cuando el diseño del equivalente circuito síncrono presenta severos obstáculos o cuando las prestaciones requeridas de una arquitectura son casi imposibles de implementar mediante circuitos síncronos.

En el área de aplicación QoS-ATM, y para la implementación asíncrona de la Unidad de Clasificación se puede citar los siguientes objetivos y logros principales:

- Nueva arquitectura: Se propone una nueva arquitectura asíncrona micropipeline para la Unidad de Clasificación del subsistema QoS-ATM de un conmutador de memoria compartida, que cumple con los requisitos de retardo de células y tasa de pérdidas de células para las normas de transmisión OC-48 y OC-12. En el diseño se emplea herramientas CAD comerciales por lo cual la metodología de diseño puede ser empleada fácilmente por los diseñadores. En la literatura especializada no se reporta alguna implementación similar a dicha arquitectura asíncrona y que su diseño se base en herramientas CAD estándar.

- Diseño Modular: El uso de circuitos de clasificación serie y paralelo de forma modular, permite el dimensionado de la Unidad de Clasificación en forma simple. Las interfaces asíncronas entre los módulos de clasificación permiten conectarlos directamente sin problemas de sincronización o alineación de fase. En el caso un equivalente síncrono de la misma arquitectura se tendría el problema de sincronizar las salidas de los circuitos serie, que conforman las diferentes colas y las cuales ingresan hacia el circuito paralelo; en la arquitectura propuesta se resuelve mediante una interfaz simple basado en el elemento-C.

- Gestión de la QoS: La QoS se realiza mediante un circuito ad-hoc y evita el uso de largas tablas de memoria y punteros. El mecanismo de descarte por prioridades de células, implica simplemente remover las posiciones de una lista que contengan la menor prioridad. La Unidad de Clasificación opera basada en un amplio rango de

prioridades para las células ATM, de hasta 2^{16} niveles en OC-12, esto permite mejorar la eficiencia de la QoS para una justa utilización del ancho de banda y latencia.

- Cualidades intrínsecas: El empleo una técnica asíncrona como micropipeline y considerando el modelo aleatorio de tráfico ATM, al nivel de sistema, permite considerar los recursos para las condiciones típicas de operación. La ausencia de la señal de reloj evita problemas de jitter y skew, bajo consumo de potencia y también cumplir con las normas de compatibilidad electromagnética.

7.3 Perspectivas y trabajo futuro

La diseminación de las técnicas de diseño asíncrono seguirá aumentado en el futuro, debido al incremento de grupos de investigación en compañías y universidades. Por otro lado, ATM es una arquitectura madura y su aplicación continuará en expansión en los próximos años, una muestra de su vigencia es el hecho que ATM es una opción para garantizar la QoS en tecnologías de acceso en boga como ADSL (Asynchronous digital subscriber line). En este sentido, la presente tesis tiene perspectivas y trabajo futuro en dos líneas principales. La primera línea es continuar el desarrollo dentro de la metodología de diseño establecida en la presente tesis. La segunda es el desarrollo de un conmutador ATM empleando técnicas asíncronas. A continuación se describen ambas líneas posibles de trabajo futuro.

Desarrollo de la metodología de diseño asíncrono:

- El desarrollo de un entorno integrado de diseño asíncrono micropipeline que permita automatizar todas las etapas del flujo de diseño. Esto implica un tratamiento conjunto de los circuitos de control y los bloques de procesamiento de datos. Es deseable que las etapas de place&route puedan especificarse desde la descripción VHDL. Igualmente que se pueda introducir en el diseño facilidades de DFT.

- La especificación de un conjunto mínimo de bloques de control como una biblioteca de LPMs para aplicaciones asíncronas. En forma unida a los LPMs generales, permitirá una independencia tecnológica sin perder eficiencia en la implementación y mantener un flujo de diseño dentro de las utilizadas por las herramientas EDA comerciales.

- La evaluación de la metodología de diseño en el caso de una implementación ASIC. Implica una verificación que los circuitos de control no presenten hazards. También el comprobar la compatibilidad de las descripciones VHDL-LPMs. Una consideración es el elevado costo del silicio en las tecnologías submicrónicas.

Desarrollo del conmutador ATM:

- Incorporación de subsistemas asíncronos en el diseño. Ello implica un sistema heterogéneo síncrono/asíncrono, donde la señal de reloj solo subsiste en aquellos módulos en los cuales sea imprescindible. Esto abre un escenario amplio de investigación para el diseño global del conmutador ATM y de los diferentes

subsistemas, desde los modelos de simulación basados en la fuente de tráfico hasta las arquitectas de implementación.

- Extensión para IP/ATM. Actualmente ATM es usado para tráfico IP (Internet Protocol). Cada célula IP es dividida en varias otras células ATM de tamaño fijo, el diseño del conmutador es el mismo que los conmutadores ATM en lo que se refiere a la estructura lógica y física. La diferencia y materia de investigación radica en la fuente del tráfico y en como sus parámetros estadísticos afectan el diseño del conmutador.

Finalmente, consideramos que este trabajo de tesis abre una línea de investigación en la aplicación de los circuitos asíncronos, especialmente en el campo de comunicaciones y la problemática de la QoS-ATM. Por otro lado deseo expresar mi agradecimiento al Instituto Catalán de Cooperación Iberoamericana (ICCI) por la beca pre-doctoral concedida.

ANEXO A

En el presente anexo se consideran aspectos adicionales que han servido de soporte en el desarrollo de la presente tesis y ampliación de conceptos. Se agrupan en 4 partes: el protocolo ATM, el modelo de tráfico, el conmutador ATM y las interfaces del conmutador. Se realiza una descripción somera considerando que las fuentes de referencia sobre dichos puntos son variadas y fácilmente disponibles.

A1: ASPECTOS ADICIONALES DEL PROTOCOLO ATM

ATM: MODO DE TRANSFERENCIA ASÍNCRONO

El modo ATM, se define en 1988 con la finalidad de soportar servicios de tiempo real y servicios con tasa de bit variable. Emplea la técnica de multiplexado estadístico por división de tiempo y conmutación de paquetes. Las células de tamaño fijo igual a 53 bytes que permite mejorar la eficiencia y el retardo de empaquetado y el modo de operación de los nodos de conmutación ATM son de circuito virtual. No se realiza control de errores en los nodos ATM a fin de minimizar el procesado de los paquetes. La operación tiende a realizarse mediante hardware con el fin de conseguir mayor rapidez en la conmutación de paquetes.

En ATM, los circuitos virtuales que se establecen son bidireccionales, el establecimiento de un circuito virtual ATM entre una estación de origen y una estación de destino lleva siempre asociado el establecimiento del circuito virtual de sentido inverso. Los principios básicos de ATM son expuestos en la recomendación de I.150 de la CCITT:

- ATM es considerada como un modo de transferencia específicamente orientado a paquetes y basado sobre células de longitud fija. Cada célula consiste de un campo de información y una cabecera el cual es usando principalmente para determinar el canal virtual y realizar el apropiado encaminamiento. La integridad de la secuencia de las células es preservada por el canal virtual.

- ATM es orientada a conexión. A la cabecera se asignan valores para cada sección de conexión y para el tiempo completo de duración de dicha conexión. La señalización e información de usuario son transportadas por un canal virtual separado.

- La información de campo de las células ATM es transportada en forma transparente a través de la red. Ningún tipo de procesamiento como control de error es realizado dentro de la red.

- Todos los servicios (voz, vídeo, datos) pueden ser transportados por ATM, incluyendo servicios que no orientados a conexión. Para acomodar varios tipos de servicios una

función de adaptación es proporcionada que permite incorporar información de todos los servicios en las células ATM y proporcionar funciones de servicios específicos por ejemplo recuperación de reloj, recuperación de células perdidas, etc.

RENDIMIENTO DE UNA RED ATM

Una red ATM que soporta simultáneamente tráfico de voz, audio, vídeo y datos debe tener unas aceptables características de prestaciones para cada tipo de servicio. Por otro lado, dichos servicios tienen diferentes requerimientos de acuerdo a ciertos parámetros, en la Tabla A.1 muestran algunos de dichos parámetros.

TRÁFICO Y DEMANDA	VOZ	ARCHIVOS	TRANSACCIÓN	VIDEO CONFERENCIA	VIDEO DIFUSIÓN
BW PROMEDIO	MUY BAJO	ALTO	BAJO	BAJO	MUY ALTO
BW PICO	BAJO	ALTO	ALTO	ALTO	MUY ALTO
RETARDO	MUY BAJO	ALTO	BAJO	BAJO	MUY ALTO
RETARDO VARIACIÓN	MUY BAJO	ALTO	ALTO	MUY BAJO	BAJO

Tabla A.1: Requerimientos del tráfico

En dicha tabla se observa que los requerimientos no solo dependen del ancho de banda sino de varios parámetros en conjunto. En función de efectos de retardos, tolerancia de error y característica de "burst", cada servicio tiene su propio requerimiento. ATM tiene definido unos parámetros de prestaciones que tienden a mantener un nivel aceptable en la red ATM para todos los servicios arriba mencionados.

PARAMETROS DE RENDIMIENTO ATM

Una red ATM queda completamente caracterizada en cuanto a su rendimiento mediante un conjunto de parámetros, es decir, que encontrando los valores de dichos parámetros a través de mediciones sobre una red ATM es posible hacer un seguimiento y tomar decisiones sobre la red.

Estas decisiones involucran acciones para asegurar la disponibilidad de la red para los usuarios y que las conexiones actuales están recibiendo un servicio adecuado basado en la clase de servicio. Estos parámetros son: cell loss ratio, cell transfer delay, cell insertion rate, severly errored cell ratio y cell transfer capacity. Dentro de los cuales los parámetros negociables son:

- CLR (cell loss ratio), la tasa de células perdidas es la fracción de células perdidas durante la vida de una conexión sobre el total de células transferidas.
- CTD (cell transfer delay), tiene dos componentes: la CTDmax referido a los retardos experimentados por las células transferidas durante la conexión y la CDVp-p (peak-to-peak cell delay variation) mide la variabilidad máxima del retardo, es la diferencia entre el CTDmax y el retardo fijo.

CONTROL DE TRÁFICO ATM

El control de tráfico se refiere a la necesidad de las redes de ATM de monitorizar el tráfico de las células que entran en la red asegurando que la red es aún capaz de establecer para los usuarios una determinada conexión con sus parámetros de rendimiento requeridos. La red ATM debe tener la capacidad de disminuir la tasa con el cual las células entran en la red como también poder descartar células que ya hayan entrado en la red. La tarea de control de tráfico in cualquier red es una función crucial de la administración de la red. Para el control del tráfico se implementan un conjunto de funciones como: priority control, connection admision control, usage parameter control, congestion control.

CALIDAD DE SERVICIO QoS

La red ATM debe aceptar una llamada solo si red tiene los recursos para entregar al usuario una requerida QoS de principio a fin dentro de la red ATM. El terminal nodo local transmisor puede tener los recursos necesarios y de igual forma el terminal nodo local receptor pero si a lo largo de cada tramo interno de los nodos ATM no son capaces de proporcionar la requerida QoS se concluye que dicha conexión no puede ser aceptada.

El aspecto más crítico de la QoS es determinar las características de la fuente de tráfico, es decir, que es lo que se requiere de la red. Esta fuente puede ser caracterizada por cuatro parámetros:

- El promedio de bit rate con la cual la fuente opera, el cual es una media aritmética en un intervalo de tiempo.

- El pico de bit rate al cual la fuente es capaz de enviar. Se define “burst ratio” como la tasa entre el pico y el promedio, el cual puede variar desde 1 a 100 ó 1000 dependiendo si es voz o vídeo comprimido o datos.

- Bit rate físico del enlace desde el usuario hacia el nodo local ATM.

- Duración del pico es una medida de como de largo puede ser el pico de bit rate de la fuente. Esto permite determinar el número máximo de células que pueden entrar en la red desde una conexión dada en cualquier intervalo de tiempo.

Con estos 4 parámetros y el tipo servicio del usuario, la red ATM puede decidir si la conexión puede ser garantizada o no en la red. En el tiempo de establecimiento de conexión (connection setup time) usualmente se negocian estos parámetros entre el usuario y el nodo local de la red. Estos parámetros describen el perfil del tráfico de la fuente. El usuario acepta generar tráfico dentro de un marco definido de características (conformat) y la red debe transportar dicho tráfico respetando la QoS especificada. Los mecanismos de Gestión de Tráfico son los encargados de hace valer los Contratos de Tráfico de aquellas conexiones en curso y garantizan la QoS aquellas conexiones que respetan su perfil de tráfico.

CATEGORÍAS DE SERVICIO ATM

Las categorías de servicio relacionan las características del tráfico y los requerimientos de QoS, hacia el comportamiento de la red. Existe un rango en el comportamiento de la red que le permiten cumplir con las especificaciones del contrato como control de admisión de conexión (CAC), el encaminamiento y la asignación de los recursos de la red. Funciones como el planeamiento y el control de congestión en los elementos de la red pueden también contribuir a un servicio justo y individualizado las fuentes de tráfico.

ATM Forum considera diferentes categorías de servicio dependiendo de las diferentes aplicaciones y tipos de tráfico:

- CBR (constant bit rate)

Denominado Clase A, sirve para aplicaciones en tiempo real como voz y vídeo que requieren valores determinados de retardo y variación de retardo. Se especifican mediante parámetros como: PCR (peak cell rate), CDV (cell delay variation), CLR (cell loss ratio), CTD (cell transfer delay).

- VBR (variable bit rate)

Sirve para aplicaciones de vídeo comprimido o vídeoconferencia, presentan un comportamiento tipo burst. Se divide en servicios en tiempo real (VBR-rt) denominado Clase B, y servicios de tiempo no real (VBR-nrt) denominado Clase C. El servicio VBR-rt puede transmitir a varias tasas, permite el uso de ancho de banda de acuerdo a demanda y se especifican mediante PCR, CDV, CLR, CTD, SCR (sustained cell rate) y BT (burst tolerance). El servicio VBR-nrt se define mediante PCR y CLR.

- UBR (Unspecified bit rate)

Se considera dentro de la Clase D, diseñado para aplicaciones tolerantes a retardos y aplicaciones en tiempo no real. Se basa en un servicio tipo “best effort”, no se especifica ningún parámetro.

- ABR (available bit rate)

Se considera dentro de la Clase D, diseñado para aplicaciones que tienen la habilidad de ajustar su tasa de transferencia de acuerdo a una capacidad reservada en la red, sirve para tráfico LAN tipo burst. Las aplicaciones no deben ser sensibles al tiempo de retardo. Se especifican mediante el mínimo (MCR), el pico (PCR) y el permitido (ACR) de la tasa de células respectivamente, además del CLR.

- GFR (guaranteed frame rate)

Este servicio es intermedio entre ABR y UBR en cuanto a prioridad y proporciona QoS al nivel de tramas. El CLR es el único parámetro para este servicio.

CONTROL DE PRIORIDAD

Las redes de ATM se caracterizan por dos formas de mecanismos de prioridad, la memoria de almacenamiento para ambos mecanismos es organizada dependiendo del algoritmo de prioridad implementado:

- El mecanismo de prioridad de espacio.

El mecanismo de prioridad de espacio permite que el conmutador sea capaz de operar con una baja CLP solo para las células que lo requieran, esto permite una mejor administración de la carga de tráfico que puede ser admitida por la red. Este mecanismo decide la operación a diferentes niveles de pérdida de células.

Las normas de ATM establecen, en la cabecera de la célula, un bit de prioridad de pérdida de célula. Una alta prioridad es indicada por el valor cero, para una baja prioridad el bit es puesto a uno. La prioridad de espacio decide si una célula es admitida en el buffer de capacidad finita.

Si el buffer llega a su límite de almacenamiento, se debe descartar células de forma selectiva células de baja prioridad para poder mantener los objetivos de rendimiento requeridos para tráfico de alta y baja prioridad. Por ejemplo en vídeo comprimido se puede usar una alta prioridad para información de sincronización y evita la necesidad de operar a niveles bajos de CLP para todas las células en la conexión.

- El mecanismo de prioridad de tiempo.

La prioridad de tiempo administra el orden en el cual las células son atendidas por el servidor y son transmitidas, está ligada con la prestación de retardo. Los diferentes niveles de prioridad no son explícitos en las normas ATM, una forma de establecer puede ser la asignación de diferentes valores o rangos de niveles de prioridad de tiempo para cada VPI/VCI.

El efecto de la prioridad de tiempo es disminuir el retardo para tráfico de alta prioridad a expensas de aumentar el retardo para tráfico de baja prioridad. En ATM el tráfico de conexiones de tiempo real como voz o vídeo interactivo pueden ganar velocidad a expensas de retardar células que no tienen restricciones de tiempo real como un tráfico de datos.

En el presente trabajo se tiene ambos mecanismos: control de prioridad de espacio y prioridad de tiempo, es una combinación de un sistema de cabeceras de cola para prioridades de retardo y un sistema de prioridad de descarte para pérdida de células. Para la prioridad de retardo, la célula de más alta prioridad es atendida primero hasta que exista al menos una célula en el buffer. El sistema de pérdidas define como las células son descartadas cuando el buffer está lleno. Nuevas células, que arriban cuando el buffer está lleno, pueden ser admitidas solo si se descarta células del buffer que tengan baja prioridad de pérdidas, sino las nuevas células se pierden.

A2: ASPECTOS ADICIONALES DEL MODELO DE TRÁFICO

BLOQUEO DE CÉLULAS

Las colas FIFO tienen el problema de bloqueo HOLB (Head of Line Blocking), donde las células solamente pueden ser transmitidas desde la cabecera de cada cola, células posteriores son bloqueadas aunque los puertos destino estén disponibles. El bloqueo HOL limita el throughput al 58%.

En el presente trabajo se considera una gestión en la cual las células que arriban son organizadas en colas basadas en la información de los bits de prioridad (2^{16} bits para OC-12) que contiene la información de VC, de la categoría de servicio y de destino. La información por destino elimina el HOLB, la información por VCs habilita la asignación de ancho de banda y la información de categoría de servicio permite la QoS.

Igualmente, dependiendo si los recursos lo permiten, se considera el uso de un buffer en cada puerto de salida. Esto es posible ya que la velocidad de la Unidad de Clasificación permite que más una célula pueda ser conmutada en un mismo slot por cada puerto de salida.

TEORÍA DE COLAS

En las redes de comunicaciones los competidores pueden ser células, grupos de células o conexiones. Para el caso de un conmutador se asume que los competidores son células ATM, se puede formalizar matemáticamente con idea de una competencia por recursos. Las células arriban al sistema de cola con necesidad de una cierta cantidad de servicio, y en caso no esté disponible inmediatamente esperan por dicho servicio en algún medio almacenamiento. Luego de un tiempo de espera son atendidos y salen del sistema.

Se tienen los siguientes parámetros en una cola:
λ : Promedio de número de arribos por unidad de tiempo (mean arrival time) o es el promedio de tiempo entre arribos (mean inter-arrival time).
μ : Tasa de servicio a las células.
s : Tiempo requerido para atender a una célula.
w : El número promedio de células en la cola esperando para ser atendidos.
t_w : Promedio de tiempo de espera en la cola.
q : El número promedio de células en el sistema esperando o siendo atendidos.
t_q : Tiempo promedio que la célula espera en el sistema
ρ : La utilización de un servidor de cola, es la fracción de tiempo que el servidor está ocupado.

Se tienen algunas relaciones básicas que se cumplen asumiendo que la capacidad del sistema es infinita, dichas relaciones son independientes de los patrones de arribo y servicio o del número de colas o el tipo de atención en la cola:
La utilización para un solo servidor de cola es: $\rho = \lambda s$.
La formula de "Little" es: $w = \lambda t_w$
También: $q = \lambda t_q$
$q = w + \rho$
$t_q = t_w + s$

Las células arriban durante el slot de tiempo, el instante exacto que arriban no es importante pero se asume que todos los arribos en un slot ocurren antes del instante de despacho de la célula en servicio en el actual slot de tiempo. Es decir, se asume que si una célula arriba en el slot n, será transmitido durante el slot (n+1) y no antes.

Debido al empleo de circuitos síncronos en ATM cuando el buffer está vacío se transmite una trama de células vacías, esto permite mantener la sincronización del servidor y establece un comportamiento determinístico. En muchos análisis lo anterior no es tomado en cuenta y se asume que la célula entra en servicio inmediatamente que ingresa en un buffer vacío en lugar de esperar el inicio del próximo slot libre. Para una carga determinada la operación asíncrona puede mejorar el retardo y pérdida de células y el que modelo de tráfico puede ser más aproximado al tráfico real. Además de evitar el gasto de recursos en sincronización.

En el comportamiento de una cola en el buffer de un conmutador ATM se puede considerar que tiene los siguientes elementos principales:

- El componente de escala de célula: Es el único presente para tráfico aleatorio o para una combinación de tráfico CBR.

- El componente de escala de burst: Se considera presente si el tráfico mixto incluye fuentes tipo “burst”, donde las combinaciones los estados activos pueden exceder la tasa de slot de célula que le conmutador puede admitir.

El buffer debe ser capaz de administrar el componente de escala de célula, para el caso de componente de escala de burst se pueden considerar opciones como:

- Restringir el número de fuentes burst, de esta forma la tasa total de entrada casi no excede la tasa de slot de célula y cualquier exceso de la tasa de células es perdido. Esta opción de escala de burst perdido se denomina “rate envelope multiplexing”. El factor límite es el retardo a través del buffer especialmente para servicios interactivos.

- Asumir el buffer tiene una capacidad suficiente como administrar el exceso de tasa de células, una parte puede ser descartada y la otra es retardada en el buffer. Esta opción se denomina “rate sharing statistical multiplexing”. Se puede segregar el tráfico a través de buffers separados, un buffer corto para células sensibles a retardos y otro buffer mayor para las células sensibles a pérdidas e incorporar un esquema de prioridades de servicio. El presente trabajo se encuentra dentro de esta opción y con las consideraciones mencionadas anteriormente.

Como valores de referencia, por ejemplo para tener una buena interacción en tiempo real en enlaces de aplicaciones reales se tiene: el retardo máximo para voz es de 200ms y en vídeo comprimido se puede tolerar un retardo máximo de 100ms, también el retardo de propagación en una dirección para un satélite en órbita geoestacionaria es cerca de 125ms. En todos los casos, el retardo de conmutación es despreciable en comparación a dichos valores.

MODELO DE TRÁFICO

El modelo de tráfico de forma general puede ser descrito mediante dos procesos aleatorios:

1) El proceso que gobierna el arribo de las células en cada slot de tiempo.

2) El proceso que describe la distribución por la cual las células que arriban eligen los puertos de destino.

MODELO DE TRÁFICO UNIFORME

Se puede aproximar los diferentes tipos de tráfico reales a modelos teóricos y realizar el análisis de sus parámetros, para los principales tipos de tráfico se pueden tener modelos simples. A continuación se describe el modelo de tráfico uniforme.

Una forma de representar la información de tiempos de un proceso de arribo es contar el número de arribos en un intervalo de tiempo definido. La aproximación en tiempo continuo es un proceso de Poisson que es una distribución de tiempo entre arribos tipo exponencial negativa:

$$Pr(k) = (\lambda T)^k \, e^{-\lambda T}/k!$$

Pr (k): Representa la probabilidad de k arribos en un tiempo T.
λ: Es la tasa de arribos.

El proceso de Poisson representa las células que arriban de una fuente hacia el buffer, con una tasa de λ células por slot de tiempo. En el caso de una distribución Poisson la duración T es un slot de tiempo.

En tiempo discreto, el tiempo entre arribos tipo geométrico es un proceso de Bernoulli y si se considera más de un slot de tiempo el número de arribos está binomialmente distribuido:

$$Pr(k) = (1-p)^{N-k} \, p^k \quad N!/(N-k)! \, k!$$

Pr(k): Representa la probabilidad de k arribos en N slots de tiempo.
p: Es el promedio del número de arribos por slot de tiempo.

Las células arriban a los puertos de entrada de acuerdo a procesos independientes e idénticamente distribuidos tipo Bernoulli, cada uno con un parámetro p ($0 < p <= 1$), significa que en un puerto de entrada y en un slot de tiempo dado, la probabilidad que arribe una célula es p, p es la carga de entrada o tasa de arribo para cada puerto.

El destino es elegido de forma uniforme e independiente entre los N puertos de salida, la probabilidad que un puerto de salida sea elegido es 1/N. La probabilidad que una célula llegue a una determinada salida durante un slot de tiempo es p/N, se considera que en un slot de tiempo pueden llegar a diferentes líneas de entrada más de una célula y cuyo destino sea la misma salida.

TRÁFICO TIPO "BURST"

Los servicios que incluyen voz, vídeo y datos tienen una fuerte correlación entre las células que provienen de la misma fuente, originando un tráfico tipo "burst". Se generan células con un pico o muy cerca al pico durante cortas duraciones, y queda casi inactivo el tiempo restante.

Un modelo simple es On/Off, donde la fuente alternativamente tiene un periodo activo o "busy" y otro de inactivo o "idle". La longitud, en slots de tiempo, del periodo activo está distribuido geométricamente. Se puede estimar la probabilidad que el periodo activo dure un tiempo en número de slots y el promedio de la duración del "burst". Se asume que el "burst" al menos tiene una duración de un slot.

Igualmente el periodo inactivo está distribuido geométricamente. Todas las células que pertenecen al mismo "burst" tienen como destino el mismo puerto de salida el cual es elegido de forma independiente de los otros "burst".

La fuente ON/OFF es solo un ejemplo particular de un modelo basado en estados, en el cual la tasa de arribos es constante, hay solo dos estados y el periodo de tiempo en un estado es una distribución exponencial negativa, geométrica o arbitraria.

Se puede generalizar para incorporar N estados, con tasas constantes en cada estado. Se denomina modelos multi-estados o procesos determinísticos modulados, son útiles para modelar por ejemplo tráfico de una fuente de vídeo.

Si los periodos de los estados tienen distribuciones arbitrarias, el proceso se denomina GMDP (Generally Modulated Deterministic Process). Si la duración de los estados son distribuciones exponenciales se denomina MMDP (Markov Modulated Deterministic Process), cada estado produce un número de células geométricamente distribuidas durante el periodo de cualquier estado.

No se restringe a un valor constante de tasa de arribo en cada estado, si el proceso de arribo es un proceso de Poisson y el promedio de la distribución de Poisson es determinado por el estado del modelo, se tiene un proceso MMPP (Markov Modulated Poisson Process) que es útil para representar un proceso de arribo de células añadido. La transición al término de un estado a otro estado esta gobernado por una matriz de probabilidades de transición.

MODELO DE TRÁFICO DE ARRIBO EN LOTES

En algunos análisis se asume procesos "batch-arribal" donde en lugar de un arribo simple con una probabilidad dada, se tiene un lote de células y el número de células en el lote puede tener cualquier distribución. Si para el buffer en un conmutador ATM, se tiene un lote de arribos procedentes de diferentes puertos hasta un máximo determinado durante un slot de tiempo, se puede considerar el número de arribos como las células en el lote durante dicho slot.

El proceso de Bernoulli con arribo en lotes es caracterizado por tener una independiente e idéntica distribución del número de arribos por periodo discreto. Igualmente se define la distribución del número de células en el lote que arriba en un slot de tiempo, de igual

forma se puede definir la distribución global del número de arribos por slot. Esta es una forma general que puede ser usada, por ejemplo, para distribuciones de Poisson y Binomial.

NOTACIÓN DE KENDALL

En la notación de Kendall A/B/X/Y/Z se considera:

A: Especifica la distribución de tiempo entre llegadas (M: es un proceso tipo Markov sin memoria, el tiempo entre arribos es tipo exponencial negativo).
B: Especifica la distribución de tiempo de servicio. (D: es determinístico, el tiempo de servicio es fijo).
X: Especifica el número de canales de servicio (1: significa un solo servidor).
Y: Especifica la capacidad del sistema (se omite cuando la capacidad es infinita).
Z: Especifica la atención de la cola (se omite cuando la atención es primero en llegar primero en ser atendido).

EL MODELO M/M/1

En un sistema de cola M/M/1, se asume que el patrón de arribos es un proceso Poisson, los tiempos entre arribos puede ser aproximado por una exponencial negativa y se cuenta el número de arribos en un intervalo de tiempo. El tiempo de servicio es descrito por una distribución exponencial negativa, pero debido a que ATM usa células de longitud fija y por lo tanto tiempos de servicio fijos, se tiene alguna imprecisión. En un sistema de colas M/M/1, se asume un buffer infinito.

Se puede estimar el promedio del número de células en el sistema "q" que está dado por: $q = \rho/(1-\rho)$

El tiempo de espera promedio en una cola M/M/1, depende de la carga "ρ" y el tiempo de slot de célula "s" para el enlace dado:
$t_q = s + \rho s/(1-\rho)$

La manera de estimar el buffer necesario es considerando la CLP (cell loss probability) requerida en ATM que es de 10^{-8} o menos. En [71] se demuestra que una cola finita puede aproximarse mediante una cola infinita y facilitar los cálculos.

EL MODELO M/D/1/K

En este modelo, para el tiempo de servicio se considera el hecho que las células ATM tienen longitud fija y que entran en servicio cuando el servidor está libre y que no esperan por el próximo slot de célula. El patrón de arribos es un proceso Poisson.

El máximo retardo es la capacidad del buffer multiplicada por el tiempo de slot de célula "s" para el enlace dado. El retardo promedio depende de la carga ρ y el tiempo de slot de célula "s" para el enlace dado. Se calcula usando la formula para un sistema M/D/1 infinito que es muy aproximado al retardo promedio de una cola finita M/D/1, debido a que la pérdida de células es muy baja:
$t_q = s + \rho s/2(1-\rho)$

En [71] se analiza el tiempo de espera promedio versus la utilización para ambos modelos de cola. El tiempo de servicio de célula es dominante hasta un 60% de utilización. También muestra que el retardo promedio se incrementa sustancialmente a partir un 80% de carga.

EL MODELO M/G/1/K

En [72] para el modelo M/G/1/K usa un modelo de servicio atención general y puede ser adaptado para diferentes aplicaciones. Este modelo es ampliado para el análisis de varios tipos de tráfico ATM.

Se asume que las llegadas son de tipo Markov cada uno con parámetro λ de tasa de arribo, se recibe todas las clases de servicio con parámetros de función de densidad de probabilidad g(y) y función de distribución para el tiempo de servicio G(y). La capacidad de memoria máxima es B y es compartida por todas las clases.

Se considera dos clases de servicio i, j donde i>j. El Modelo A asume que la clase i tiene mayor prioridad de pérdida de células y también mayor prioridad de retraso de célula respecto a la clase j. El modelo B asume que la clase i tiene mayor prioridad de pérdida de células y una menor prioridad de retardo de célula respecto a j.

El análisis considera varias clases de prioridad, por ejemplo CBR y rt-VBR tienen ciertos requisitos de prioridad que difieren de la prioridad de nrt-VBR y estos a la vez difieren para tráfico ABR. Por lo cual se evalúa para 3 clases de prioridad ABR (X_1), nrt-VBR (X_2), CBR/rt-VBR (X_3) para estimar las características de la cola de prioridades y la dimensión del buffer.

Se muestra que las diferencias de los modelos son más significativas cuando la CLP es igual a menor que 10^{-10}. Los valores de CLP alrededor de 10^{-4} dan resultados que concuerdan con los resultados analíticos.

EL MODELO D-BMAP

El modelo D-BMAP (Discrete-time Batch Markovian Arrival Process) [37], es un modelo flexible que puede ser usado tanto para tráfico ATM y tráfico IP o una mezcla de ambos tráficos.

El sistema en su conjunto puede ser modelado como un sistema de cola D-BMAP/D/1/B, donde el tiempo de servicio es igual a un slot de tiempo y la capacidad del buffer es finita e igual a B células ATM. La elección de parámetros al nivel de sistema no afecta el modelo de tráfico y se asume que patrón de tráfico es conocido, influyen los tipos de flujos de tráfico y sus intensidades.

El modelo D-BMAP tiene la ventaja de representar un punto del proceso que puede ser usado como una entrada del proceso, independientemente del sistema de colas. También produce aproximaciones razonablemente precisas para un gran rango de parámetros del sistema.

Formalmente el modelo D-BMAP, puede ser definido como un proceso de Markov de tiempo discreto bi-dimensional, “m” estados en la cadena Markov de modulación y una

matriz de transición. Dos vectores de variables aleatorias representan el número de células que arriban en un slot de tiempo y estado de la cadena Markov.

Las estadísticas los tráficos ATM y/o IP se usan para derivar la matriz de transiciones del modelo D-BMAP. Para derivar la distribución de la ocupación del buffer y la CLP, se usa una solución denominada "matrix-geometric".

En la arquitectura del conmutador de memoria compartida, el único puerto de salida del multiplexor se trata junto a cola lógica asociada a dicho puerto. Asumiendo que el tráfico destinado para diferentes puertos de salida es independiente, se puede calcular la probabilidad de ocupación del buffer compartido para un conmutador de N puertos de salida.

TRÁFICO MULTICAST

Una característica importante en un conmutador es la capacidad de administrar células multicast. La célula multicast es transmitida a varios puertos de salida de forma simultánea. El tráfico multicast permite comunicaciones punto-multipunto, una ventaja de un conmutador de memoria compartida es que permite una fácil implementación de esta característica.

Se tienen varios esquemas para implementar esta característica en el conmutador, se puede tener una cola separada para células multicast. Se lee una sola vez y se envía a todos los puertos de salida. Solo se procesa una célula multicast en un slot de tiempo por lo cual el throughput de los canales multicast es limitado y se incrementa la tasa de llamadas multicast, el throughput del conmutador disminuye.

Otro posible esquema es administrar de manera igual las células unicast y multicast, las direcciones de las células multicast son divididas y copiadas antes de ingresar a la cola, por lo que no existe degradación con el incremento la tasa de las células multicast.

En presente trabajo, debido a la arquitectura de la Unidad de Clasificación, se puede considerar un puerto de entrada de reinserción. Donde para el tráfico multicast se divide cada slot en una fase de dispersión y una fase de reinserción, siempre los conflictos son resueltos mediante los valores de prioridad y las células de menor prioridad pueden volver a ser puestos en la cola de reinserción, detalles de la implementación quedan como un trabajo futuro dentro del desarrollo completo del conmutador ATM.

A3: ASPECTOS ADICIONALES DEL CONMUTADOR ATM

EL CONMUTADOR DE MEMORIA COMPARTIDA

De una manera general se tienen las siguientes categorías para los conmutadores ATM:

- Space-division.

En este tipo de conmutadores se puede establecer múltiples caminos concurrentes entre los puertos de entrada y salida, lo que permite transmitir muchas células simultáneamente a través del conmutador. Se construyen con SEs (Switching Elements)

construidos de manera idéntica y conectados en una topología específica mediante enlaces. Por ejemplo dentro de esta categoría se tienen los conmutadores tipo Crossbar, tipo Banyan.

- Time-division.

Se caracterizan por que las células transitan a través de un recurso que es compartido por todos los puertos de entrada y salida. El recurso puede ser una memoria común, un medio compartido como un bus o anillo. La capacidad el conmutador está limitado por el ancho de banda del recurso compartido.

El conmutador de memoria compartida (shared-memory), se puede ser considerado dentro de esta categoría.

La arquitectura "shared-memory" es un conmutador MxN. Cada puerto de entrada y salida tiene una capacidad de manejar V células por segundo. Todas las colas lógicas usan una misma memoria y estas colas son implementadas mediante listas de enlace y son administradas por un controlador de memoria. Cada lista de enlace representa una cola lógica asociada con un puerto de salida.

Para una concreta CLP (Cell Loss Probability), esta arquitectura tiene la ventaja de requerir un reducido tamaño de buffer por cada puerto de salida debido a que se comparte los recursos entre todos los puertos. También se puede considerar como un puerto de salida y una arquitectura de buffer que garantiza un throughput determinado siempre y cuando los circuitos asociados lo permitan.

El principal problema de esta arquitectura es que la frecuencia de acceso a la memoria compartida tiene que ser mayor o igual a (M+N)V. Esto se resuelve dividiendo la memoria en K bloques accesibles desde todas las entradas y salidas lo cual permite que cada bloque soporte solo una frecuencia de acceso de N operaciones por segundo.

Es decir, que en el peor caso las salidas acezan a un solo bloque de manera simultánea mientras que las entradas pueden ser controladas para almacenar en bloques de memoria desocupados. De esta forma la frecuencia de acceso de la memoria es independiente del número de entradas con la condición de que el número de puertos entradas y salidas cumpla:

$$M+N <= NK$$

El conmutador puede ser implementado en S slices en paralelo para soportar mayor throughput y proporcionar tolerancia a fallos. Cuanto más slices se usan, menor es el ancho de la palabra necesaria y mayores frecuencias de acceso se logran si se asume ciertos requerimientos para los slices de memoria.

Los avances en la tecnología de las memorias ofrecen altas frecuencias de acceso y mayores anchos de palabra. Por lo cual se puede considerar resuelto el problema de acceso de memoria en este tipo de arquitectura.

Los avances en la tecnología de memorias permiten capacidades para altos tráficos, por ejemplo SDRAM-DDR puede proporcionar hasta 3.2 Gbps. Por ejemplo la SRAM

IBM 041811TLAB-7 de 64Kx18, cuando opera en modo pipeline síncrono, tiene un tiempo de ciclo de 7.0ns.

Las DRAM ofrecen similar throughput pero alta capacidad por unidad de costo. La tecnología Rambus [69] ofrece alto throughput y bajo costo por pin, tiene un throughput pico de 12.8 Gbps por chip de memoria (RDRAM).

A4: INTERFACES AL NIVEL DE CONMUTADOR

PCI

El bus PCI nace en 1990 de la necesidad de INTEL para desarrollar un bus de alta para los procesadores Pentium. La versión 2.0 fue terminada en 1993. Máxima tasa de transferencia de 264 Mbytes/seg. Independiente del procesador. Arbitraje síncrono centralizado. Bajo costo, dimensiones pequeñas. Todas las transacciones en el bus son realizadas de manera síncrona, es decir, cada evento de la transacción está asociado al flanco de bajada del ciclo del reloj. Cada dispositivo en el bus PCI tiene un único par de señales de pedido REQ y confirmación GNT cableados a un árbitro central.

FUTUREBUS+

Es un bus normalizado de alto rendimiento desarrollado por la IEEE, la versión inicial fue publicada en 1987. Las normas 896 incluyen la capa lógica, la capa física, recomendaciones de su uso y numerosos documentos de referencia. El bus ha sido usado como bus local de procesador y memoria, también ha sido usado para periféricos de alta velocidad. El bus intenta proporcionar flexibilidad y una amplia funcionalidad. Está orientado a sistemas de alto rendimiento y alto costo en consecuencia. Máxima tasa de transferencia de 3Gbits/seg. Arbitraje asíncrono centralizado o distribuido. No está aún ampliamente difundido.

INTERFAZ SONET

ATM forum establece las normas del transporte de las células utilizando las interfaces físicas existentes como E1, DS1, E3, DS3, etc. La capa física en la RDSI-BA tiene la siguiente estructuración de capas:

- Subcapa de Convergencia de la Transmisión (TC). Se encarga de transformar el flujo asíncrono de las células que le entrega la capa ATM en un flujo continuo de bits que entrega a la siguiente subcapa.

- Subcapa dependiente del Medio Físico (PMD). Se encarga de transmitir los bits que le entrega la subcapa TC a través del medio físico.

Para SONET STS-3 (Synchronous Transport Signal) ATM Forum establece las siguientes funciones para la subcapa TC, para SONET STS-12 se tiene las mismas características que la anterior pero a 622.08 Mbits/s.

- Control de errores de cabecera. La generación de los bits de paridad como la detección/corrección de errores en la cabecera reside en la subcapa TC de la capa física.

- Desacoplamiento de la tasa de células. Debido a que las células son entregadas a la capa física de forma asíncrona. Para su transmisión por parte de la subcapa PMD es necesario transformar esta secuencia en un flujo continuo de bits. Para ello, el UIT-T especifica la inserción de células sin asignar (unassigned cells), para los cuales se reserva VPI/VCI = 0/0.

- Generación de la trama de transmisión. SONET STS-3 ó OC-3 (Optical signal), son equivalentes a la trama SDH STM-1 (Synchronous Transfer Module). La trama STM-1 tiene una duración de 125 microsegundos, una disposición de 9 filas de bytes por 270 columnas de bytes, que es una capacidad de 155.52 Mbps (9x270x8 /125). La trama tiene bits de cabecera o tara de sección (Section Overhead, SOH) que cumplen funciones específicass de SDH, el campo de datos que es la cabida útil de 261x9 filas (Payload), punteros para la función de multiplexado. Una vez desacopladas las células ATM mediante células sin asignar, se transportan en la cabida útil.

- Aleatorización de las células. Para evitar que la presencia de secuencias de bits en el campo de datos de las células que conduzcan a una delimitación errónea de las células, se ha prescrito un procedimiento de aleatorización del campo de datos.

- Delimitación de la célula. La célula ATM no tiene ningún campo de sincronismo de célula, esto es ningún indicador de inicio ni de fin de la célula. El número y los límites de las células ATM transportadas en una trama STM-1 son transparentes a la capa física. Para delimitar el inicio de cada célula en el flujo combinado de células ATM y de células por asignar, se ha prescrito la utilización de un algoritmo que hace uso del campo HEC. El receptor establece una hipótesis sobre el inicio de la célula ATM y comprueba, bajo la suposición de ausencia de errores en la transmisión, si los HEC corresponden con los calculados a partir del resto de los bits de cabecera. De esta forma se delimitan las células si se producen “n” correspondencias sucesivas. Existen C.I. específicos para ayudar a realizar dichas tareas por ejemplo en [73] se tiene un convertidor serie a paralelo para sistemas de transmisión OC-12.

INTERFAZ UTOPIA

UTOPIA (Universal Test & Operations PHY Interface for ATM) es una interfaz entre la capa física y la capa ATM. Se tienen dos tipos de interfaces nivel 1 y nivel 2.

Nivel 1: Especifica la interfaz entre la capa ATM y una simple capa física PHY para 155.52 Mbits/s (OC-3/STM-1). Usa un bus de datos de 8 bits o de 16 bits y opera a 25 Mhz.

Nivel 2: Especifica la interfaz entre la capa ATM y una simple capa física PHY para 622.08 Mbits/s (OC-12/STM-4).

ANEXO B

En el presente anexo se describen los aspectos técnicos más relevantes de los FPGAs empleados tanto de XILINX y ALTERA. Se intenta resumir los datos técnicos de las referencias y que son utilizados a lo largo del trabajo. Se describen aspectos sobre el consumo de potencia y, además, incorporar datos actualizados sobre algunos de los FPGAs recientes.

ESPECIFICACIONES TÉCNICAS DE LOS FPGAs

XILINX

La figura B1 muestra el diagrama lógico de un CLB (configurable logic block) y la figura B2 muestra la configuración interna del FPGA XC4010.

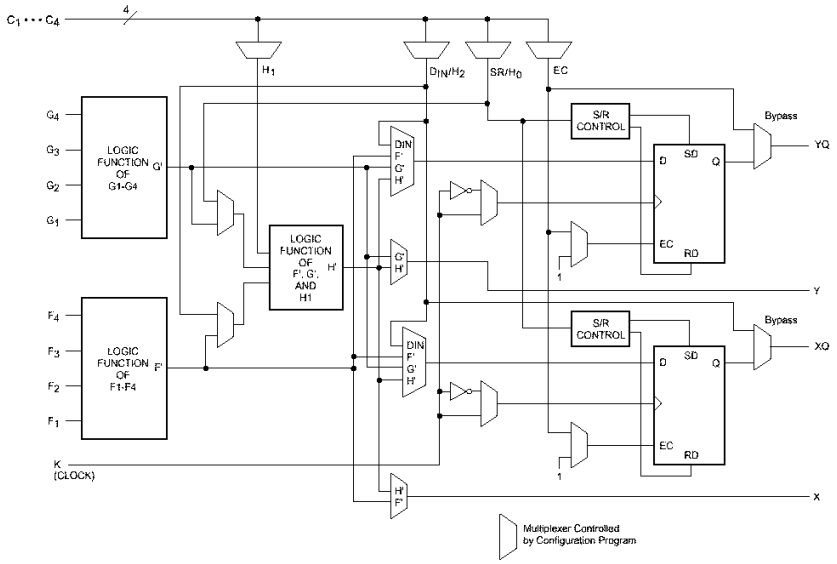

Fig. B1: Diagrama lógico de un CLB XC4010

En la familia XC4000 cada CLB consiste de 3 generadores de funciones, 2 FFs y varios multiplexores. Los generadores de funciones pueden implementar cualquier arbitraria función boolena para sus4 entradas, tienen entradas y salidas independientes y pueden ser tratados independientemente, se implementan como LUTs (lookup table), por lo tanto el retardo de propagación es independiente de la función que realizan. Cada CLB contiene circuitos dedicados para funciones aritméticas, un CBL puede ser usado para implementar dos funciones arbitrarias independientes de hasta 4 variables, o una función arbitraria de 5 variables o cualquier función de 4 variables junto a algunas funciones de 5 variables o algunas funciones de hasta 9 variables. Tanto el número de CLBs y el retardo respectivo son minimizados implementado funciones que ocupen al

máximo cada CLB, de esta forma se incrementa la densidad y la velocidad. Xilinx XC4000 permite que los latches configurados sean agrupados de forma relativa uno al lado del otro.

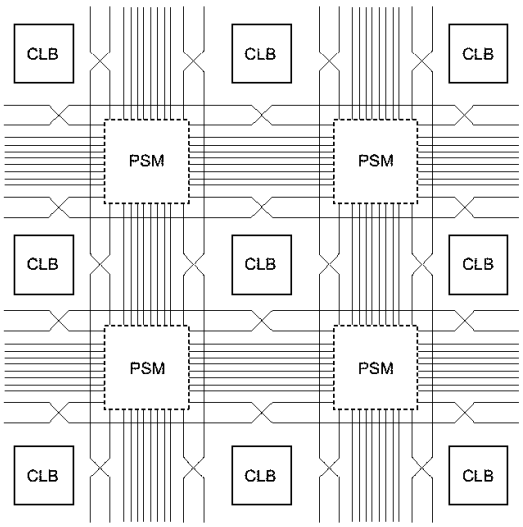

Fig. B2: Configuración interna del FPGA XC4010

Las herramientas de place&route permiten ubicar los componentes muy cerca uno de otro. El compilador FPGA de Xilinx estima los retardos de conexión mediante el fanout de las líneas, los bloques Programmable Switch Matrices (PSMs) realizan la conexión de los CLBs, esta estructura segmentada de routing minimiza la capacidad de interconexión. Para la familia XC4000 el modelo "wire-load" considera las líneas entre CLBs y IOBs (input/output blocks), este modelo permite estimar los retardos lógicos con una alta precisión.

Para el dispositivo XC4010XLPC84-3, se tiene las siguientes especificaciones:
LCs= 950
CLBs= 400 (20x20)
Puertas= 10,000
RAM= 12,800 bits
Vcc= 3.3 vol.
Input threshold= 1.6 vol (compatible con CMOS y TTL)
Tecnología SRAM= 0.35 micras
Velocidad= 80Mhz (para el máximo speed grade -1)
Frecuencia Máxima I/O = 53 Mhz
Temperatura: 0 - 85 ºC
Retardos combinacionales dentro de un CBL (en nanosegundos):
Tilo= 1.6 (F/G inputs to X/Y outputs)
Tiho= 2.7 (F/G inputs via H' to X/Y outputs)
Tito= 2.9 (F/G inputs via transparent latch to Q outputs)
Topcy= 2.7 (Operand inputs (F1, F2, G1, G4) to C OUT)
Tascy= 3.3 (Add/Subtract input (F3) to C OUT)
Tincy= 2.0 (Initialization inputs (F1, F3) to C OUT)

Como resumen de parámetros temporales principales se tiene:
t_{CO}= 1.5ns - 2.1ns

t_D = 0.1ns (en el CLB), 20ns (para varios "switching points")
t_{LC}= 1.6ns - 3.3ns, (dependiendo que entradas/salidas se usa)
t_{SU}= 0.6ns - 1.1ns, para todos los casos se tiene hold time = 0ns
T(ur)= 3.2ns (Valor práctico estimado de una unidad de retardo)

En el cálculo de la potencia de circuitos síncronos se observa que la potencia estática es despreciable, pocos miliwatts. La potencia dinámica es directamente proporcional a la frecuencia de operación y depende del diseño, dicha potencia dinámica se puede estimar mediante:

P= VxKxNxF

Donde:
V= voltaje de fuente.
K= factor del dispositivo (K= 28)
N= número de los registros activos
F= frecuencia en Mhz

El factor K predice la corriente del dispositivo para diseños típicos, es calculado cuando se implementa contadores de 16 bits en el todo el FPGA y se hace la medición de corriente a una frecuencia de 1Mhz.

Es importante observar que el consumo de potencia depende directamente de los registros activos y la frecuencia, lo cual significa que en el caso de un diseño asíncrono se consigue un ahorro de dicho consumo.

VIRTEX

Xilinx presenta como productos recientes las plataformas de FPGAs Virtex, por ejemplo Virtex-II en la tecnología de 0.15 micras y 8 capas de metal, transistores de alta velocidad de 0.12 micras, voltaje de núcleo de 1.5V y fuente de 3.3V. Un aspecto a considerar es que la familia Virtex no soporta logiBLOX y se debe utilizar la herramienta Core Generator, la cual está más integrada a las herramientas de Xilinx y mantiene la compatibilidad del diseño mediante las descripciones VHDL al nivel de comportamiento.

A fin de comparar esta nueva familia y específicamente para el dispositivo XC2V40-6, de máximo "speed grade", se tiene los siguientes datos técnicos:

El mayor retardo de un LUT, que depende las entradas y salidas utilizadas, es:
Tif5x= 0.76ns

El retardo máximo de buffer de salida hacia la matriz de interconexión es:
Tio = 0.45 ns

Luego el valor teórico de la unidad de retardo: T(ur) = 0.76ns + 0.45ns = 1.21ns

Mediante simulación, en el entorno ISE 5.2i, el valor práctico de una unidad de retardo es T(ur)= 1.13ns y es medido considerando el retraso de los bloques de procesamiento.

La implementación en esta nueva familia de FPGAs de Xilinx queda con una alternativa a considerar en el futuro. Aunque estos valores se deben considerar provisionales, se observa que el valor de T(ur) es mayor que el de Stratix de Altera, una razón es el retardo que introducen los transistores de paso de los "switching points" dentro de los PSM.

ALTERA

La figura B3 muestra el diagrama lógico de un LE (Logic Element) y en la figura B4 se tiene la configuración interna del FPGA FLEX 10K.

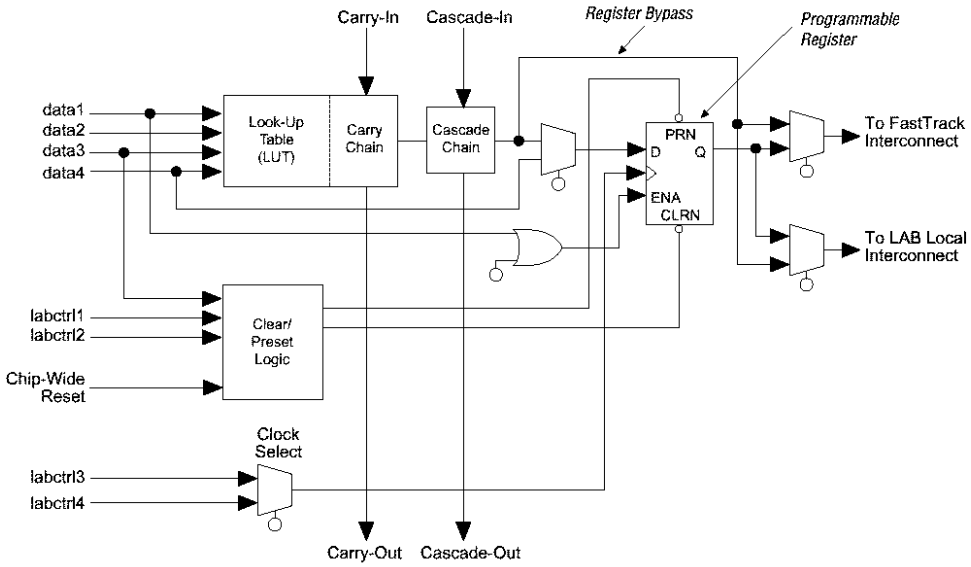

Fig. B3: Diagrama lógico de un LE FLEX10K

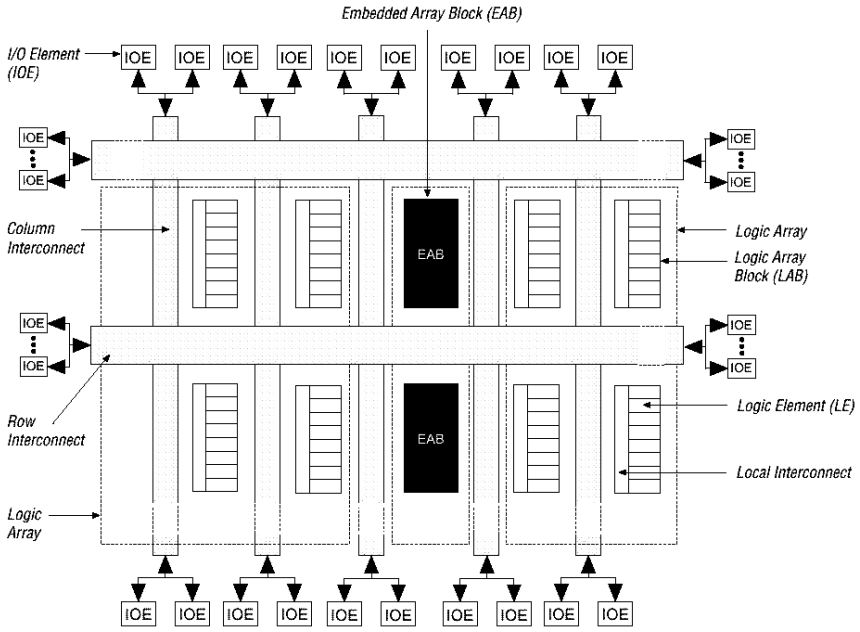

Fig. B4: Configuración interna del FPGA FLEX 10K

La arquitectura básica se compone de LEs agrupados en LABs (Logic Array Block), 8 LEs conforma un LAB. Dentro de cada LAB la salida de cada LE puede ser la entrada de otro LE mediante un esquema de conexión local (Local Fast Track). Esta configuración permite una arquitectura de grano grueso y alta velocidad, al mismo tiempo una alta utilización por su arquitectura de grano fino.

Los LABs se comunican con otros LABs mediante filas y columnas, se conectan mediante líneas de metal denominados FastTracks que se distribuyen en toda el área y transportan las señales entre los IOEs (input/output elements). Por lo cual eliminan los retardos acumulativos que presentan las conexiones mediante segmentos.

Un LE es unidad lógica más pequeña, contiene una LUT (lookup table) de 4 entradas, un flip flop programable, una cadena en cascada para funciones de más entradas y una cadena de transporte (carry chain) que proporciona una función de transporte rápida entre LEs adyacentes para circuitos aritméticos.

Los IOE son los pines de entrada y salida de propósito general y pines de dedicados para señales de control rápidas y globales. Cada IOE tiene un registro que puede ser usado como entrada o salida. Tiene opción de inversión de señales, el buffer de salida puede ajustar el slew rate para bajo ruido o alta velocidad.

Para el dispositivo FLEX EPF10K20RC240-4, se tiene las siguientes especificaciones principales:
LEs= 1,152 (144x8)
EAB= 6 (cada EAB= 2,048 bits de memoria)
LABs= 144
Puertas= 20,000
Slew rate max= 2.9ns
Vcc= 5 vol.
Tecnología CMOS= 0.5 micras
Temperatura ambiente = 0 hasta 70 ºC (valor típico = 25ºC)
t_{CO}= 1.5ns
t_D = 0.6ns (dentro del LAB)
$t_D = t_{SAMEROW}$ = 3.9ns (LAB en la misma fila)
$t_D = t_{DIFFROW}$ = 5.5ns (LAB en diferentes filas)
$t_D = t_{SAMECOLUMN}$ = 1.6ns (LE to IOE en la misma columna)
$t_{LE} = t_{LUT} + t_{COMB}$ = 1.7 + 0.6 = 2.3ns.
t_{SU}= 2.5ns
t_{INSU}= 6.0 ns (setup time del reloj global en un registro IOE)
t_{INH}= 0.0ns (hold time del reloj global en un registro IOE)
t_{OUTCO}= 2.0 - 8.4ns (retardo clock-to-output del reloj global en un registro IOE)
I_{CC0} = 0.5(typ) – 10mA (max)
T(ur)= 2.9ns (Valor práctico estimado de una unidad de retardo)

Para el cálculo de la potencia de circuitos síncronos y para el dispositivo FLEX 10K, la potencia puede ser estimada mediante:

$$P = P_{INT} + P_{IO} = (I_{CCSTANDBY} + I_{CCACTIVE}) \times V_{CC} + P_{IO}$$

La corriente V_{CC} de reposo es el parámetro $I_{CC0} = I_{CCSTANDBY}$

La corriente $I_{CCACTIVE}$ depende de la frecuencia de conmutación y de la aplicación de la lógica, es calculado en base de la corriente típica que cada LE consume. Puede ser calculada como:

$I_{CCACTIVE} = K \times f_{MAX} \times N \times tog_{LE}$

Donde:

f_{MAX} = La máxima frecuencia de operación en Mhz.

N = Es el número total de LEs usados en el dispositivo.

tog_{LE} = Es el promedio en porcentaje de los LEs que conmutan en cada ciclo de reloj, típicamente es el 12.5%.

K = Constante = 82 - 95 (EPF10K20 K= 89)

La potencia P_{IO} depende de la característica carga de la salida del dispositivo y de la frecuencia de conmutación, luego puede ser calculada según los parámetros del dispositivo.

De lo anterior, un aspecto a tomar en cuenta es que el consumo de potencia tiene una dependencia directa con la frecuencia de operación y el porcentaje de LEs que conmutan en cada periodo. Por cual un diseño asíncrono permite un ahorro de potencia aunque la manera exacta de cálculo de la potencia queda como un estudio a realizar.

En la figura B5 se tiene el diagrama lógico de un LE APEX 20K y la figura B6 muestra la estructura interna de dicho FPGA. Como se observa son similares al FPGA FLEX 10K, aunque en este caso un LAB contiene 10 LEs.

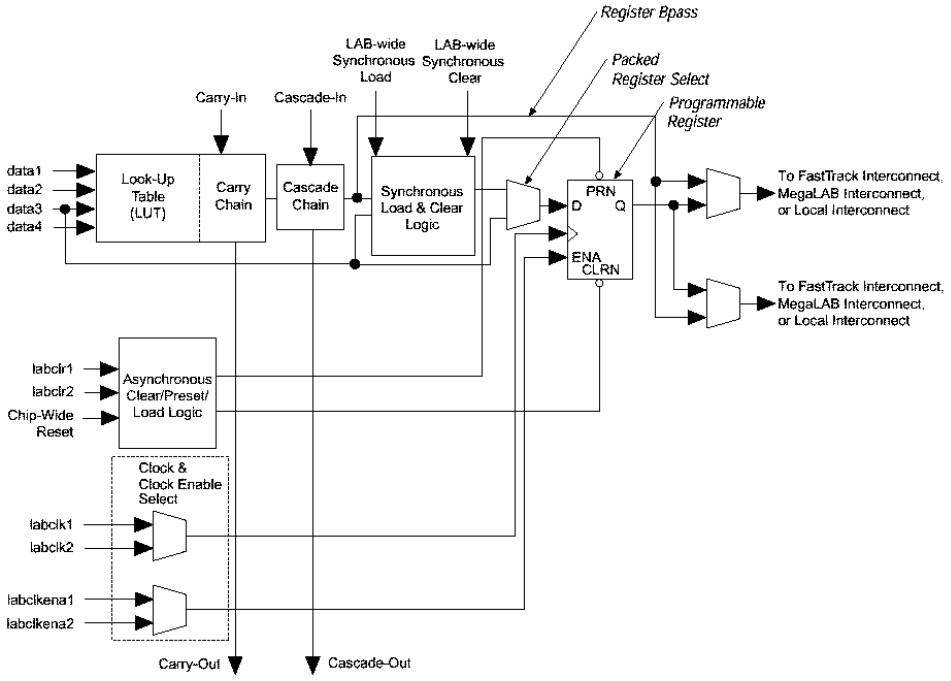

Fig. B5: Diagrama lógico de un LE APEX 20K

El compilador QUARTUS II, cuando se emplean LPMs, automáticamente habilita el modo apropiado para funciones comunes como contadores, sumadores y otras funciones aritméticas.

En el presente trabajo se ha utilizado QUARTUS II versión 1.1 ya que permite las mismas facilidades empleadas en el compilador MAX+PLUS II, versión 9.6. Versiones posteriores de QUARTUS II presentan algunas características distintas pero permiten actualizar los resultados en las simulaciones.

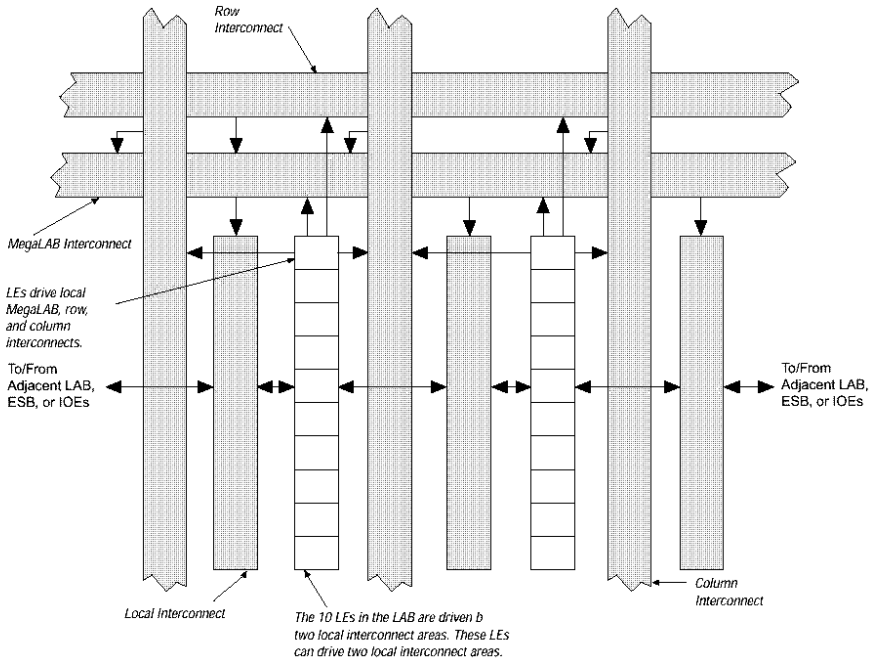

Fig. B6: Configuración interna del FPGA APEX 20K

Para el dispositivo APEX EP20K100E–2, se tiene las principales especificaciones:
LEs= 4160 (416x10)
LABs= 416
ESBs= 26 (embedded system blocks)
MEGALABs= 16 LABs + 1 ESB
RAM = 53,248 bits
Vcc= 1.8 vol. (interfaces para 1.8v, 2.5v, 3.3v)
Tecnología CMOS= 0.15 micras
t_{CO}= 0.28ns (retardo clock-to-output en el registro LE)
t_{LE}= t_{LUT} = 0.95ns (retardo data-in to data-out en el LUT)
t_{SU}= 0.25ns (setup time en el registro LE)
t_{H}= 0.25ns (hold time en el register LE)
t_{INSU}= 2.32ns (setup time del reloj global en el registro IOE)
t_{INH}= 0.00ns (hold time del reloj global en el registro IOE)
t_{OUTCO}= 2.00-5.35ns (retardo clock-to-output del reloj global en el registro IOE)
t_{F1-4} = 0.27ns (retardo de fan-out de una interconexión local)
t_{F5-20} = 1.26ns (retardo de fan-out de una interconexión megalab)
t_{F20+} = 1.36ns (retardo de fan-out de una interconexión fast-track)
T(ur)= 1.12ns (Valor práctico estimado de una unidad de retardo)

Para el dispositivo APEX EP20K100E–1, se tiene las especificaciones:
t_{CO}= 0.28ns
t_{LE}= t_{LUT} = 0.80ns
t_{SU}= 0.25ns

t_H= 0.25ns
t_{INSU}= 2.23 ns
t_{INH}= 0.00ns
t_{OUTCO}= 2.00-4.86 ns
t_{F1-4} = 0.24ns
t_{F5-20} = 1.04ns
t_{F20+} = 1.12ns
T(ur)= 0.96ns (Valor práctico estimado de una unidad de retardo)

STRATIX y CYCLONE

Como productos recientes de ALTERA se tiene Cyclone que es un FPGA de 1.5V tecnología SRAM de 0.13 micras y todas las capas con líneas de cobre. Stratix es un PLD en la misma tecnología que el anterior. Como parámetros relevantes, aunque los valores son provisionales, para ambos se tiene:

Por ejemplo para el dispositivo EP1S10F780C5 de Stratix:

T lut= 417ps (retardo combinacional data-in to data-out de un LUT)
T local= 313ps (retardo de interconexión local de un LAB)

Lo cual permite estimar el parámetro T(ur) teórico = 417ps + 313ps = 730ps.

Mediante simulación en QUARTUS II (versión 2.1) se tiene el valor T(ur)= 710ps, que es valor práctico de una unidad de retardo estimado mediante el retraso de los bloques de procesamiento.

Aunque estos valores son provisionales, este valor significaría una mejora con respecto a la serie APEX (-1) de alrededor de un 25% en el throughput de la Unidad de Clasificación cuyo diseño es la aplicación del presente trabajo. La implementación en dichos FPGAs recientes queda como un trabajo futuro.

REFERENCIAS

[1] O.A. Petlin, C. Farnsworth, S.B. Furber. “Design for Testability of an Asynchronous Adder”. htpp://www.cs.man.ac.uk/amulet, AMULET Group, The University of Manchester, UK.

[2] L.A Hollaar. “Direct Implementation of Asynchronous Control Units”. IEEE Transactions on Computers, Vol. C-31, No 12, pp. 1133-1141, Diciembre 1982.

[3] S.M. Nowick, D.L. Dill. “Automatic Synthesis of Locally-Clocked Asynchronous States Machines”. Proceedings of ICCAD, pp. 318-321, 1991.

[4] A.J. Martin. “The limitations to Delay-Insensitivity in Asynchronous Circuits”. Proceedings of the 1990 MIT Conference on Advanced Research in VLSI, pp. 263-278, 1990.

[5] C.E. Molnar, T.P. Fang, F.U. Rosenberger. “Synthesis of Delay-Insensitive Modules”. Proceedings of the 1985 Chapel Hill Conference on Advanced Research in VLSI, pp. 67-86, 1985.

[6] I.E. Sutherland. "Micropipelines". Communications of the ACM, Vol. 32, No 6, pp. 720-738, Junio 1989.

[7] A. Khoche, E. Brunvand. “Testing Micropipelines”. Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems, Async94, pp. 239-246, IEEE Computer Society Press, Noviembre 1994.

[8] M. Roncken. “Partial Scan Test for Asynchronous Circuits illustrated on a DCC Error Corrector”. International Symposium on Advanced Research in Asynchronous Circuits and Systems, Async94, pp. 247-256, IEEE Computer Society Press, Noviembre 1994.

[9] O.A. Petlin, S.B. Furber. “Scan Testing of Asynchronous Sequential Circuits”. htpp://www.cs.man.ac.uk/amulet, AMULET Group, The University of Manchester, UK.

[10] T. Murata “Petri Nets: Properties, Analysis and Applications”. Proceedings of the IEEE, Vol. 77, No. 4, pp. 541-580, 1989.

[11] P.A. Beerel, T.H.Y. Meng. “Automatic Gate-Level Synthesis of Speed-Independent Circuits”. Proceedings of ICCAD, pp. 581-586, 1992.

[12] L. Lavagno, K. Keutzer, A. Sangiovanni-Vincentelli. “Algorithms for Synthesis of Hazard-free Asynchronous Circuits”. Proceedings of DAC, pp. 302-308, 1991.

[13] G. Birtwistle, Y. Liu. “Specification of Manchester Amulet 1: Top Level Specification”. C.S. Department, University of Calgary, Diciembre 1994.

[14] S.M. Nowick, D.L Dill. “Exact two-level minimization of hazard-free logic with multiple-input changes”.IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 14(8):986-997, Agosto 1995.

[15] D.L Dill, S.M. Nowick, R.F. Sproull. “Specification and Automatic Verification of Self-timed Queues”. Formal Methods in System Design”, 1(1):29-60, Julio 1992.

[16] S. Chakraborty, D.L. Dill, K.Y. Chang, K.Y. Yun. “Timing Analysis for extended burst-mode circuits”. International Symposium on Advanced Research in Asynchronous Circuits and Systems, Async97, IEEE Computer Society Press, Abril 1997.

[17] “Design, Automation and Test for Asynchronous Circuits and Systems”. Information Society Technologies Programme, IST-1999-29119, Enero 2002.

[18] S. Rotem, K. Stevens, R. Ginosar, P. Beerel, C. Myers, K. Yun, R. Kol, C. Dike, M. Roncken, B. Agapiev. “RAPPID: An Asynchronous Industrial Length Decoder”. International Symposium on Advanced Research in Asynchronous Circuits and Systems, Async99, IEEE Computer Society Press, pp. 60-70, Abril 1999.

[19] J. Riera, L. Sintes, J. Escudero, J. Carrabina. “An Efficient Asynchronous Digital Filter using our Cell Library DCVSL_LIB”. European Microelectronics Application Conference, pp 10-13, EMAC’97.

[20] A. Kondratyev, K. Lwin. “Design of Asynchronous Circuits Using CAD Tools”. IEEE Design & Test of Computers, pp. 107-117, Julio-Agosto 2002.

[21] S.B. Furber, A. Efthymiou, J.D. Garside, D.W. Lloyd, M.J.G. Lewis, S. Temple. “Power Management in the Amulet Microprocessors”. IEEE Design & Test of Computers, pp. 42-51, Marzo-Abril 2001.

[22] http://www. fulcrummicro.com, FULCRUM MICROSYSTEMS INC.

[23] S.B. Furber, D.A. Edwards, J.D. Garside. “AMULET3: a 100 MIPS Asynchronous Embedded Processor”. htpp://www.cs.man.ac.uk/amulet, AMULET Group, The University of Manchester, UK.

[24] “GEM 301 GSM Baseband Processor”. Preliminary Information, Mitel Semiconductors, 1997.

[25] M. Lewis, L. Brackenbury. “CADRE: A Low-Power, Low-EMI DSP Architecture for Digital Mobile Phones”. htpp://www.cs.man.ac.uk/amulet, AMULET Group, University of Manchester, UK.

[26] K.Y. Yun, K.W. James, R.H. Fairlie-Cuninghame, S. Chakraborty, R.L. Cruz. “A Self-Timed Real-Time Sorting Network”. IEEE Transactions on VLSI Systems, Marzo 2000.

[27] http://www.inrialpes.fr, INRIA Rhone-Alpes.

[28] K.van Berkel, R. Burgess, J. Kessels, Ad Peeters, M. Roncken, F. Schalij, R.van de Wiel. “A Single-rail re-implementation of DCC Error Corrector using a Generic Standard-cell library”. Asynchronous Design Methodologies, IEEE Computer Society Press, pp. 72-79, Mayo 1995.

[29] http://www.theseus.com, THESEUS LOGIC.

[30] http://www.atmforum.com, ATM-FORUM.

[31] A. Nikologiannis, M. Katevenis. “Efficient Per-Flow Queuing in DRAM at OC-192 Line Rate using Out-of-Order Execution Techniques”. Institute of Computer Science, http://www.ics.forth.gr, Grecia.

[32] M. Katevenis, E. Markatos, I. Mavroidis. “Weighted Round-Robin Scheduler using Per-Class Urgency Counters”. Institute of Computer Science, http://www.ics.forth.gr, Grecia.

[33] R. H. Hofmann, R. Muller. “A Multifunctional High-Speed Switch Element for ATM Applications”. IEEE Journal of Solid-State Circuits, Vol. 27, No 7, pp. 1036-1040, Julio 1992.

[34] W.O. Budde, H. Keller, H. Reumerman, P. van de Wiel. Philips Research Labs. “An Asynchronous High-Speed Packet Switching Component”. IEEE Design & Test of Computers, pp. 33-42, Summer 1994.

[35] G. Chen, I. Stavrakakis. “ATM Traffic Management with Diversified Loss and Delay Requirements”. Proceedings of IEEE INFOCOM, pp. 1037-1044, 1996.

[36] R.L. Cruz. “Quality of Service Guarantees in Virtual Circuit Switched Networks”. IEEE Journal on Selected Areas in Communications, Vol. 13, No 6, pp. 1048-1056, Agosto 1995.

[37] A.G. Wassal, M.A. Hasan. ”Low-Power System-Level Design of VLSI Packet Switching Fabrics”. IEEE Transactions on computer-aided design of integrated circuits and systems, Vol. 20, No 6, pp. 723-738, Junio 2001.

[38] K. Maheswaran. “Implementing Self-Timed Circuits in Field Programmable Gate Arrays”. Computer Engineering Research Laboratory, University of California Davis, USA.

[39] http://www.altera.com, Reportes y Manuales de ALTERA CORPORATION.

[40] http://www.xilinx.com, Reportes y Manuales de XILINX CORPORATION.

[41] Steve Furber. “Computing without Clocks: Micropipelining the ARM Processor”. htpp://www.cs.man.ac.uk/amulet, AMULET Group, The University of Manchester, UK.

[42] Sutherland, Sproull, Harris. “Logical Effort: Designing Fast CMOS Circuits”. Morgan Kaufmann Publishers, 1999.

[43] S. Sawitzki, St. Köhler, R.G. Spallek, J. Scheneider, St. Rülke. “Quantitative Comparison of Different Design Approaches for FPGA-based Design Flows”. International Workshop on IP Based Synthesis and System Design, pp. 241-246, Grenoble, Francia 1999.

[44] http://www.edif.org, ELECTRONIC DESIGN INTERFACE FORMAT.

[45] K.E. Batcher. “Sorting Networks and their Applications”. AFIPS Spring Joint Computer Conference, Vol. 32, pp. 307-314, New Jersey, USA, 1968.

[46] Y.C. Lin. “On Balancing Sorting on a Linear Array”. IEEE Transactions on Parallel and Distributed Systems, 4(5):566-571, Mayo 1993.

[47] G.C. Heinze, R. Palmer, I. Dresser, N. Leister. “A Three chip set for ATM Switching”. IEEE Custom Integrated Circuits Conference, pp. 14.3.1-14.3.4, Mayo 1992.

[48] D. Picker, R.D. Fellman. “A VLSI Priority Queue with Inheritance and Overwriting”. IEEE Transactions on VLSI Systems, 3(2):245-253, Junio 1995.

[49] B. Ahn, J.M. Murray. “A Pipelined, Expandable VLSI Sorting Engine implemented in CMOS technology”. IEEE International Conference on Circuits and Systems, 3:134-137, 1989.

[50] D.T. Lee, HSU Chang, C.K. Wong. “An on-chip Compare/steer Bubble Sorter”. IEEE Transactions on Computers, C-30(6):396-404, Junio 1981.

[51] R. Lin, S. Olariu. “Reconfigurable Buses with Shift Switching: Concepts and Applications”. IEEE Transactions on Parallel and Distributed Systems, Vol. 6, No 1, pp. 93-102, Enero 1995.

[52] M. Down, Y. Perl, M. Saks, L. Rudolph. “The Periodic Balanced Sorting Network”. pp. 738-757, Journal of the ACM 36.

[53] J.D. Lee, K.E. Batcher. “Minimizing Communication in the Bitonic Sort”. IEEE Transactions on parallel and distributed systems, Vol. 11, No 5, pp. 459-473, Mayo 2000.

[54] L.E. Winslow, Y.C. Chow. “The Analysis and Design of some new Sorting Machines”. IEEE Transactions on Computers, Vol. C-32, pp. 677-683, Julio 1983.

[55] K. van Berkel. “Handshake circuits. An Asynchronous Architecture for VLSI Programming”. Cambridge University Press, 1993.

[56] M.R. Samatham, D.K. Pradhan. “The De Bruijin Multiprocessor Network: A versatile Parallel Processing and Sorting network for VLSI”. IEEE Transactions on Computers, Vol. 38, pp. 567-581, Abril 1989.

[57] K. Kantawala, D.L. Tao. “Design, Analysis, and Evaluation of Concurrent Checking Sorting Networks”, IEEE transactions on very large scale integration (VLSI) systems”, Vol. 5, No 3, pp. 338-473, Septiembre 1997.

[58] Y.H. Choi, M. Malek. “A Fault Tolerant Systolic Sorter”. Transactions on Computers, Vol. 37, pp. 617-624, Mayo 1988.

[59] O.A. Petlin, C. Farnsworth, S.B. Furber. “Built-In Self Test Design of an Asynchronous Block Sorter”. htpp://www.cs.man.ac.uk/amulet, AMULET Group, The University of Manchester, UK.

[60] J.N. Seizovic. “Pipeline Synchronization”. International Symposium on Advanced Research in Asynchronous Circuits and Systems, Async94, IEEE Computer Society Press, pp. 87-96, Noviembre 1999.

[61] K.Y. Yun, A.E. Dooply. “Plausible Clocking Based Heterogeneous Systems”. IEEE Transactions on VLSI Systems, Vol. 7, No. 4, pp. 482-487, Diciembre 1999.

[62] M.B Josephs, J.T. Yantchev. “CMOS Design of the tree arbiter element”. IEEE Transactions on VLSI Systems, Vol. 4, No 4, pp. 472-476, Diciembre 1996.

[63] http://www.lsilogic.com, LSI LOGIC CORPORATION.

[64] M. Katevenis, D. Serpanos, E. Markatos. “Multi-Queue Management and Scheduling for Improved QoS in Communication Networks””. http://www.ics.forth.gr, Institute of Computer Science, Grecia.

[65] G. Kornaros, C. Kozyrakis, P. Vatsolaki, M. Katevenis. “Pipelined Multi-Queue Management in a VLSI ATM Switch Chip with Credit-Based Flow-Control”. http://www.ics.forth.gr, Institute of Computer Science, Grecia.

[66] Xilinx Alliance Series 2.1i, Quick Start Guide, Chapter 2: Implementation Tools Tutorial.

[67] S. Temple, S.B. Furber. "On-chip timing reference for self-timed microprocessor". Electronic Letters, Vol. 36, No 11, pp. 942-943, Mayo 2000.

[68] D.A. Edwards, W.B. Toms. “The Status of Asynchronous Design in Industry”. Information Society Technologies Programme, IST-1999-29119, Enero 2003.

[69] http://www.rambus.com

[70] B. Monder, G. Pacific, C. Zulowski. “The Cylindric Switch: an Architecture for a Manageable VLSI Giga-cell Switch”, Proc. IEEE ICC90, Atlanta, GA, pp. 567-571, Abril 1990.

[71] J.M. Pitts, J.A. Schormans. "Introduction to ATM Design and Performance: with Applications Analysis Software". John Willey & Sons Ltd. 1996.

[72] A.G. Wassal, M.A. Hasan. "Prioritised ATM Switches on Board Satellites: Architectural Analysis and Design". IEE Proc.-Commun. Vol. 147, No 5, pp. 277-284, Octubre 2000.

[73] http://www.maxim-ic.com, MAX3680 Reportes Técnicos, Maxim Semiconductor.

[74] R.Alarcón Matutti. "Synthesis of Asynchronous Digital Circuits applied to Communications" IV Forum on Asynchronous Circuits. London-UK, Julio 1998.

[75] R. Alarcón, J.Carrabina. "FPGA Implementation of a Sorter Circuit using Asynchronous Micropipeline". International Workshop on IP Based Synthesis and System Design.Grenoble-Francia, Diciembre 1999.

[76] R.Alarcón, J.Carrabina. "A Comparative Evaluation of an Asynchronous Implementation on FPGAs". XVI Conference on Design of Circuits and Integrated Systems, DCIS. Porto-Portugal, Noviembre 2001.

[77] R.Alarcón, J. Carrabina. "Methodology of Asynchronous design: Micropipeline Implementation for QoS-ATM Management". Working Group on Asynchronous Circuit Design, ACiD-WG Workshop. Creta-Grecia, Enero 2003.

[78] R.Alarcón, J. Carrabina. "Asynchronous Circuits Applied to Communications: A Modular Micropipeline Architecture for QoS-ATM Management". International Conference on Computer, Communication and Control Technologies, CCCT. Florida-USA, Julio 2003.

[26] K.Y. Yun, K.W. James, R.H. Fairlie-Cuninghame, S. Chakraborty, R.L. Cruz. “A Self-Timed Real-Time Sorting Network”. IEEE Transactions on VLSI Systems, Marzo 2000.

[27] http://www.inrialpes.fr, INRIA Rhone-Alpes.

[28] K.van Berkel, R. Burgess, J. Kessels, Ad Peeters, M. Roncken, F. Schalij, R.van de Wiel. “A Single-rail re-implementation of DCC Error Corrector using a Generic Standard-cell library”. Asynchronous Design Methodologies, IEEE Computer Society Press, pp. 72-79, Mayo 1995.

[29] http://www.theseus.com, THESEUS LOGIC.

[30] http://www.atmforum.com, ATM-FORUM.

[31] A. Nikologiannis, M. Katevenis. “Efficient Per-Flow Queuing in DRAM at OC-192 Line Rate using Out-of-Order Execution Techniques”. Institute of Computer Science, http://www.ics.forth.gr, Grecia.

[32] M. Katevenis, E. Markatos, I. Mavroidis. “Weighted Round-Robin Scheduler using Per-Class Urgency Counters”. Institute of Computer Science, http://www.ics.forth.gr, Grecia.

[33] R. H. Hofmann, R. Muller. “A Multifunctional High-Speed Switch Element for ATM Applications”. IEEE Journal of Solid-State Circuits, Vol. 27, No 7, pp. 1036-1040, Julio 1992.

[34] W.O. Budde, H. Keller, H. Reumerman, P. van de Wiel. Philips Research Labs. “An Asynchronous High-Speed Packet Switching Component”. IEEE Design & Test of Computers, pp. 33-42, Summer 1994.

[35] G. Chen, I. Stavrakakis. “ATM Traffic Management with Diversified Loss and Delay Requirements”. Proceedings of IEEE INFOCOM, pp. 1037-1044, 1996.

[36] R.L. Cruz. “Quality of Service Guarantees in Virtual Circuit Switched Networks”. IEEE Journal on Selected Areas in Communications, Vol. 13, No 6, pp. 1048-1056, Agosto 1995.

[37] A.G. Wassal, M.A. Hasan. ”Low-Power System-Level Design of VLSI Packet Switching Fabrics”. IEEE Transactions on computer-aided design of integrated circuits and systems, Vol. 20, No 6, pp. 723-738, Junio 2001.

[38] K. Maheswaran. “Implementing Self-Timed Circuits in Field Programmable Gate Arrays”. Computer Engineering Research Laboratory, University of California Davis, USA.

[39] http://www.altera.com, Reportes y Manuales de ALTERA CORPORATION.

[40] http://www.xilinx.com, Reportes y Manuales de XILINX CORPORATION.

[41] Steve Furber. “Computing without Clocks: Micropipelining the ARM Processor”. htpp://www.cs.man.ac.uk/amulet, AMULET Group, The University of Manchester, UK.

[42] Sutherland, Sproull, Harris. “Logical Effort: Designing Fast CMOS Circuits”. Morgan Kaufmann Publishers, 1999.

[43] S. Sawitzki, St. Köhler, R.G. Spallek, J. Scheneider, St. Rülke. “Quantitative Comparison of Different Design Approaches for FPGA-based Design Flows”. International Workshop on IP Based Synthesis and System Design, pp. 241-246, Grenoble, Francia 1999.

[44] http://www.edif.org, ELECTRONIC DESIGN INTERFACE FORMAT.

[45] K.E. Batcher. “Sorting Networks and their Applications”. AFIPS Spring Joint Computer Conference, Vol. 32, pp. 307-314, New Jersey, USA, 1968.

[46] Y.C. Lin. “On Balancing Sorting on a Linear Array”. IEEE Transactions on Parallel and Distributed Systems, 4(5):566-571, Mayo 1993.

[47] G.C. Heinze, R. Palmer, I. Dresser, N. Leister. “A Three chip set for ATM Switching”. IEEE Custom Integrated Circuits Conference, pp. 14.3.1-14.3.4, Mayo 1992.

[48] D. Picker, R.D. Fellman. “A VLSI Priority Queue with Inheritance and Overwriting”. IEEE Transactions on VLSI Systems, 3(2):245-253, Junio 1995.

[49] B. Ahn, J.M. Murray. “A Pipelined, Expandable VLSI Sorting Engine implemented in CMOS technology”. IEEE International Conference on Circuits and Systems, 3:134-137, 1989.

[50] D.T. Lee, HSU Chang, C.K. Wong. “An on-chip Compare/steer Bubble Sorter”. IEEE Transactions on Computers, C-30(6):396-404, Junio 1981.

[51] R. Lin, S. Olariu. “Reconfigurable Buses with Shift Switching: Concepts and Applications”. IEEE Transactions on Parallel and Distributed Systems, Vol. 6, No 1, pp. 93-102, Enero 1995.

[52] M. Down, Y. Perl, M. Saks, L. Rudolph. “The Periodic Balanced Sorting Network”. pp. 738-757, Journal of the ACM 36.

[53] J.D. Lee, K.E. Batcher. “Minimizing Communication in the Bitonic Sort”. IEEE Transactions on parallel and distributed systems, Vol. 11, No 5, pp. 459-473, Mayo 2000.

[54] L.E. Winslow, Y.C. Chow. “The Analysis and Design of some new Sorting Machines”. IEEE Transactions on Computers, Vol. C-32, pp. 677-683, Julio 1983.

[55] K. van Berkel. “Handshake circuits. An Asynchronous Architecture for VLSI Programming”. Cambridge University Press, 1993.

[56] M.R. Samatham, D.K. Pradhan. “The De Bruijin Multiprocessor Network: A versatile Parallel Processing and Sorting network for VLSI”. IEEE Transactions on Computers, Vol. 38, pp. 567-581, Abril 1989.

[57] K. Kantawala, D.L. Tao. “Design, Analysis, and Evaluation of Concurrent Checking Sorting Networks”, IEEE transactions on very large scale integration (VLSI) systems”, Vol. 5, No 3, pp. 338-473, Septiembre 1997.

[58] Y.H. Choi, M. Malek. “A Fault Tolerant Systolic Sorter”. Transactions on Computers, Vol. 37, pp. 617-624, Mayo 1988.

[59] O.A. Petlin, C. Farnsworth, S.B. Furber. “Built-In Self Test Design of an Asynchronous Block Sorter”. htpp://www.cs.man.ac.uk/amulet, AMULET Group, The University of Manchester, UK.

[60] J.N. Seizovic. “Pipeline Synchronization”. International Symposium on Advanced Research in Asynchronous Circuits and Systems, Async94, IEEE Computer Society Press, pp. 87-96, Noviembre 1999.

[61] K.Y. Yun, A.E. Dooply. “Plausible Clocking Based Heterogeneous Systems”. IEEE Transactions on VLSI Systems, Vol. 7, No. 4, pp. 482-487, Diciembre 1999.

[62] M.B Josephs, J.T. Yantchev. “CMOS Design of the tree arbiter element”. IEEE Transactions on VLSI Systems, Vol. 4, No 4, pp. 472-476, Diciembre 1996.

[63] http://www.lsilogic.com, LSI LOGIC CORPORATION.

[64] M. Katevenis, D. Serpanos, E. Markatos. “Multi-Queue Management and Scheduling for Improved QoS in Communication Networks””. http://www.ics.forth.gr, Institute of Computer Science, Grecia.

[65] G. Kornaros, C. Kozyrakis, P. Vatsolaki, M. Katevenis. “Pipelined Multi-Queue Management in a VLSI ATM Switch Chip with Credit-Based Flow-Control”. http://www.ics.forth.gr, Institute of Computer Science, Grecia.

[66] Xilinx Alliance Series 2.1i, Quick Start Guide, Chapter 2: Implementation Tools Tutorial.

[67] S. Temple, S.B. Furber. "On-chip timing reference for self-timed microprocessor". Electronic Letters, Vol. 36, No 11, pp. 942-943, Mayo 2000.

[68] D.A. Edwards, W.B. Toms. “The Status of Asynchronous Design in Industry”. Information Society Technologies Programme, IST-1999-29119, Enero 2003.

[69] http://www.rambus.com

[70] B. Monder, G. Pacific, C. Zulowski. “The Cylindric Switch: an Architecture for a Manageable VLSI Giga-cell Switch”, Proc. IEEE ICC90, Atlanta, GA, pp. 567-571, Abril 1990.

[71] J.M. Pitts, J.A. Schormans. "Introduction to ATM Design and Performance: with Applications Analysis Software". John Willey & Sons Ltd. 1996.

[72] A.G. Wassal, M.A. Hasan. "Prioritised ATM Switches on Board Satellites: Architectural Analysis and Design". IEE Proc.-Commun. Vol. 147, No 5, pp. 277-284, Octubre 2000.

[73] http://www.maxim-ic.com, MAX3680 Reportes Técnicos, Maxim Semiconductor.

[74] R.Alarcón Matutti. "Synthesis of Asynchronous Digital Circuits applied to Communications" IV Forum on Asynchronous Circuits. London-UK, Julio 1998.

[75] R. Alarcón, J.Carrabina. "FPGA Implementation of a Sorter Circuit using Asynchronous Micropipeline". International Workshop on IP Based Synthesis and System Design.Grenoble-Francia, Diciembre 1999.

[76] R.Alarcón, J.Carrabina. "A Comparative Evaluation of an Asynchronous Implementation on FPGAs". XVI Conference on Design of Circuits and Integrated Systems, DCIS. Porto-Portugal, Noviembre 2001.

[77] R.Alarcón, J. Carrabina. "Methodology of Asynchronous design: Micropipeline Implementation for QoS-ATM Management". Working Group on Asynchronous Circuit Design, ACiD-WG Workshop. Creta-Grecia, Enero 2003.

[78] R.Alarcón, J. Carrabina. "Asynchronous Circuits Applied to Communications: A Modular Micropipeline Architecture for QoS-ATM Management". International Conference on Computer, Communication and Control Technologies, CCCT. Florida-USA, Julio 2003.

Universitat Autònoma de Barcelona
Escola Tècnica Superior d'Enginyeria

"SÍNTESIS DE CIRCUITOS DIGITALES ASÍNCRONOS APLICADOS A COMUNICACIONES: UNA APROXIMACIÓN MICROPIPELINE PARA QoS-ATM"

APÉNDICE: "BIBLIOTECA DE LOS MÓDULOS DE CONTROL Y PROCESAMIENTO"

Apéndice de la tesis
presentada por
Rubén V. Alarcón Matutti
para optar al grado de
Doctor en Ing. Electrónica

Bellaterra, julio de 2003

INDICE

“BIBLIOTECA DE LOS MÓDULOS DE CONTROL Y PROCESAMIENTO”

En la presente biblioteca y con referencia a la tesis desarrollada; se tienen las descripciones VHDL de los módulos constitutivos, los programas se han diseñado utilizado simples LPMs de la biblioteca de ALTERA. Se muestran todos los módulos utilizados en ambas arquitecturas paralela y serie, se incluyen los módulos de procesamiento, control y latches del micropipeline. Para la plataforma de XILINX los equivalentes son los logiBLOX y actualmente el Core Generator Tool, se puede obtener la descripción VHDL de comportamiento directamente desde la compilación y desarrollar los módulos equivalentes a los anteriores, por lo cual solo se muestran los listados de algunos módulos básicos.

DESCRIPCIÓN DE LOS MÓDULOS VHDL-LPMs

ELEMENTO-C

```
--- Componente  CNOTM:
--- Puerta elemento-C, input b negado
--- Basado en un LPM_ADD_SUB
--- Señal “aclr” de reset

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY cnotm IS

PORT (
a                       : IN  STD_LOGIC;
b                       : IN  STD_LOGIC;
aclr                    : IN  STD_LOGIC;
c                       : OUT STD_LOGIC
        );

END cnotm;

ARCHITECTURE xx OF cnotm IS

---Definición de variables auxiliares:
signal OUT1 : STD_LOGIC;
signal OUT2,OUT3 : STD_LOGIC_VECTOR(0 DOWNTO 0);

BEGIN

--Defición de componente:
prueba1   : lpm_add_sub

GENERIC MAP (LPM_WIDTH => 1, LPM_REPRESENTATION => "UNSIGNED")
PORT MAP (dataa => OUT3,
datab => OUT2,
cout => OUT1, cin => OUT1 );

c    <= OUT1;
OUT3(0) <= (NOT aclr) AND a;
OUT2(0) <= (NOT aclr) AND (NOT b);

END xx;
```

MÓDULO RETARDO DE REFERENCIA

```
--- Componente DELAY1:
--- Unidad de referencia de retardo
--- Basado en una puerta elemento-C
--- En cadena forma los “matched delay”

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY delay1 IS

PORT   (
ain                  : IN  STD_LOGIC;
ady                  : OUT STD_LOGIC );

END delay1;

ARCHITECTURE xx OF delay1 IS

---Definición de variables auxiliares:
signal OUT1 : STD_LOGIC;
signal OUT2 : STD_LOGIC_VECTOR(0 DOWNTO 0);

BEGIN

--Definición de componente:
prueba1   : lpm_add_sub

GENERIC MAP (LPM_WIDTH => 1, LPM_REPRESENTATION => "UNSIGNED")
PORT MAP (dataa => OUT2,
datab => OUT2,
cout => OUT1, cin => OUT1 );

ady  <= OUT1;
OUT2(0) <= ain;

END xx;
```

MÓDULO ALATCH (ARQUITECTURA PARALELA)

```
--- Componente ALATCH:
--- Las  entradas son 2 vectores de N bits cada uno: dx[N],  dy[N]
--- Captura ambos vectores con la activación del control "c"
--- Las salidas son   qx[N],   qy[N]
--- Señal "a" clear

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY alatch IS

GENERIC  (
LPM_WIDTH : integer :=16
              );

PORT (
dx                    : IN   STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
dy                    : IN   STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
c                     : IN   STD_LOGIC;
a                     : IN   STD_LOGIC;
qx                    : OUT  STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
qy                    : OUT  STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0)
      );

END alatch;

ARCHITECTURE xx OF alatch IS

---Definición de variables auxiliares:
signal OUT1, OUT2, OUTA, OUTB : STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
signal OUT3, OUT4, OUTC, OUTD : STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
signal AA,BB                  : STD_LOGIC;

BEGIN

---Definición de componentes:
prueba1   : lpm_latch
GENERIC MAP (LPM_WIDTH => LPM_WIDTH)
PORT MAP (data => dx,
q => OUTA,
gate => AA, aclr => a
              );

prueba2   : lpm_latch
GENERIC MAP (LPM_WIDTH => LPM_WIDTH)
PORT MAP (data => dx,
q => OUTB,
gate => BB, aclr => a
              );

prueba3   : lpm_latch
GENERIC MAP (LPM_WIDTH => LPM_WIDTH)
PORT MAP (data => dy,
q => OUTC,
gate => AA, aclr => a
               );
```

```
prueba4   : lpm_latch
GENERIC MAP (LPM_WIDTH => LPM_WIDTH)
PORT MAP (data => dy,
q => OUTD,
gate => BB, aclr => a
                );

---lazo de asignación:
L1: for i in 0 to LPM_WIDTH-1 generate

OUT1(i) <= OUTA(i) AND c;
OUT2(i) <= OUTB(i) AND (NOT c);
OUT3(i) <= OUTC(i) AND c;
OUT4(i) <= OUTD(i) AND (NOT c);

end generate;

AA <= NOT c;
BB <= c;
qx <= OUT1 OR OUT2;
qy <= OUT3 OR OUT4;

END xx;

--- Definición  de ALATCH_PACKAGE:

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

PACKAGE alatch_package IS

COMPONENT alatch
GENERIC  (
LPM_WIDTH : integer :=16
            );

PORT   (
dx                  : IN   STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
dy                  : IN   STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
c                   : IN   STD_LOGIC;
a                   : IN   STD_LOGIC;
qx                  : OUT  STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
qy                  : OUT  STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0)
        );

END COMPONENT;

END alatch_package;
```

MÓDULO BLOUNO (ARQUITECTURA PARALELA)

```
--- Componente BLOUNO:
--- Bloque básico para el comparador bitónico: 2 MUX y 1 COMPARADOR.
--- Entradas: xin , yin de N bits
--- Salidas: xoutmax y youtmin de N bits

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY blouno IS

GENERIC  (
LPM_WIDTH : integer :=16
                );

PORT  (
xin                     : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
yin                     : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
xoutmax                 : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
youtmin                 : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
        );

END blouno;

ARCHITECTURE xx OF blouno IS

---Definición de variables auxiliares:
signal OUT1             : STD_LOGIC;
signal OUTA, OUTB : STD_LOGIC_VECTOR(0 DOWNTO 0);
signal zin                  :  STD_LOGIC_2D(1 DOWNTO 0 , LPM_WIDTH-1 DOWNTO 0);

BEGIN

---Definición de componentes:
prueba1   : lpm_mux
GENERIC MAP (LPM_WIDTH => LPM_WIDTH, LPM_SIZE => 2, LPM_WIDTHS => 1)
PORT MAP (data => zin,
result => xoutmax,
sel => OUTA
                );

prueba2   : lpm_compare
GENERIC MAP (LPM_WIDTH => LPM_WIDTH, LPM_REPRESENTATION => "UNSIGNED")
PORT MAP (dataa => xin, datab => yin,
ageb => OUT1
                );

prueba3   : lpm_mux
GENERIC MAP (LPM_WIDTH => LPM_WIDTH, LPM_SIZE => 2, LPM_WIDTHS => 1)
PORT MAP (data => zin,
result => youtmin,
sel => OUTB
                 );

---lazo de asignación:
L1: for i in 0 to LPM_WIDTH-1 generate
zin(1,i) <= xin(i);
zin(0,i) <= yin(i);

end generate;
```

```
OUTA(0) <= OUT1;
OUTB(0) <= NOT OUT1;

END xx;


--- Definición de BLOUNO_PACKAGE:

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

PACKAGE blouno_package IS

COMPONENT blouno

GENERIC  (
LPM_WIDTH : integer :=16
                );

PORT   (
xin                     : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
yin                     : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
xoutmax                 : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
youtmin                 : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
          );

END COMPONENT;

END blouno_package;
```

MÓDULO BLODOS (ARQUITECTURA PARALELA)

```
--- Componente BLODOS:
--- Bloque básico para el comparador bitónico: 2 MUX y 1 COMPARADOR.
--- Entradas: xin , yin de N bits
--- Salidas: youtmax ,  xoutmin de N bits

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY blodos IS
GENERIC  (
LPM_WIDTH : integer :=16
              );

PORT (
xin                  : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
yin                  : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
youtmax              : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
xoutmin              : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
        );


END blodos;

ARCHITECTURE xx OF blodos IS

---Definición de variables auxiliares:
signal OUT1            : STD_LOGIC;
signal OUTA, OUTB : STD_LOGIC_VECTOR(0 DOWNTO 0);
signal zin             : STD_LOGIC_2D(1 DOWNTO 0 , LPM_WIDTH-1 DOWNTO 0);

BEGIN

---Definición de componentes:
prueba1   : lpm_mux
GENERIC MAP (LPM_WIDTH => LPM_WIDTH, LPM_SIZE => 2, LPM_WIDTHS => 1)
PORT MAP (data => zin,
result => youtmax,
sel => OUTA
                );

prueba2   : lpm_compare
GENERIC MAP (LPM_WIDTH => LPM_WIDTH, LPM_REPRESENTATION => "UNSIGNED")
PORT MAP (dataa => xin, datab => yin,
ageb => OUT1
                 );

prueba3   : lpm_mux
GENERIC MAP (LPM_WIDTH => LPM_WIDTH, LPM_SIZE => 2, LPM_WIDTHS => 1)
PORT MAP (data => zin,
result => xoutmin,
sel => OUTB
                );

---lazo de asignación:
L1: for i in 0 to LPM_WIDTH-1 generate
zin(1,i) <= xin(i);
zin(0,i) <= yin(i);

end generate;
```

```
OUTA(0) <= OUT1;
OUTB(0) <= NOT OUT1;

END xx;


--- Definición de BLODOS_PACKAGE


LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

PACKAGE blodos_package IS

COMPONENT blodos

GENERIC  (
LPM_WIDTH : integer :=16
              );

PORT   (
xin                  : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
yin                  : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
youtmax              : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
xoutmin              : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
          );

END COMPONENT;

END blodos_package;
```

MÓDULO BLOUNOAMP (ARQUITECTURA PARALELA)

```
--- Componente BLOUNOAMP:
--- En base a  BLOUNO y ALATCH
--- Conforma las etapas del micropipeline

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

PACKAGE blouno_package IS

COMPONENT blouno

GENERIC  (
LPM_WIDTH : integer :=16
                );

PORT   (
xin                     : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
yin                     : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
xoutmax                 : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
youtmin                 : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
          );

END COMPONENT;

END blouno_package;


PACKAGE alatch_package IS

COMPONENT alatch
GENERIC  (
LPM_WIDTH : integer :=16
                );

PORT   (
dx                      : IN   STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
dy                      : IN   STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
c                       : IN   STD_LOGIC;
a                       : IN   STD_LOGIC;
qx                      : OUT  STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
qy                      : OUT  STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0)
         );

END COMPONENT;

END alatch_package;

LIBRARY work;
USE work.blouno_package. ALL;
USE work.alatch_package.ALL;

ENTITY blounoamp IS

GENERIC  (
LPM_WIDTH : integer :=16
                );
```

```
PORT  (
xinput, yinput  : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrl, aclr          : IN   STD_LOGIC;
xmax, ymin      : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
          );

END blounoamp;


ARCHITECTURE xx OF blounoamp IS

---Definición de variables auxiliares:
signal OUTA, OUTB : STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);

BEGIN

comp1 : blouno PORT MAP (xin => xinput, yin => yinput,
xoutmax => OUTA, youtmin => OUTB);

comp2 : alatch PORT MAP (dx => OUTA, dy => OUTB, c => ctrl, a => aclr,
qx => xmax, qy => ymin);

END xx;


---- Definición de BLOUNOAMP_PACKAGE

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

PACKAGE blounoamp_package IS

COMPONENT blounoamp

GENERIC  (
LPM_WIDTH : integer :=16
                );

PORT   (
xinput, yinput  : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrl, aclr          : IN   STD_LOGIC;
xmax, ymin      : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
          );

END COMPONENT;

END blounoamp_package;
```

MÓDULO BLODOSAMP (ARQUITECTURA PARALELA)

```
--- Componente BLODOSAMP:
--- En base a  BLODOS y ALATCH
--- Conforma las etapas del micropipeline

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

PACKAGE blodos_package IS

COMPONENT blodos

GENERIC  (
LPM_WIDTH : integer :=16
                );

PORT   (
xin                  : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
yin                  : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
youtmax              : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
xoutmin              : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
          );

END COMPONENT;

END blodos_package;


PACKAGE alatch_package IS

COMPONENT alatch

GENERIC  (
LPM_WIDTH : integer :=16
               );

PORT   (
dx                   : IN   STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
dy                   : IN   STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
c                    : IN   STD_LOGIC;
a                    : IN   STD_LOGIC;
qx                   : OUT  STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
qy                   : OUT  STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0)
         );

END COMPONENT;

END alatch_package;

LIBRARY work;
USE work.blodos_package. ALL;
USE work.alatch_package.ALL;

ENTITY blodosamp IS

GENERIC  (
LPM_WIDTH : integer :=16
               );
```

```
PORT  (
xinput, yinput  : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrl, aclr          : IN   STD_LOGIC;
ymax, xmin      : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
          );

END blodosamp;

ARCHITECTURE xx OF blodosamp IS

---Definición de variables auxiliares:
signal OUTA, OUTB : STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);


BEGIN

comp1 : blodos PORT MAP (xin => xinput, yin => yinput,
youtmax => OUTA, xoutmin => OUTB);

comp2 : alatch PORT MAP (dx => OUTB, dy => OUTA, c => ctrl, a => aclr,

qx => xmin, qy => ymax);

END xx;


--- Definición de BLODOSAMP_PACKAGE


LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

PACKAGE blodosamp_package IS

COMPONENT blodosamp

GENERIC  (
LPM_WIDTH : integer :=16
                );

PORT   (
xinput, yinput  : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrl, aclr          : IN   STD_LOGIC;
ymax, xmin      : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
            );

END COMPONENT;

END blodosamp_package;
```

MÓDULO BLOQUE1 (ARQUITECTURA PARALELA)

```
--- Componente  BLOQUE1:
--- En base a BLOUNOAMP y BLODOSAMP
--- Conforma el nivel 1 del comparador bitónico


LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

PACKAGE blounoamp_package IS

COMPONENT blounoamp

GENERIC  (
LPM_WIDTH : integer :=16
                );

PORT   (
xinput, yinput  : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrl, aclr          : IN   STD_LOGIC;
xmax, ymin      : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
            );

END COMPONENT;

END blounoamp_package;


PACKAGE blodosamp_package IS

COMPONENT blodosamp

GENERIC  (
LPM_WIDTH : integer :=16
                );

PORT   (
xinput, yinput  : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrl, aclr          : IN   STD_LOGIC;
ymax, xmin      : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
            );

END COMPONENT;

END blodosamp_package;

LIBRARY work;
USE work.blounoamp_package. ALL;
USE work.blodosamp_package. ALL;


ENTITY bloque1 IS

GENERIC  (
LPM_WIDTH : integer :=16
                );
```

```
PORT  (
xb11,xb12,xb13,xb14,xb15,xb16,xb17,xb18  : IN STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrlb1, aclr                                : IN   STD_LOGIC;
yb11,yb12,yb13,yb14,yb15,yb16,yb17,yb18  : OUT STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
        );

END bloque1;

ARCHITECTURE xx OF bloque1 IS

BEGIN

---interconexión de componentes:
comp1 : blounoamp PORT MAP (xinput => xb11, yinput => xb15,
aclr => aclr, ctrl => ctrlb1,
xmax => yb11, ymin => yb12);

comp2 : blodosamp PORT MAP (xinput => xb12, yinput => xb16,
aclr => aclr, ctrl => ctrlb1,
ymax => yb14, xmin => yb13);

comp3 : blodosamp PORT MAP (xinput => xb13, yinput => xb17,
aclr => aclr, ctrl => ctrlb1,
ymax => yb16, xmin => yb15);

comp4 : blounoamp PORT MAP (xinput => xb14, yinput => xb18,
aclr => aclr, ctrl => ctrlb1,
xmax => yb17, ymin => yb18);

END xx;
```

MÓDULO BLOQUE2 (ARQUITECTURA PARALELA)

```
--- Componente  BLOQUE2:
--- En base a BLOUNOAMP y BLODOSAMP
--- Conforma el nivel 2 del comparador bitónico

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

PACKAGE blounoamp_package IS

COMPONENT blounoamp

GENERIC  (
LPM_WIDTH : integer :=16
                );

PORT   (
xinput, yinput  : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrl, aclr         : IN   STD_LOGIC;
xmax, ymin     : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
            );

END COMPONENT;

END blounoamp_package;


PACKAGE blodosamp_package IS

COMPONENT blodosamp

GENERIC  (
LPM_WIDTH : integer :=16
                );

PORT   (
xinput, yinput  : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrl, aclr         : IN   STD_LOGIC;
ymax, xmin     : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
            );

END COMPONENT;

END blodosamp_package;

LIBRARY work;
USE work.blounoamp_package. ALL;
USE work.blodosamp_package. ALL;

ENTITY bloque2 IS

GENERIC  (
LPM_WIDTH : integer :=16
                 );

PORT  (
xb21,xb22,xb23,xb24,xb25,xb26,xb27,xb28  : IN STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrlb2, aclr                                          : IN   STD_LOGIC;
yb21,yb22,yb23,yb24,yb25,yb26,yb27,yb28  : OUT STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
            );
END bloque2;
```

```
ARCHITECTURE xx OF bloque2 IS

BEGIN

---interconexión de componentes:
comp1 : blounoamp PORT MAP (xinput => xb21, yinput => xb23,
aclr => aclr, ctrl => ctrlb2,
xmax => yb21, ymin => yb22);

comp2 : blounoamp PORT MAP (xinput => xb22, yinput => xb24,
aclr => aclr, ctrl => ctrlb2,
xmax => yb23, ymin => yb24);

comp3 : blodosamp PORT MAP (xinput => xb25, yinput => xb27,
aclr => aclr, ctrl => ctrlb2,
ymax => yb26, xmin => yb25);

comp4 : blodosamp PORT MAP (xinput => xb26, yinput => xb28,
aclr => aclr, ctrl => ctrlb2,
ymax => yb28, xmin => yb27);

END xx;
```

MÓDULO BLOQUE3 (ARQUITECTURA PARALELA)

```
--- Componente  BLOQUE3:
--- En base a BLOUNOAMP y BLODOSAMP
--- Conforma el nivel 3 del comparador bitónico

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

PACKAGE blounoamp_package IS

COMPONENT blounoamp

GENERIC  (
LPM_WIDTH : integer :=16
              );

PORT   (
xinput, yinput          : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrl, aclr              : IN    STD_LOGIC;
xmax, ymin              : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
         );

END COMPONENT;

END blounoamp_package;

PACKAGE blodosamp_package IS

COMPONENT blodosamp

GENERIC  (
LPM_WIDTH : integer :=16
              );

PORT   (
xinput, yinput  : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrl, aclr          : IN   STD_LOGIC;
ymax, xmin     : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
          );

END COMPONENT;

END blodosamp_package;

LIBRARY work;
USE work.blounoamp_package. ALL;
USE work.blodosamp_package. ALL;

ENTITY bloque3 IS

GENERIC  (
LPM_WIDTH : integer :=16
              );

PORT  (
xb31,xb32,xb33,xb34,xb35,xb36,xb37,xb38  : IN STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrlb3, aclr                                        : IN   STD_LOGIC;
yb31,yb32,yb33,yb34,yb35,yb36,yb37,yb38  : OUT STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
         );

END bloque3;
```

```
ARCHITECTURE xx OF bloque3 IS

BEGIN

---interconexión de componentes:
comp1 : blounoamp PORT MAP (xinput => xb31, yinput => xb33,
aclr => aclr, ctrl => ctrlb3,
xmax => yb31, ymin => yb32);

comp2 : blounoamp PORT MAP (xinput => xb32, yinput => xb34,
aclr => aclr, ctrl => ctrlb3,
xmax => yb33, ymin => yb34);

comp3 : blodosamp PORT MAP (xinput => xb35, yinput => xb37,
aclr => aclr, ctrl => ctrlb3,
ymax => yb36, xmin => yb35);

comp4 : blodosamp PORT MAP (xinput => xb36, yinput => xb38,
aclr => aclr, ctrl => ctrlb3,
ymax => yb38, xmin => yb37);

END xx;
```

MÓDULO BLOQUE4 (ARQUITECTURA PARALELA)

```
--- Componente  BLOQUE4:
--- En base a BLOUNOAMP y BLODOSAMP
--- Conforma el nivel 4 del comparador bitónico

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

PACKAGE blounoamp_package IS

COMPONENT blounoamp
GENERIC  (
LPM_WIDTH : integer :=16
              );

PORT   (
xinput, yinput  : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrl, aclr          : IN    STD_LOGIC;
xmax, ymin     : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
         );

END COMPONENT;

END blounoamp_package;

PACKAGE blodosamp_package IS

COMPONENT blodosamp
GENERIC  (
LPM_WIDTH : integer :=16

             );

PORT   (
xinput, yinput  : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrl, aclr          : IN   STD_LOGIC;
ymax, xmin     : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
         );

END COMPONENT;

END blodosamp_package;

LIBRARY work;
USE work.blounoamp_package. ALL;
USE work.blodosamp_package. ALL;

ENTITY bloque4 IS

GENERIC  (
LPM_WIDTH : integer :=16
              );

PORT  (
xb41,xb42,xb43,xb44,xb45,xb46,xb47,xb48  : IN STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrlb4, aclr                                          : IN   STD_LOGIC;
yb41,yb42,yb43,yb44,yb45,yb46,yb47,yb48  : OUT STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
         );
END bloque4;
```

```
ARCHITECTURE xx OF bloque4 IS

BEGIN

---interconexión de componentes:
comp1 : blounoamp PORT MAP (xinput => xb41, yinput => xb45,
aclr => aclr, ctrl => ctrlb4,
xmax => yb41, ymin => yb42);

comp2 : blounoamp PORT MAP (xinput => xb42, yinput => xb46,
aclr => aclr, ctrl => ctrlb4,
xmax => yb43, ymin => yb44);

comp3 : blounoamp PORT MAP (xinput => xb43, yinput => xb47,
aclr => aclr, ctrl => ctrlb4,
xmax => yb45, ymin => yb46);

comp4 : blounoamp PORT MAP (xinput => xb44, yinput => xb48,
aclr => aclr, ctrl => ctrlb4,
xmax => yb47, ymin => yb48);

END xx;
```

MÓDULO BLOQUE5 (ARQUITECTURA PARALELA)

```
--- Componente  BLOQUE5:
--- En base a BLOUNOAMP y BLODOSAMP
--- Conforma el nivel 5 del comparador bitónico

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

PACKAGE blounoamp_package IS

COMPONENT blounoamp
GENERIC  (
LPM_WIDTH : integer :=16
              );

PORT   (
xinput, yinput             : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrl, aclr                 : IN    STD_LOGIC;
xmax, ymin                 : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
          );

END COMPONENT;

END blounoamp_package;


PACKAGE blodosamp_package IS
COMPONENT blodosamp

GENERIC  (
LPM_WIDTH : integer :=16
              );

PORT   (
xinput, yinput  : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrl, aclr          : IN   STD_LOGIC;
ymax, xmin     : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
           );

END COMPONENT;

END blodosamp_package;

LIBRARY work;
USE work.blounoamp_package. ALL;
USE work.blodosamp_package. ALL;

ENTITY bloque5 IS

GENERIC  (
LPM_WIDTH : integer :=16
               );

PORT  (
xb51,xb52,xb53,xb54,xb55,xb56,xb57,xb58  : IN STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrlb5, aclr                                          : IN   STD_LOGIC;
yb51,yb52,yb53,yb54,yb55,yb56,yb57,yb58  : OUT STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
          );
END bloque5;
```

```
ARCHITECTURE xx OF bloque5 IS

BEGIN

---interconexión de componentes:
comp1 : blounoamp PORT MAP (xinput => xb51, yinput => xb55,
aclr => aclr, ctrl => ctrlb5,
xmax => yb51, ymin => yb52);

comp2 : blounoamp PORT MAP (xinput => xb53, yinput => xb57,
aclr => aclr, ctrl => ctrlb5,
xmax => yb53, ymin => yb54);

comp3 : blounoamp PORT MAP (xinput => xb52, yinput => xb56,
aclr => aclr, ctrl => ctrlb5,
xmax => yb55, ymin => yb56);

comp4 : blounoamp PORT MAP (xinput => xb54, yinput => xb58,
aclr => aclr, ctrl => ctrlb5,
xmax => yb57, ymin => yb58);

END xx;
```

MÓDULO BLOQUE6 (ARQUITECTURA PARALELA)

```
--- Componente  BLOQUE6:
--- En base a BLOUNOAMP y BLODOSAMP
--- Conforma el nivel 6 del comparador bitónico


LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

PACKAGE blounoamp_package IS

COMPONENT blounoamp
GENERIC  (
LPM_WIDTH : integer :=16
            );

PORT   (
xinput, yinput            : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrl, aclr                : IN    STD_LOGIC;
xmax, ymin                : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
         );

END COMPONENT;

END blounoamp_package;

PACKAGE blodosamp_package IS
COMPONENT blodosamp

GENERIC  (
LPM_WIDTH : integer :=16
            );

PORT   (
xinput, yinput  : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrl, aclr          : IN   STD_LOGIC;
ymax, xmin     : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
         );

END COMPONENT;

END blodosamp_package;

LIBRARY work;
USE work.blounoamp_package. ALL;
USE work.blodosamp_package. ALL;

ENTITY bloque6 IS

GENERIC  (
LPM_WIDTH : integer :=16
            );

PORT  (
xb61,xb62,xb63,xb64,xb65,xb66,xb67,xb68  : IN STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrlb6, aclr                             : IN   STD_LOGIC;
yb61,yb62,yb63,yb64,yb65,yb66,yb67,yb68  : OUT STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
         );
END bloque6;
```

```
ARCHITECTURE xx OF bloque6 IS

BEGIN

---interconexión de componentes:
comp1 : blounoamp PORT MAP (xinput => xb61, yinput => xb63,
aclr => aclr, ctrl => ctrlb6,
xmax => yb61, ymin => yb62);

comp2 : blounoamp PORT MAP (xinput => xb62, yinput => xb64,
aclr => aclr, ctrl => ctrlb6,
xmax => yb63, ymin => yb64);

comp3 : blounoamp PORT MAP (xinput => xb65, yinput => xb67,
aclr => aclr, ctrl => ctrlb6,
xmax => yb65, ymin => yb66);

comp4 : blounoamp PORT MAP (xinput => xb66, yinput => xb68,
aclr => aclr, ctrl => ctrlb6,
xmax => yb67, ymin => yb68);

END xx;
```

MÓDULO BITOLAT (ARQUITECTURA PARALELA)

```
--- Componente  BITOLAT:
--- Basándose en todos los módulos BLOQUE1 hasta BLOQUE6
--- Conforma el comparador bitónico micropipeline (módulo de más alta jerarquia)

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

PACKAGE bloque1_package IS

COMPONENT bloque1

GENERIC  (
LPM_WIDTH : integer :=16
              );

PORT   (
xb11, xb12, xb13, xb14, xb15, xb16, xb17, xb18:IN STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrlb1, aclr                                   : IN    STD_LOGIC;
yb11, yb12, yb13, yb14, yb15, yb16, yb17, yb18:OUT STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
           );

END COMPONENT;
END bloque1_package;

PACKAGE bloque2_package IS

COMPONENT bloque2

GENERIC  (
LPM_WIDTH : integer :=16
              );

PORT   (
xb21, xb22, xb23, xb24, xb25, xb26, xb27, xb28:IN STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrlb2, aclr                                    : IN    STD_LOGIC;
yb21, yb22, yb23, yb24, yb25, yb26, yb27, yb28:OUT STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
          );

END COMPONENT;
END bloque2_package;

PACKAGE bloque3_package IS

COMPONENT bloque3

GENERIC  (
LPM_WIDTH : integer :=16
              );

PORT   (
xb31, xb32, xb33, xb34, xb35, xb36, xb37, xb38:IN STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrlb3, aclr                                   : IN    STD_LOGIC;
yb31, yb32, yb33, yb34, yb35, yb36, yb37, yb38:OUT STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
          );

END COMPONENT;
END bloque3_package;

PACKAGE bloque4_package IS
```

```
COMPONENT bloque4
GENERIC  (
LPM_WIDTH : integer :=16
              );

PORT   (
xb41, xb42, xb43, xb44, xb45, xb46, xb47, xb48:IN STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrlb4, aclr                                    : IN    STD_LOGIC;
yb41, yb42, yb43, yb44, yb45, yb46, yb47, yb48:OUT STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
          );

END COMPONENT;
END bloque4_package;

PACKAGE bloque5_package IS

COMPONENT bloque5
GENERIC  (
LPM_WIDTH : integer :=16
              );

PORT   (
xb51, xb52, xb53, xb54, xb55, xb56, xb57, xb58:IN STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrlb5, aclr                                    : IN    STD_LOGIC;
yb51, yb52, yb53, yb54, yb55, yb56, yb57, yb58:OUT STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
          );

END COMPONENT;
END bloque5_package;


PACKAGE bloque6_package IS

COMPONENT bloque6
GENERIC  (
LPM_WIDTH : integer :=16
               );

PORT   (
xb61, xb62, xb63, xb64, xb65, xb66, xb67, xb68:IN STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrlb6, aclr                                    : IN    STD_LOGIC;
yb61, yb62, yb63, yb64, yb65, yb66, yb67, yb68:OUT STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
          );

END COMPONENT;
END bloque6_package;

LIBRARY work;
USE work.bloque1_package. ALL;
USE work.bloque2_package. ALL;
USE work.bloque3_package. ALL;
USE work.bloque4_package. ALL;
USE work.bloque5_package. ALL;
USE work.bloque6_package. ALL;

ENTITY bitolat IS
GENERIC  (
LPM_WIDTH : integer :=16
              );
PORT  (
x1, x2, x3, x4, x5, x6, x7, x8  : IN STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrl1, ctrl2,ctrl3, ctrl4,ctrl5, ctrl6, aclr      : IN   STD_LOGIC;
y1, y2, y3, y4, y5, y6, y7, y8  : OUT STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
         );
END bitolat;
```

```
ARCHITECTURE xx OF bitolat IS

--Definición de variables auxiliares:
signal OUb11,OUb12,OUb13,OUb14,OUb15,OUb16,OUb17,OUb18 : STD_LOGIC_VECTOR(LPM_WIDTH-1
DOWNTO 0);
signal OUb21,OUb22,OUb23,OUb24,OUb25,OUb26,OUb27,OUb28 : STD_LOGIC_VECTOR(LPM_WIDTH-1
DOWNTO 0);
signal OUb31,OUb32,OUb33,OUb34,OUb35,OUb36,OUb37,OUb38 : STD_LOGIC_VECTOR(LPM_WIDTH-1
DOWNTO 0);
signal OUb41,OUb42,OUb43,OUb44,OUb45,OUb46,OUb47,OUb48 : STD_LOGIC_VECTOR(LPM_WIDTH-1
DOWNTO 0);
signal OUb51,OUb52,OUb53,OUb54,OUb55,OUb56,OUb57,OUb58 : STD_LOGIC_VECTOR(LPM_WIDTH-1
DOWNTO 0);

BEGIN

---interconexión de componentes:
comp1 : bloque1 PORT MAP (
xb11=>x1, xb12=>x2, xb13=>x3, xb14=>x4, xb15=>x5, xb16=>x6, xb17=> x7, xb18 => x8,
aclr => aclr, ctrlb1 => ctrl1,
yb11=>OUb11,yb12=>OUb12,yb13=>OUb13,yb14=>OUb14,yb15=>OUb15,yb16=>OUb16,yb17=>OUb17,yb18=
>OUb18
                                    );
comp2 : bloque2 PORT MAP (

xb21=>OUb11,xb22=>OUb12,xb23=>OUb13,xb24=>OUb14,xb25=>OUb15,xb26=>OUb16,xb27=>OUb17,xb28=
>OUb18,
aclr => aclr, ctrlb2 => ctrl2,
yb21=>OUb21,yb22=>OUb22,yb23=>OUb23,yb24=>OUb24,yb25=>OUb25,yb26=>OUb26,yb27=>OUb27,yb28=
>OUb28
                                     );
comp3 : bloque3 PORT MAP (

xb31=>OUb21,xb32=>OUb22,xb33=>OUb23,xb34=>OUb24,xb35=>OUb25,xb36=>OUb26,xb37=>OUb27,xb38=
>OUb28,
aclr => aclr, ctrlb3 => ctrl3,
yb31=>OUb31,yb32=>OUb32,yb33=>OUb33,yb34=>OUb34,yb35=>OUb35,yb36=>OUb36,yb37=>OUb37,yb38=
>OUb38
                                    );
comp4 : bloque4 PORT MAP (

xb41=>OUb31,xb42=>OUb32,xb43=>OUb33,xb44=>OUb34,xb45=>OUb35,xb46=>OUb36,xb47=>OUb37,xb48=
>OUb38,
aclr => aclr, ctrlb4 => ctrl4,
yb41=>OUb41,yb42=>OUb42,yb43=>OUb43,yb44=>OUb44,yb45=>OUb45,yb46=>OUb46,yb47=>OUb47,yb48=
>OUb48
                                    );
comp5 : bloque5 PORT MAP (
xb51=>OUb41,xb52=>OUb42,xb53=>OUb43,xb54=>OUb44,xb55=>OUb45,xb56=>OUb46,xb57=>OUb47,xb58=
>OUb48,
aclr => aclr, ctrlb5 => ctrl5,
yb51=>OUb51,yb52=>OUb52,yb53=>OUb53,yb54=>OUb54,yb55=>OUb55,yb56=>OUb56,yb57=>OUb57,yb58=
>OUb58
                                    );
comp6 : bloque6 PORT MAP (

xb61=>OUb51,xb62=>OUb52,xb63=>OUb53,xb64=>OUb54,xb65=>OUb55,xb66=>OUb56,xb67=>OUb57,xb68=
>OUb58,
aclr => aclr, ctrlb6 => ctrl6,
yb61=>y1,yb62=>y2,yb63=>y3,yb64=>y4,yb65=>y5,yb66=>y6,yb67=>y7,yb68=>y8
                                    );
END xx;
```

MÓDULO BLO8LAT (ARQUITECTURA PARALELA)

```
--- Componente BLO8LAT:
--- Transporta los vectores de direcciones de memoria
--- Captura 8 vectores cuando se activa “ctrl”
--- La entrada son 8 vectores de N bits cada uno, Xinp1...Xinp2
--- La salida son  X1..X8 de N bits cada uno.
--- Señal “aclr”  de reset

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
LIBRARY lpm;
USE lpm.lpm_components.ALL;

PACKAGE alatch_package IS
COMPONENT alatch

GENERIC  (
LPM_WIDTH : integer :=16
            );
PORT   (
dx                  : IN   STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
dy                  : IN   STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
c                   : IN   STD_LOGIC;
a                   : IN   STD_LOGIC;
qx                  : OUT  STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
qy                  : OUT  STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0)
         );

END COMPONENT;
END alatch_package;

LIBRARY work;
USE work.alatch_package.ALL;

ENTITY blo8lat IS
GENERIC  (
LPM_WIDTH : integer :=16
            );

PORT  (
xinp1,xinp2,xinp3,xinp4,xinp5,xinp6,xinp7,xinp8:IN STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
ctrl, aclr                                      : IN   STD_LOGIC;
x1,x2,x3,x4,x5,x6,x7,x8                         : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
         );

END blo8lat;

ARCHITECTURE xx OF blo8lat IS
BEGIN

---interconexión de componentes:
comp1 : alatch PORT MAP (dx => xinp1, dy => xinp2, c => ctrl, a => aclr,
qx => x1, qy => x2);

comp2 : alatch PORT MAP (dx => xinp3, dy => xinp4, c => ctrl, a => aclr,
qx => x3, qy => x4);

comp3 : alatch PORT MAP (dx => xinp5, dy => xinp6, c => ctrl, a => aclr,
qx => x5, qy => x6);

comp4 : alatch PORT MAP (dx => xinp7, dy => xinp8, c => ctrl, a => aclr,
qx => x7, qy => x8);

END xx;
```

MÓDULO SLATCH (ARQUITECTURA SERIE)

```
--- Componente SLATCH:
--- Captura la entrada cuando se activa "c"
--- La entrada es el vector dx[N] de N bits
--- La salida es el vector qx[N] de N bits
--- La señal "a" de preset


LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY slatch IS

GENERIC  (
LPM_WIDTH : integer :=16
              );

PORT    (

dx                    : IN   STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
c                     : IN   STD_LOGIC;
a                     : IN   STD_LOGIC;
qx                    : OUT  STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0)
          );

END slatch;

ARCHITECTURE xx OF slatch IS

---Definición de variables auxiliares:
signal OUT1, OUT2, OUTA, OUTB : STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
signal AA,BB                  : STD_LOGIC;

BEGIN

---Definición de componentes:
prueba1   : lpm_latch

GENERIC MAP (LPM_WIDTH => LPM_WIDTH)
PORT MAP (data => dx,
q => OUTA,
gate => AA, aset => a
               );

prueba2   : lpm_latch

GENERIC MAP (LPM_WIDTH => LPM_WIDTH)
PORT MAP (data => dx,
q => OUTB,
gate => BB, aset => a
               );

---lazo de asignación:
L1: for i in 0 to LPM_WIDTH-1 generate

OUT1(i) <= OUTA(i) AND c;
OUT2(i) <= OUTB(i) AND (NOT c);

end generate;
```

```
AA <= NOT c;
BB <= c;
qx <= OUT1 OR OUT2;

END xx;



--- Definición de SLATCH_PACKAGE

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

PACKAGE slatch_package IS

COMPONENT slatch

GENERIC  (
LPM_WIDTH : integer :=16
            );

PORT   (

dx                    : IN   STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
c                     : IN   STD_LOGIC;
a                     : IN   STD_LOGIC;
qx                    : OUT  STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0)

          );

END COMPONENT;

END slatch_package;
```

MÓDULO SINCT (ARQUITECTURA SERIE)

```
--- Componente SINCT:
--- Actualiza el registro de comparación
--- La salida lout Depende de las señales ctrl, ce
--- Señal rs de reset
---La señal  "enable" del FF  se habita antes de ctrl.


LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY sinct IS

PORT    (

ctrl         : IN   STD_LOGIC;
ce           : IN   STD_LOGIC;
rs           : IN   STD_LOGIC;
lout         : OUT  STD_LOGIC
           );

END sinct;

ARCHITECTURE xx OF sinct IS

---Definición de variables auxiliares:
signal AA,BB      : STD_LOGIC;
signal QL,QH      : STD_LOGIC_VECTOR(0 DOWNTO 0);
signal DL,DH      : STD_LOGIC_VECTOR(0 DOWNTO 0);

BEGIN

---definición de componentes:
prueba1   : lpm_ff
GENERIC MAP ( LPM_WIDTH => 1, LPM_FFTYPE => "TFF" )
PORT MAP (data => DL,
q => QL,
clock => AA, aclr => rs,
enable => ce
                );

prueba2   : lpm_ff
GENERIC MAP ( LPM_WIDTH => 1, LPM_FFTYPE => "TFF" )
PORT MAP (data => DH,
q => QH,
clock => BB, aclr => rs,
enable => ce
                );

AA <= NOT ctrl;
BB <= ctrl;
DL(0) <= '1';
DH(0) <= '1';
lout <= QL(0) XOR QH(0);

END xx;
```

```
--- Definición de SINCT_PACKAGE

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;


LIBRARY lpm;
USE lpm.lpm_components.ALL;


PACKAGE sinct_package IS

COMPONENT sinct

PORT   (

ctrl        : IN   STD_LOGIC;
ce          : IN   STD_LOGIC;
rs          : IN   STD_LOGIC;
lout        : OUT  STD_LOGIC
           );

END COMPONENT;

END sinct_package;
```

MÓDULO CELL (ARQUITECTURA SERIE)

```
--- Componente CELL:
--- Bloque de comparación serie (conforma el circuito-base de comparación serie)
--- La entrada es el vector de N  bits
--- La salida se conecta al siguiente bloque


LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

PACKAGE slatch_package IS

COMPONENT slatch

GENERIC  (
LPM_WIDTH : integer :=16
              );

PORT   (

dx                    : IN   STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
c                     : IN   STD_LOGIC;
a                     : IN   STD_LOGIC;
qx                    : OUT  STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0)
         );

END COMPONENT;

END slatch_package;


PACKAGE sinct_package IS

COMPONENT sinct

PORT   (

ctrl        : IN   STD_LOGIC;
ce          : IN   STD_LOGIC;
rs          : IN   STD_LOGIC;
lout        : OUT  STD_LOGIC
           );

END COMPONENT;

END sinct_package;


LIBRARY work;
USE work.slatch_package.ALL;
USE work.sinct_package. ALL;

ENTITY cell IS

GENERIC  (
LPM_WIDTH : integer :=16
              );

PORT  (

qain       : IN    STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0);
```

```
retra       : IN   STD_LOGIC;
rin         : IN   STD_LOGIC;
preset    : IN   STD_LOGIC;
dataout   : OUT   STD_LOGIC_VECTOR(LPM_WIDTH - 1 DOWNTO 0)
        );

END cell;

ARCHITECTURE xx OF cell IS

---Definición de variables auxiliares:
signal qa, qr, qb                  : STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
signal loadr, comp,  notsel        : STD_LOGIC;
signal zin                         :  STD_LOGIC_2D(1 DOWNTO 0 , LPM_WIDTH-1 DOWNTO 0);
signal selmux,  detec              : STD_LOGIC_VECTOR(0 DOWNTO 0);
signal ain                         :  STD_LOGIC_2D( LPM_WIDTH-1 DOWNTO 0 , 0 DOWNTO 0);

BEGIN

---Interconexión de componentes:
compra      : slatch PORT MAP (dx => qain, c => rin, a => preset,
qx => qa);

compr2      : slatch PORT MAP (dx => qa, c => loadr, a => preset,
qx => qr);

compr1      : slatch PORT MAP (dx => qr, c => rin, a => preset,
qx => qb);

compsin     : sinct PORT MAP ( ctrl=> retra , ce => notsel, rs  => preset,
lout => loadr);

prueba1   : lpm_compare
GENERIC MAP (LPM_WIDTH => LPM_WIDTH, LPM_REPRESENTATION => "UNSIGNED")
PORT MAP (dataa => qa, datab => qb,
alb => comp
                );

prueba2   : lpm_mux
GENERIC MAP (LPM_WIDTH => LPM_WIDTH, LPM_SIZE => 2, LPM_WIDTHS => 1)
PORT MAP (data => zin,
result => dataout,
sel => selmux
                );

prueba3   : lpm_or
GENERIC MAP (LPM_WIDTH => 1, LPM_SIZE => LPM_WIDTH )
PORT MAP (data => ain,
result => detec
                );

---Lazo de asignación:
L1: for i in 0 to LPM_WIDTH-1 generate
zin(1,i) <= qa(i);
zin(0,i) <= qb(i);
ain(i,0) <= qa(i);

end generate;

selmux(0) <= detec(0) AND comp;
notsel <= NOT selmux(0);

END xx;
```

DESCRIPCION DE LOS MÓDULOS VHDL-logiBLOX

MÓDULO LATCH

```
-- MÓDULO LATCH DE 16 BITS
-- LogiBLOX DATA_REG Module "lat_16b"
-- Created by LogiBLOX version M1.5.19
--    MODTYPE = DATA_REG
--    BUS_WIDTH = 16
--    STYLE = CLB_FD
--    ASYNC_VAL = 1
--    USE_RPM = FALSE
-- This is a behaviorial model only and cannot be synthesized.
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
LIBRARY logiblox;
USE logiblox.mvlutil.ALL;
USE logiblox.mvlarith.ALL;
USE logiblox.logiblox.ALL;
ENTITY lat_16b IS
  PORT(
    D_IN: IN std_logic_vector(15 DOWNTO 0);
    ASYNC_CTRL: IN std_logic;
    CLOCK: IN std_logic;
    Q_OUT: OUT std_logic_vector(15 DOWNTO 0));
END lat_16b;
ARCHITECTURE sim OF lat_16b IS
    SIGNAL START_PULSE: std_logic := '1';
BEGIN
  PROCESS
    VARIABLE VD_IN: std_logic_vector(15 DOWNTO 0);
    VARIABLE VASYNC_CTRL: std_logic;
    VARIABLE VSYNC_CTRL: std_logic;
    VARIABLE VCLK_EN: std_logic;
    VARIABLE VCLOCK: std_logic;
    VARIABLE VQ_OUT: std_logic_vector(15 DOWNTO 0);
    BEGIN
      VD_IN := stdvec2mvl(D_IN);
      VASYNC_CTRL := stdbit2mvl(ASYNC_CTRL);
      VSYNC_CTRL := '0';
      xb_data_reg(
            START_PULSE,
            VD_IN,
            VASYNC_CTRL,
            ('0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','1'),
            VSYNC_CTRL,
            ('0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0'),
            VQ_OUT);
      IF(
       (CLOCK'EVENT AND stdbit2mvl(CLOCK)='1' AND stdbit2mvl(CLOCK'LAST_VALUE)='0')
       OR (ASYNC_CTRL'EVENT AND stdbit2mvl(ASYNC_CTRL) /= '0')
       OR ( stdbit2mvl(START_PULSE)='1')
      ) THEN
      Q_OUT <= VQ_OUT;
      ELSIF(
       (stdbit2mvl(CLOCK) = 'X' AND stdbit2mvl(ASYNC_CTRL) /= '1')
      ) THEN
      VQ_OUT := ('X','X','X','X','X','X','X','X','X','X','X','X','X','X','X','X');
      Q_OUT <= VQ_OUT;
      END IF;
      IF (START_PULSE='1') THEN
        START_PULSE <= '0' AFTER 1 ns;
      END IF;
      WAIT ON D_IN, ASYNC_CTRL, CLOCK, START_PULSE;
  END PROCESS;
END sim;
```

MÓDULO COMPARADOR

```
-- MÓDULO COMPARADOR DE 16 BITS
-- LogiBLOX COMPARE Module "cmp16b"
-- Created by LogiBLOX version M1.5.19
--    MODTYPE = COMPARE
--    BUS_WIDTH = 16
--    STYLE = MAX_SPEED
--    ENCODING = UNSIGNED
-- This is a behaviorial model only and cannot be synthesized.

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
LIBRARY logiblox;
USE logiblox.mvlutil.ALL;
USE logiblox.mvlarith.ALL;
USE logiblox.logiblox.ALL;

ENTITY cmp16b IS
  PORT(
    A: IN std_logic_vector(15 DOWNTO 0);
    B: IN std_logic_vector(15 DOWNTO 0);
    A_LE_B: OUT std_logic);
END cmp16b;

ARCHITECTURE sim OF cmp16b IS
    SIGNAL START_PULSE: std_logic := '1';
BEGIN
  PROCESS
    VARIABLE VA: std_logic_vector(15 DOWNTO 0);
    VARIABLE VB: std_logic_vector(15 DOWNTO 0);
    VARIABLE VA_EQ_B: std_logic;
    VARIABLE VA_NE_B: std_logic;
    VARIABLE VA_LT_B: std_logic;
    VARIABLE VA_GT_B: std_logic;
    VARIABLE VA_LE_B: std_logic;
    VARIABLE VA_GE_B: std_logic;
    BEGIN
      VA := stdvec2mvl(A);
      VB := stdvec2mvl(B);
      xb_compare(
         VA,
         VB,
         UNSIGNED,
         VA_EQ_B,
         VA_NE_B,
         VA_LT_B,
         VA_GE_B,
         VA_GT_B,
         VA_LE_B);
      A_LE_B <= VA_LE_B;
      IF (START_PULSE='1') THEN
        START_PULSE <= '0' AFTER 1 ns;
      END IF;
      WAIT ON A, B, START_PULSE;
  END PROCESS;
END sim;
```

MÓDULO MULTIPLEX

```
-- MÓDULO MULTIPLEX DE 16 BITS
-- LogiBLOX MUX Module "mux16b"
-- Created by LogiBLOX version M1.5.19
--    MODTYPE = MUX
--    BUS_WIDTH = 16
--    OPTYPE = TYPE_2
--    INPUT_BUSSES = 2
--    ENCODING = BINARY
--    STYLE = MAX_SPEED
-- This is a behaviorial model only and cannot be synthesized.

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
LIBRARY logiblox;
USE logiblox.mvlutil.ALL;
USE logiblox.mvlarith.ALL;
USE logiblox.logiblox.ALL;

ENTITY mux16b IS
  PORT(
    S: IN std_logic;
    O: OUT std_logic_vector(15 DOWNTO 0);
    MA: IN std_logic_vector(15 DOWNTO 0);
    MB: IN std_logic_vector(15 DOWNTO 0));
END mux16b;

ARCHITECTURE sim OF mux16b IS
    SIGNAL START_PULSE: std_logic := '1';
BEGIN
  PROCESS
    VARIABLE VS: std_logic;
    VARIABLE VO: std_logic_vector(15 DOWNTO 0);
    VARIABLE VMA: std_logic_vector(15 DOWNTO 0);
    VARIABLE VMB: std_logic_vector(15 DOWNTO 0);
    VARIABLE VMC: std_logic_vector(15 DOWNTO 0);
    VARIABLE VMD: std_logic_vector(15 DOWNTO 0);
    VARIABLE VME: std_logic_vector(15 DOWNTO 0);
    VARIABLE VMF: std_logic_vector(15 DOWNTO 0);
    VARIABLE VMG: std_logic_vector(15 DOWNTO 0);
    VARIABLE VMH: std_logic_vector(15 DOWNTO 0);
    BEGIN
      VS := stdbit2mvl(S);
      VMA := stdvec2mvl(MA);
      VMB := stdvec2mvl(MB);
      xb_mux_type_2(
            VMA,
            VMB,
            VMC,
            VMD,
            VME,
            VMF,
            VMG,
            VMH,
            VS,
            BINARY,
            VO);
      O <= VO;
      IF (START_PULSE='1') THEN
        START_PULSE <= '0' AFTER 1 ns;
      END IF;
      WAIT ON S, MA, MB, START_PULSE;
  END PROCESS;
END sim;
```