UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ESCUELA UNIVERSITARIA DE
INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

Síntesis e Implementación de un Receptor SDH en Spartan-3

TITULACIÓN: **INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN**
ESPECIALIDAD EN SISTEMAS ELECTRÓNICOS

TUTORES : **Dr. D. ROBERTO ESPER-CHAÍN FALCÓN**
D. RUBÉN PASCUAL ARTEAGA MESA

AUTOR : **D. OLIVERIO RAMÍREZ SANTANA**

FECHA : **JULIO DEL 2007**

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ESCUELA UNIVERSITARIA DE
INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

Síntesis e Implementación de un Receptor SDH en Spartan 3

Presidente: Secretario: Vocal:

Tutores: Autor:

NOTA :

TITULACIÓN: INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN
ESPECIALIDAD EN SISTEMAS ELECTRÓNICOS

TUTORES : Dr. D. ROBERTO ESPER-CHAÍN FALCÓN
D. RUBÉN PASCUAL ARTEAGA MESA

AUTOR : D. OLIVERIO RAMÍREZ SANTANA

FECHA : JULIO DEL 2007

Índice

PARTE IV: CONCLUSIONES

PARTE V: BIBLIOGRAFÍA

PARTE VI: PRESUPUESTO

PARTE VII: APÉNDICES

Índice de Figuras

Índice de Tablas

Peticionario

Actúa como peticionario la Escuela Universitaria de Ingeniería Técnica de Telecomunicación, como requisito indispensable para la obtención del título de Ingeniero Técnico de Telecomunicaciones en la especialidad de Sistemas Electrónicos.

PARTE I
INTRODUCCIÓN

Capítulo 1

Introducción

El mundo parece moverse a pasos agigantados con los nuevos avances tecnológicos que se suceden continuamente. En ningún otro campo son estos descubrimientos tan rápidos y excitantes como en el mundo de las telecomunicaciones. Dentro de estos avances la industria introdujo una nueva tecnología cuya misión es transportar y gestionar gran cantidad de tipos de tráfico diferentes sobre la misma infraestructura física. Esta tecnología es la *SDH* (Jerarquía Digital Síncrona) y su equivalente norteamericana *SONET* (Red Óptica Síncrona).

1.1 Planteamiento del problema

Antes de que hubieran ordenadores que necesitaran conectarse para compartir recursos y comunicarse, las compañías telefónicas construyeron una red internacional que transportaba las llamadas. Estas redes de área extensa fueron optimizadas para transportar

múltiples llamadas telefónicas punto a punto usando hilo de cobre. Cuando las limitaciones del cable de cobre afectaron a los nuevos servicios, los operadores actualizaron el medio físico a fibra óptica.

Los operadores vieron la fibra óptica como una parte esencial de su futuro al conocer el ancho de banda que proporcionaba. Sin embargo, otras limitaciones de la red telefónica todavía permanecían. Entre ellas, la falta de estándares que permitieran la conexión de equipos de diferentes fabricantes que afectaba tanto al hilo de cobre como a la nueva red y el coste del mantenimiento de la nueva red.

A lo largo del tiempo, los ordenadores fueron tomando posiciones dentro del mundo. La conexión global de todos estos equipos fue deseable y beneficiosa. Cuando los ordenadores se conectaban a larga distancia, usaban las antiguas redes de área extensa transmitiendo datos en lugar de voz, por lo que estas redes no se comportaban de manera eficiente. Por lo tanto, alguna redes fueron construidas específicamente para transportar datos. En este momento se hizo evidente la idea de disponer de una única red para la transferencia de datos, voz e imagen.

1.2 Motivación del proyecto

Para dirigir estas propuestas, el *CCITT* (actualmente *ITU-T*) y otros grupos de estandarización empezaron a trabajar en la década de los años ochenta con el fin de establecer una serie de recomendaciones sobre las técnicas de transmisión, conmutación, señalización y control. Esta red, basada en la fibra óptica, debería resolver los distintos problemas y permitir el transporte eficiente de todo tipo de información. Esta red fue llamada fue llamada Red Digital de Servicios Integrados de Banda Ancha (*RDSI-BA*). En 1990, se tomó la decisión de basar *RDSI-BA* en *SDH/SONET*.

SONET describe los estándares ópticos para la transmisión de datos. El estándar *SONET/SDH* especifica cómo la información puede ser encapsulada, multiplexada y transmitida sobre una red óptica. Los principales objetivos fueron asegurar que la interoperabilidad y control de los equipos de distintos fabricantes fuera posible, y que la transmisión y multiplexación de los datos fuera independiente del tipo de datos, además de

posibilitar la introducción de nuevos servicios sin modificar la red. Esto se consigue empaquetando los datos en unos contenedores para insertarlos posteriormente en el área de carga útil de las señales *SDH*, denominadas tramas, que es lo que finalmente se transmite.

Una de las principales funciones de un equipo de transmisión *SDH* es, por tanto, la inserción y extracción de señales de dichos contenedores en las tramas *SDH*, función que puede desempeñar perfectamente el transceptor *SDH* de la figura 1.1.

El transmisor es el encargado de la inserción de los contenedores y el receptor de su extracción. El serializador no es más que un multiplexor utilizado para la transmisión de las tramas resultantes, ya que éstas deben ser enviadas en serie. De esta forma, el transmisor puede procesar las tramas en paralelo con una señal de reloj de frecuencia menor a la de transmisión. De la misma forma, las tramas recibidas son demultiplexadas por el paralelizador para que el receptor pueda procesarlas también en paralelo.

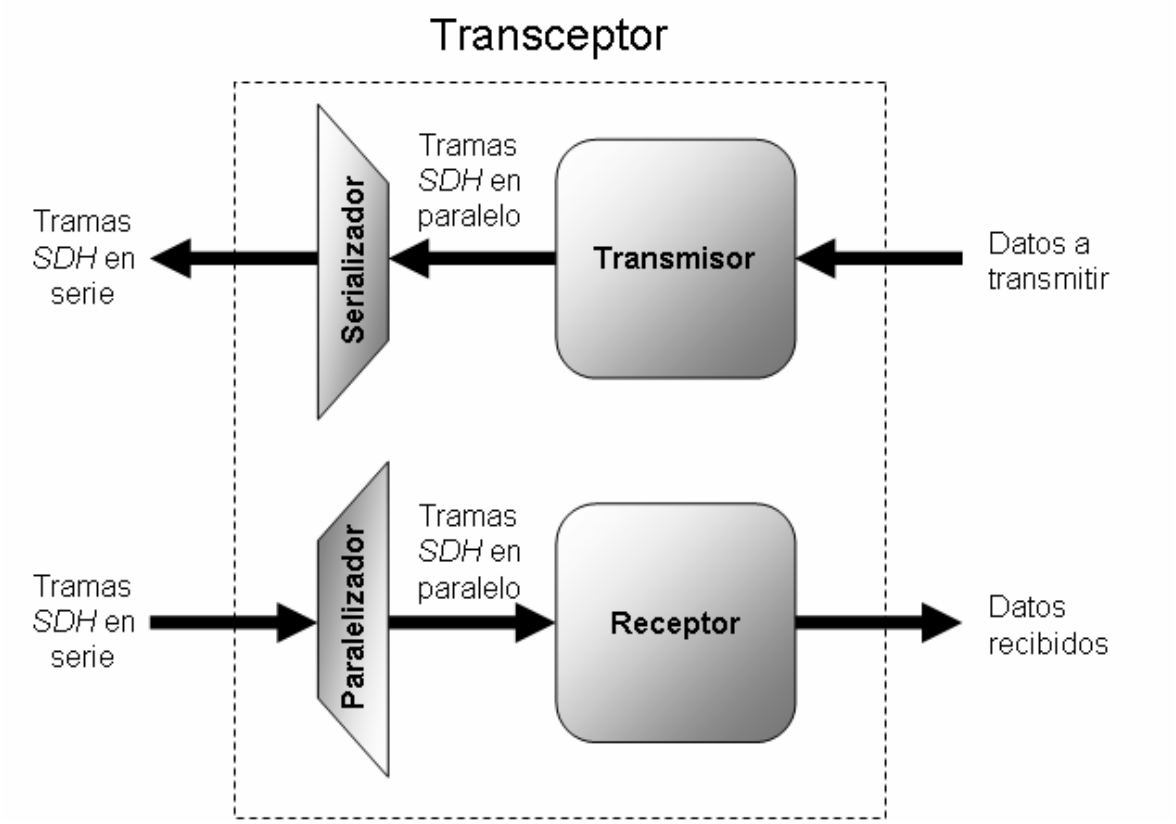


Figura 1.1: Transceptor *SDH*

1.3 Objetivo del proyecto

Teniendo en mente estas ideas, el siguiente proyecto propone el estudio del estándar *SDH* para desarrollar un receptor que procese las tramas *SDH*, extrayendo los datos transportados en la misma, así como ciertos octetos de sus taras. Este proyecto va en conjunto con otro sobre un transmisor *SDH*, que se encarga de realizar el proceso contrario, es decir, introducir datos que se deseen enviar en tramas *SDH* que los transportarán. El receptor y el trasmisor formarán parte de un sistema superior, el transceptor *SDH*. Además, un proyecto de esta envergadura obliga a seguir una metodología de diseño exigente que permita asegurar la calidad del producto final.

Para conseguir este objetivo, una vez efectuado el estudio teórico y analizada la viabilidad del proyecto, y siguiendo lo que sería un flujo de diseño típico para un dispositivo electrónico, se definirán, en primer lugar las especificaciones del sistema.

Posteriormente se presentará la implementación física elegida y su división en unidades funcionales.

Una vez realizados los dos primeros pasos, se procederá a la descripción del sistema mediante el lenguaje de descripción *hardware Verilog* a nivel de transferencia de registros (*RTL*) y a continuación se realizará la verificación funcional, a través de simulación, del mismo mediante la definición de varios tests que permitan asegurar que el resultado obtenido se ajusta a las especificaciones.

Finalmente se sintetizará el código resultante y se implementará en la *FPGA XC3S1000* de la serie *Spartan-3* de *Xilinx* y se verificará mediante la placa de prototipado *Diligent Spartan-3 Starter Board*.

El desarrollo del trabajo realizado en el presente Proyecto de Fin de Carrera se ha estructurado en siete capítulos, siendo el primero la presente introducción, y cuyo contenido se detalla a continuación:

Capítulo 2. En él se presenta un breve estudio teórico sobre las comunicaciones digitales, particularizando en la *SDH*.

Capítulo 3. En él se describen las características de la *FPGA* de la serie *Spartan-3* de *Xilinx*, y de la placa de prototipado *Diligent Spartan-3 Starter Board*.

Capítulo 4. En él se pretende explicar cuáles son los diferentes pasos a seguir en el diseño de dispositivos electrónicos, describiendo brevemente la función que se realiza en cada uno de ellos. Se detallan los aspectos más relevantes de los lenguajes de descripción *hardware* y en particular del lenguaje *Verilog*, así como la manera más adecuada de utilizarlo en el diseño de un dispositivo electrónico, resaltándose las consideraciones a seguir para la descripción de un sistema a nivel *RTL*.

Capítulo 5. En él se exponen las especificaciones generales del sistema, su implementación física a través de una arquitectura modular y la descripción funcional de cada uno de los módulos que lo forman.

Capítulo 6. En él se presentan las verificaciones realizadas, tanto a nivel de módulo como a nivel de sistema, de las cuales se desprende el correcto funcionamiento del sistema.

Capítulo 7. En él se presentan las conclusiones que se obtienen de este proyecto así como las líneas futuras a seguir si ello se considerase oportuno.

PARTE II

ESTUDIOS PREVIOS

Capítulo 2

SDH

Los sistemas de transmisión digital se pueden describir extensamente como una colección de procesos encargados de la multiplexación, la creación de las tramas, transporte, enrutado, temporización y protección de los datos. Los detalles de cada proceso y la forma en la cual se relacionan, se describen como una jerarquía de transmisión digital. Cada uno de estos procesos se puede descomponer en un conjunto de tareas menores. Las jerarquías digitales de transmisión se pueden descomponer en un conjunto de capas cada una de las cuales contiene estas tareas.

En el presente capítulo se pretenden sintetizar las características imprescindibles de las jerarquías digitales y en particular de la *SDH*. Para más información, se pueden consultar las referencias [1], [2], [3] y [4] de la bibliografía.

2.1 Introducción a las redes digitales

Desde mediados del siglo pasado, la invención del transistor cambió la forma de vivir y comunicarse del planeta. Este dispositivo permitió la evolución de la ingeniería electrónica alcanzando espectaculares avances en los últimos 50 años.

Hemos asistido a la aparición de los telégrafos, teléfonos, ordenadores, redes de comunicaciones de datos, . . . y a un sinfín de aplicaciones que propician la comunicación a través del mundo de una forma transparente.

La evolución de los antiguos servicios y la aparición de otros nuevos, el uso más generalizado de la fibra óptica y la globalización de la red han provocado la necesidad de integrar y estandarizar todos los servicios de voz y datos en una única red.

El teléfono, internet, la videoconferencia, la música o el cine son servicios que deben superar sin problemas las fronteras de los diferentes países y unificar regiones que han desarrollado sus propios estándares. Para cumplir estos requisitos se asume una red digital de alta velocidad y ámbito mundial que soporte todas las aplicaciones anteriores.

2.1.1 Características de las redes de datos digitales

Los servicios pueden ser integrados con mucha mayor facilidad en las redes digitales que en las analógicas, esto se debe a que se han estudiado técnicas de conversión, digitalización y compresión de los datos para el mejor aprovechamiento de los recursos. Gracias a estas características, estos servicios pueden ser multiplexados en el tiempo constituyendo un único flujo de data de alta velocidad.

La transmisión digital presenta características innatas que le dan ventajas sobre las antiguas redes analógicas:

- Alta inmunidad al ruido

Cualquier señal que se propague por un medio se verá afectada por diversas fuentes de ruido degradando la calidad de ésta. Cuando se sitúa un repetidor en "el lugar apropiado" se amplificarán tanto la señal como el ruido. Por lo tanto, a lo largo de una línea larga el ruido se irá acumulando, de tal forma que una vez alcance su destino puede ser que la relación señal a ruido haya bajado enormemente y la señal no pueda ser recuperada.

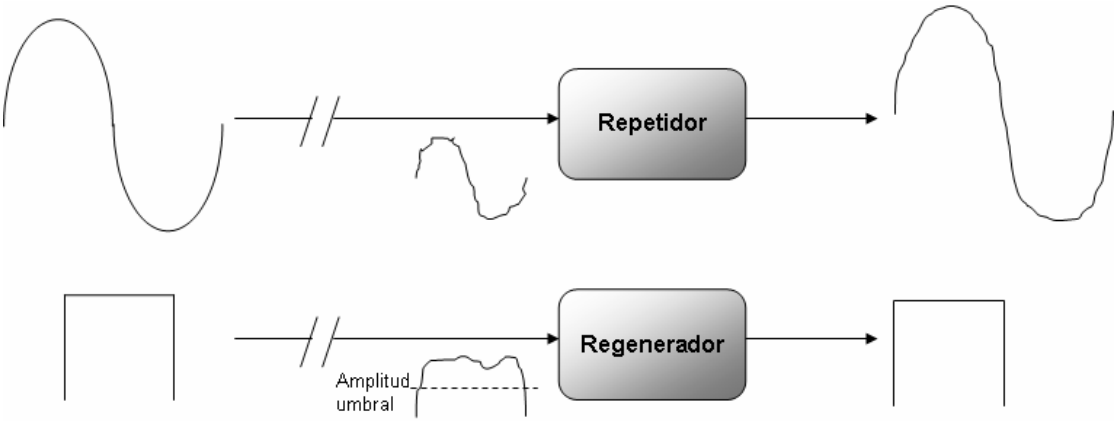


Figura 2.1: Transmisión analógica frente a digital

Por el contrario, en una transmisión digital, lo que se utilizan son módulos regeneradores. Dichos bloques se situarán en los lugares precisos. Si la señal entrante en el módulo regenerador supera un cierto umbral de amplitud, la salida será un nivel alto, es decir un "1" lógico; si esto no ocurriese el módulo proporcionaría a su salida un nivel bajo, un "0" lógico. De esta forma el ruido no se propaga por toda la línea.

- Integración de servicios

Permite la integración de televisión, voz, datos, etc., en un mismo medio. Este hecho resulta de vital importancia dada la demanda de aplicaciones relacionadas con información multimedia o sistemas colaborativos y su coexistencia con aplicaciones más clásicas, sobretodo a partir de la aparición de Internet. Para que las nuevas

tecnologías en comunicaciones puedan ofrecer un buen servicio es necesario revisar, potenciar y ampliar las actuales arquitecturas, servicios y protocolos de comunicaciones.

- Facilidad de multiplexación

Como puede verse en la figura 2.2 para multiplexar varias señales analógicas se aplica una *MDF* (multiplexación por división en frecuencia). Cada señal se modula y filtra para restringir la señal a su banda de frecuencias. El tercer paso es la mezcla de las diferentes señales.

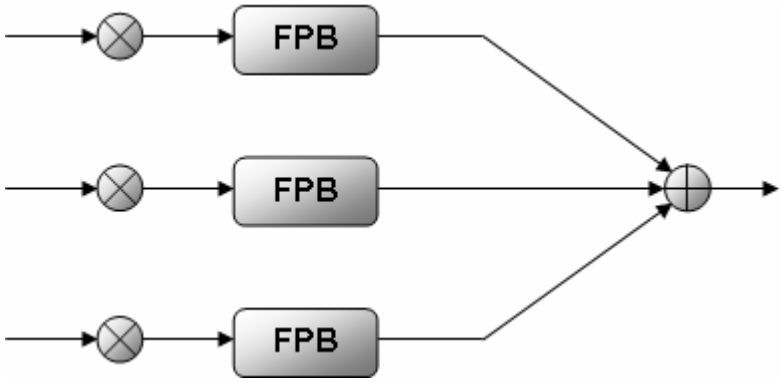


Figura 2.2: Multiplexación por división en frecuencia (analógico)

Para realizar el multiplexado de señales digitales se utiliza la multiplexación por división en el tiempo o *MDT*. En la figura 2.3 se aplica este procedimiento a señales de baja velocidad, las cuales se introducen en un conmutador lógico y se mezclan formando un flujo de datos de alta velocidad. Como se observa, el multiplexado digital es más sencillo que el analógico.

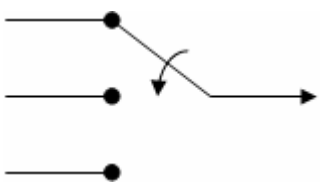


Figura 2.3: Multiplexación por división en el tiempo (digital)

También se puede utilizar la *MDT* con señales analógicas pero el procedimiento es más complejo que con las señales digitales.

2.1.2 Aparición de *SONET/SDH*

A medida que se iba haciendo efectivo el paso de la transmisión analógica a la digital fueron surgiendo nuevas redes digitales que poseían diferentes estructuras, velocidades y protocolos. Esta diversidad de redes hacía que las interfaces entre las mismas fuesen muy complejas y a menudo imposibles.

A principios de los 80 y bajo este marco, nace *SONET* (red óptica síncrona) en Norteamérica. *SONET* es una red óptica síncrona que implementa una familia de estándares o protocolos. Surge como el nexo de unión de las diferentes redes digitales existentes en ese momento e incluso de las futuras, permitiendo la interconexión de equipamiento de diferentes fabricantes.

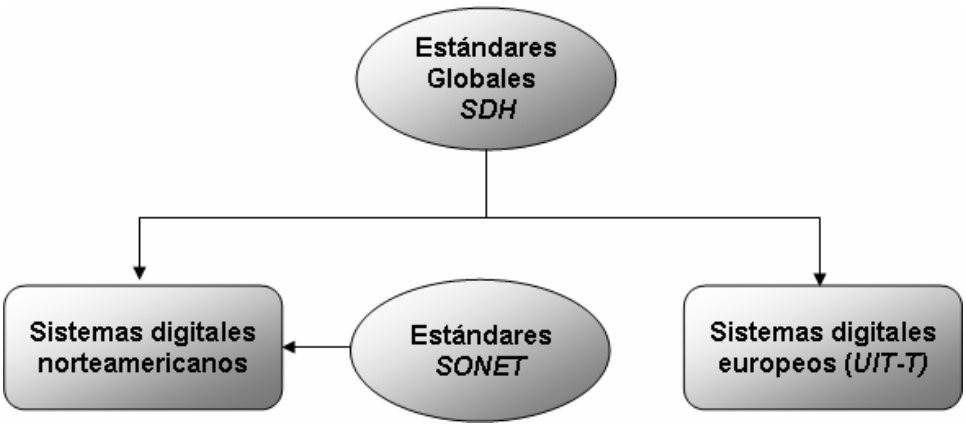


Figura 2.4: Estándares *SONET* y *SDH*

Más tarde aparece la *SDH* (jerarquía digital síncrona). Podría verse a la *SONET* como madre de la *SDH*, no obstante esto no es del todo exacto ya que, a pesar de haber surgido y

de haberse basado en la *SONET* para la creación de la *SDH*, esta última es un conjunto de estándares dentro de los cuales se hallan contenidos los estándares de la *SONET*. En definitiva, puede decirse que la *SONET* es un subconjunto de la *SDH*.

Como puede observarse en la figura 2.4, la red digital global está dividida en dos partes: La red digital norteamericana y la red digital *UIT- T*, que tiene su base en los países europeos. Los estándares *SDH* son aplicables para ambas redes mientras que *SONET* fue inventada para la red digital norteamericana.

En un futuro próximo toda aquella compañía que desee tener un ámbito internacional deberá utilizar *SDH*.

2.1.3 *SDH* vs *PDH*

Desde el principio se ha intentado conseguir una red perfectamente sincronizada, no obstante, esto requería una distribución de reloj muy buena y un oscilador maestro muy preciso. Hasta hace unos años esto no era posible, por lo que se creó una jerarquía cuasi síncrona llamada *PDH* (jerarquía digital plesiócrona), aún vigente en nuestros días.

El término plesiócrono indica la característica de una señal en virtud de la cual, sus instantes significativos se presentan con la misma cadencia nominal y cualquier variación de esta cadencia se mantiene dentro de límites especificados. Dos señales que tengan la misma velocidad digital nominal y no provengan del mismo reloj serán, siempre, plesiócronas.

Una de las razones por las cuales ha sido desarrollada la *SDH* es la imperfección de las características de la *PDH*, relatadas a continuación y comentadas en las siguientes secciones:

- Existencia de tres jerarquías digitales diferentes.

- Varios formatos de señal y velocidades.

- Diferentes esquemas de supresión de ceros.
- Diversos procedimientos de multiplexado.
- Varios porcentajes de redundancia
- Limitada escalabilidad.

2.1.3.1 Jerarquías digitales *PDH*

Actualmente existen 3 jerarquías digitales en uso:

- Jerarquía norteamericana: Utilizada en USA, Canadá, Taiwán y Corea del Sur. Su señal base es *DS0* cuya velocidad es de 64 Kbps, resultado de muestrear la señal de voz a 8000 muestras por segundo y codificada siguiendo la ley μ de 8 bits.
- Jerarquía europea: Su señal base es *E0*, presenta una velocidad de 64 Kbps. Es, o será utilizada en los países europeos, asiáticos (exceptuando Taiwán, Corea del Sur y Japón), africanos y en países del centro y sur de América. La diferencia entre la *DS0* y la *E0* consiste en que en esta última se utiliza la ley *A* para la codificación de la señal de voz.
- Jerarquía japonesa: En vías de desaparición.

  Debido a la existencia de estas tres jerarquías, para realizar la interconexión de las diferentes regiones, será necesario convertir la señal de un formato a otro. Evidentemente, si sólo existiese una única jerarquía digital, como es el caso de la *SONET/SDH*, esta conversión sería del todo innecesaria.

Como se muestra en la figura 2.5, esta mezcla de varias jerarquías digitales *PDH* en una única jerarquía digital hará que la interconexión sea más sencilla. La señales *OC-N* pertenecientes a la *SONET* (N=1..192) y *STM-N* (N=1..64) pertenecientes a la *SDH* se tratarán más tarde. No obstante, puede observarse que entre ambas jerarquías existen equivalencias.

USA	**Europa**	**Japón**
64 Kbps [DS0]	64 Kbps [E-0]	64 Kbps
1,544 Mbps [DS1]*		1,54 Mbps
	2,048 Mbps [E-1;CEPT-1]*	
3,152 Mbps [DS1C]*		3,152 Mbps*
6,312 Mbps [DS2]*		6,312 Mbps*
	8,448 Mbps [E-2;CEPT-2]	
		32,064 Mbps
	34,368 Mbps [E-3;CEPT-3]	
44,736 Mbps [DS3]*		
91,053 Mbps [DS3C]*		
		97,728 Mbps
	139,264 Mbps [E-4;CEPT-4]*	
274,176 Mbps [DS4]		
		397,2 Mbps

Estas señales pueden ser incluidas como carga útil *SONET*

Tabla 2.1: Las tres jerarquías digitales

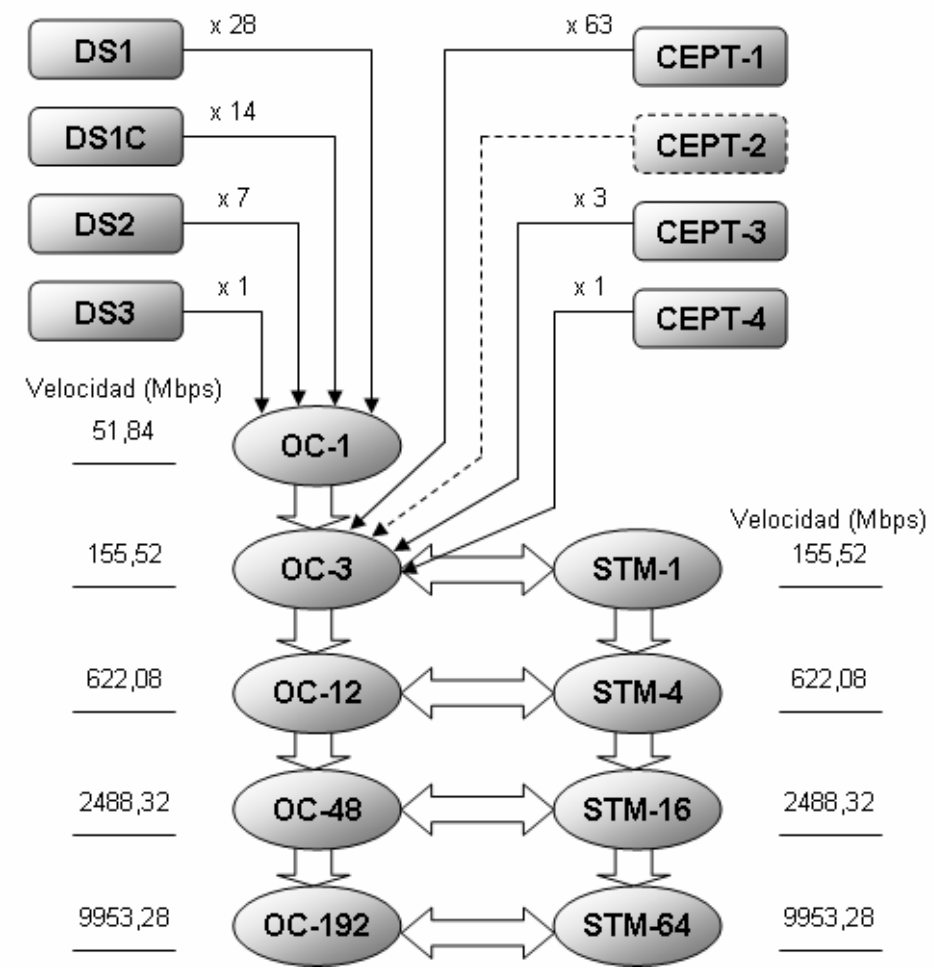


Figura 2.5: Interfaz de nodo de la red digital

2.1.3.2 Variedad de formatos y velocidades

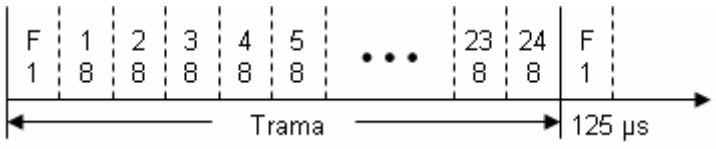


Figura 2.6: Formato de la señal *DS1*

Tanto la señal *DS1* como la *CEPT-1* tienen un periodo de trama de 125 µs, que es el intervalo entre dos muestras adyacentes de una señal analógica muestreada a 8 KHz, es

decir, 1/8000, que es la velocidad de muestreo de *Nyquist* para la digitalización de señales de voz.

Una trama *DS1* está formada por un bit de redundancia seguido por 24 segmentos temporales capaces de transportar 8 bits de información, ya sea voz, datos o señal de video. Por lo tanto cada trama (125 μs) estará formada por 193 bits (1 + (24 x 8)). El primer bit de cada trama está destinado a la sincronización. El receptor se sincronizará con 12 de estos bits, es decir, se necesitarán 12 tramas *DS1*. El conjunto de estas 12 tramas se denomina supertrama.

Además del sincronismo, todo sistema de transmisión digital requiere algún tipo de sistema de control, esto se denomina señalización y puede cumplir diferentes funciones: supervisión, función administrativa, mantenimiento, etc. No obstante, en una señal *DS1* no existen suficientes bits disponibles para estas operaciones, ya que de los 193 bits que conforman una trama, uno se utiliza para el encabezamiento y los 192 restantes se reparten entre los 24 usuarios o canales. Para poder transmitir la información de señalización dentro de cada supertrama *DS1*, se toman los bits menos significativos de cada segmento temporal en las tramas 6" y 12". A este método se le conoce con el nombre de *rob-bit*. Si se está transmitiendo voz, la degradación de la señal es imperceptible. Si son datos, entonces la señal no se codifica a 8 bits sino a 7 y esto lleva a que en cada segmento de tiempo exista un bit que se puede destinar a la señalización.

Sin embargo, para las redes *RDSI* este sistema no es válido, ya que ésta necesita un canal de 64 Kbps de señalización. Por lo tanto, para cubrir esta necesidad, se ha creado una nueva trama *DS1* que se ha llamado supertrama extendida (*STE*) formada por 2 supertramas, es decir, 24 tramas de 193 bits cada una. Aplicando este sistema no es necesario utilizar el método *rob-bit*, ya que se puede conseguir un canal de 64 Kbps reasignando los 24 bits de comienzo de trama que han sido distribuidos en sendas tramas que componen una *STE*, de forma que este bit puede desempeñar otras funciones además de la sincronización.

La primera diferencia existente entre una trama *CEPT-1* y una *DS1* es que la primera posee 32 canales, frente a los 24 de la segunda. Estos 32 segmentos temporales son transmitidos cada 125 μs. Cada trama está encabezada por un canal dedicado que transmite información

de control y sincronismo. El segmento 16 también se utiliza para la señalización. Por lo tanto una trama *CEPT-1* posee 30 canales destinados a la transmisión de información.

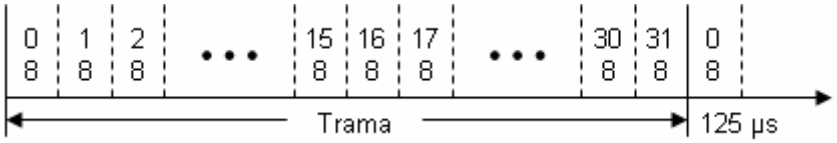


Figura 2.7: Formato de la señal *CEPT-1*

A pesar de las diferencias en cuanto al número de canales entre *DS1* y *CEPT-1* la diferencia más notoria radica en el modo de transmitir la redundancia. La *DS1* utiliza un método distribuido mientras que la *CEPT-1* lo utiliza concentrado.

Aunque el método distribuido muestra una mayor eficiencia en la transmisión que el concentrado, 99,48% ([1- 1/193] x 100) frente al 93,75% ([1- 2/32] x 100), el concentrado aventaja al distribuido en cuanto a su estructura puesto que se asemeja a la de la *RDSI*, es decir, señalización y datos separados.

	DS1	**CEPT-1**
Velocidad de la señal (Mbps)	1,544	2,048
Nº de Canales	24	32/30
Organización de la redundancia	Menos concentrado	Más concentrado

Tabla 2.2 Sumario de las señales *DS1* y *CEPT-1*

La solución de la SONET / SDH es la modularidad y la redundancia concentrado:

- Las señales son modulares. Por ejemplo, una *STM-N* tiene una velocidad de *N* x 155,52 Mbps (*N* veces la velocidad de una *STM-1*).

- Dentro de una trama de 125 µs los principales bits de redundancia están totalmente separados de la información.

2.1.3.3 Diferentes formas de ausencia de transiciones

Para que un receptor pueda recuperar la señal con una tasa de error aceptable, todo sistema digital necesita un método de codificación preciso y síncrono, que permita una buena recuperación del reloj, además de un canal con bajo ruido. Se hace necesaria por tanto la existencia de un buen reloj en el receptor que asegure un nivel mínimo de precisión, al igual que es imprescindible un buen sistema de recuperación de la información temporal.

En la figura 2.8 puede verse cómo se utiliza un *CRU* (unidad de recuperación de reloj) para sincronizar el reloj del receptor al flujo de datos entrante. Mediante este sistema puede minimizarse la tasa de bits erróneos.

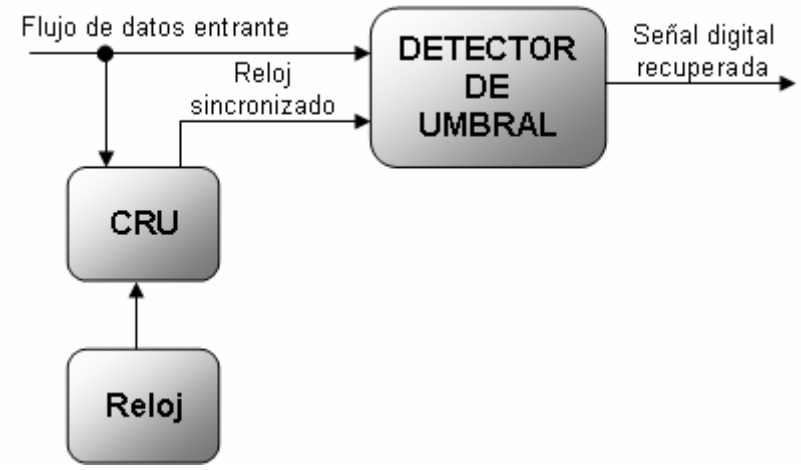


Figura 2.8: Recuperación de reloj en el receptor

Para que la recuperación temporal se lleve a cabo con éxito, el flujo de bits transmitido debe poseer suficiente densidad de transiciones, es decir, la señal digital debe presentar gran cantidad de cambios de uno a cero y de cero a uno. La tecnología utilizada para cambiar una señal carente de transiciones a una señal con gran número de éstas manteniendo la integridad de la información, se denomina tecnología de supresión de ceros. Dicha tecnología puede ser implementada utilizando un código de línea en particular o un aleatorizador.

Los códigos de línea más comunes son los siguientes:

- En la señal *DS1* se utiliza *AMI* o bipolar y *B8ZS*.
- Para la señal *DS2* se utiliza el *B6ZS*.
- Para la *DS3* el *B8ZS*.
- Para la *CEPT-1* y *CEPT-3* se utiliza el *HDB3*.
- Para la *CEPT-4*, se utiliza el *CMI*.
- En *Ethernet* (red de área local) se utiliza el código *Manchester*.

La *SONET/SDH* utiliza un aleatorizador para todas las tramas y el código de línea *NRZ*.

2.1.3.4 Varios procedimientos de multiplexado

En toda transmisión digital, las señales de los diferentes canales de baja velocidad serán multiplexados o mezclados en el tiempo, formando un flujo de datos de alta velocidad.

Como puede apreciarse en la figura 2.9, este multiplexado puede ser a nivel de bit o a nivel de byte. Ambos métodos eran utilizados indistintamente por los sistemas digitales. Por este motivo y para alcanzar un sistema de comunicación global se ha tenido que optar por uno solo de estos métodos.

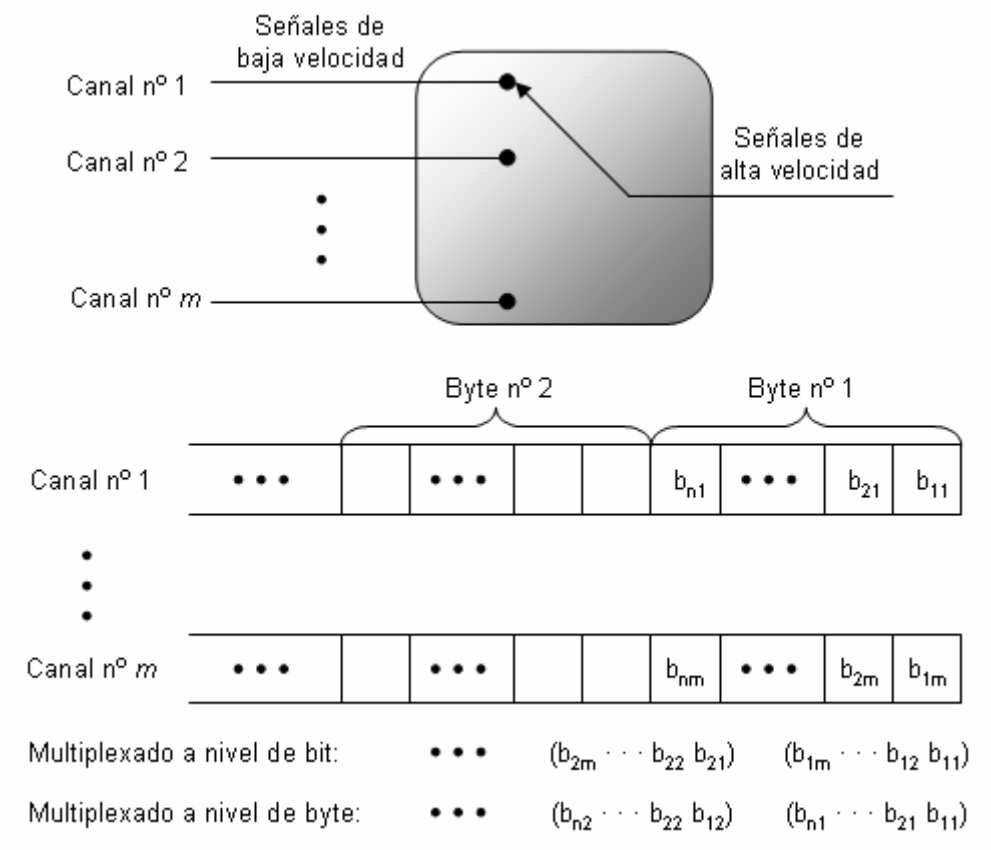


Figura 2.9: Tipos de multiplexado

Los productos *SONET / SDH* han optado por el multiplexado a nivel de byte. El *MSB* (bit más significativo) de un byte *SONET / SDH* de ocho bits, es siempre transmitido en primer lugar. La asignación en el byte es: $b_7b_6b_5b_4b_3b_2b_1b_0$, donde b_7 es el *MSB* y b_0 es el *LSB* (bit menos significativo).

2.1.3.5 Redundancia

Como ya se ha dicho, desde el principio se ha procurado elaborar una jerarquía cuyos diferentes niveles estuviesen sincronizados entre sí. Sin embargo esta estructura presenta numerosas dificultades, ya que esto significaría que todos los sistemas de transmisión deberían poseer una misma señal de reloj generada desde un oscilador maestro común. Hoy en día esto no funciona así:

- Los multiplexores de orden superior no están sincronizados con las señales del nivel inferior.

- Los sistemas de transmisión de orden superior utilizan técnicas de relleno (justificación), para compensar las ligeras desviaciones de velocidad de las señales en relación a la velocidad con la que son admitidas en el multiplexor, debido al uso de osciladores independientes.

Por lo tanto en cada nivel jerárquico además de los bits de control y señalización de trama, también se incorporarán bits de relleno (o de justificación) si fuesen necesarios. Todos estos bits, es decir, todo aquello que no forme parte de la carga útil, se denomina redundancia.

La fórmula de la redundancia es la siguiente:

$$encabezado(\%) = \frac{C_t - C_i}{C_t} \times 100\,,$$

donde C_t es la capacidad total de la trama y C_i representa la cantidad de información, es decir, la carga útil.

Como puede observarse en la tabla 2.3, a medida que la velocidad de transmisión aumenta la capacidad de la carga útil disminuye. Esto se debe al sistema de multiplexado utilizado en la *PDH*. Véase la figura 2.10, aquí el OH_1 es la redundancia introducido en la primera etapa de multiplexación, su valor es del 0,52% de la capacidad total de la trama. El OH_2 es del 2,66% y el OH_3 del 3,862%. El aumento paulatino del porcentaje de redundancia en los sucesivos multiplexores se debe a cierta redundancia existente entre los diferentes *OH*.

La *SONET/SDH* mantiene un porcentaje de redundancia fijo. La figura 2.11 muestra como la señal *OC-1* está constituida por 24 señales *DS1* (cada una de las cuales presenta una redundancia propio, *OH1*) más una redundancia *OH*. La redundancia alcanzada después de la primera etapa de multiplexado es del 17%, es decir un 83% de eficiencia en la

transmisión, esta eficiencia es comparativamente baja, no obstante y a diferencia de la *PDH*, en los sucesivos multiplexados no se hace necesaria la incorporación de redundancia adicional, manteniéndose este porcentaje constante para cualquier velocidad.

Una vez introducido las redes digitales, en el siguiente apartado se presenta una descripción un poco más profunda sobre el estándar *SDH*.

Señal digital	Porcentaje de capacidad	Porcentaje de redundancia
DS1	99,48	0,52
DS2	97,30	2,70
DS3	96,10	3,90
DS4	93,40	6,60
CEPT-1	93,75	6,25
CEPT-2	90,91	9,09
CEPT-3	89,40	10,60
CEPT-4	88,24	11,76

DSN: Señal digital de nivel *N*
CEPT-N: Señal digital de la *UIT-T* de nivel *N*

Tabla 2.3: Eficiencia de transmisión y porcentaje de redundancia

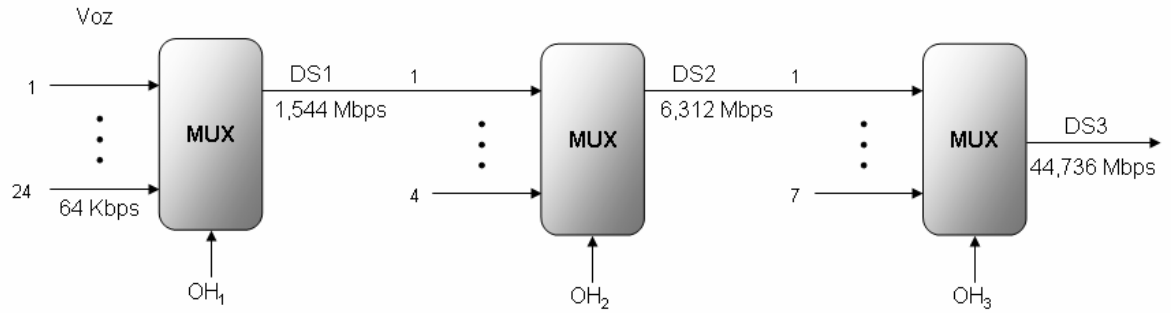


Figura 2.10: Inserción de la redundancia en la *PDH*

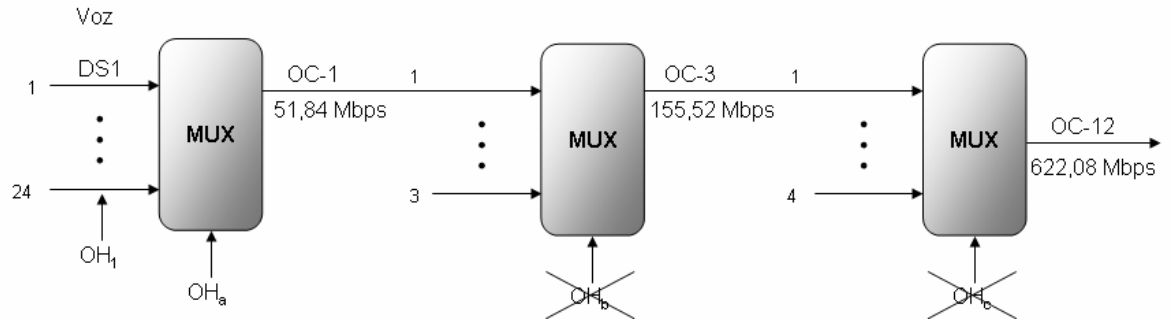


Figura 2.11: Inserción de la redundancia en la *SONET*

2.2 Red digital *SDH*

SDH ha proporcionado a la transmisión redes caracterizadas por ser independientes del vendedor y tener una estructura de señales sofisticadas que proporcionan un rico conjunto de características. Esto ha resultado en nuevas aplicaciones de red, la expansión de los nuevos equipos en las nuevas topologías, y el aumento de las capacidades de la administración de la red.

Las redes digitales incrementaron su complejidad a principios de los años 80. La demanda de los operadores de red y de sus clientes creció rápidamente provocando que los estándares de ese momento quedaran incapacitados para proporcionar el soporte a los nuevos servicios. La solución más evidente fue el aumento de la tasa de bits, pero quedó limitada por el alto precio del ancho de banda de transmisión y de los dispositivos digitales. Las técnicas de multiplexación permitieron la combinación de señales ligeramente asíncronas que condujeron a la creación de la *PDH*.

El desarrollo de la transmisión por fibra óptica y los circuitos integrados hicieron posible el aumento de la complejidad en los estándares. Hubieron peticiones para mejorar e incrementar los servicios que requirieron gran cantidad de ancho de banda, incluir utilidades para mejorar el rendimiento de la monitorización, y dotar de mayor flexibilidad a la red. Los dos principales factores que influyeron en el nuevo estándar fueron:

1. Propuestas del *CCITT* (Actualmente *ITU*) para un red digital de servicios integrados de banda ancha.

2. La liberalización de las comunicaciones en EEUU en 1984 obligó al desarrollo de un estándar que permitiese el intercambio de información entre los distintos operadores.

Se aceptó extensamente que el método de multiplexado fuera síncrono y a nivel de byte.

Los principales beneficios que se obtuvieron fueron:

- Reducción del coste:

    1. Incremento de la competencia entre fabricantes diseñando interfaces sencillos y compatibles.

    2. Diseño adecuado de la redundancia para facilitar las operaciones de administración y mantenimiento.

- Posibilidad de diseñar elementos de red integrados que permitieron diferentes topologías y funcionalidad.

- Uso de la fibra óptica para la eliminación de los cuellos de botella.

2.2.1 Conceptos y definiciones

A continuación se describen los conceptos fundamentales relacionados con la *SDH*. La figura 2.12 muestra la relación entre los diferentes elementos y las posibles estructuras de multiplexación que aquí se definen.

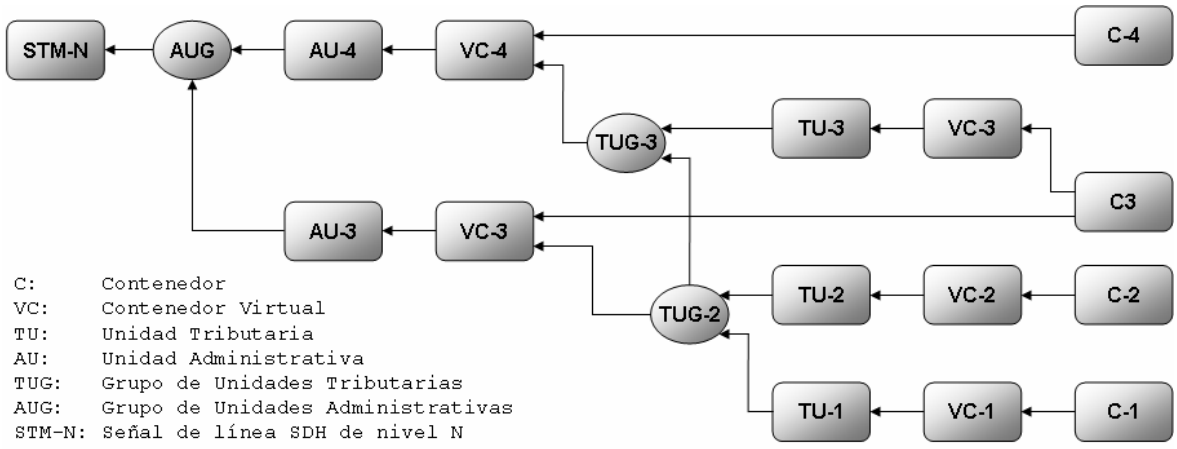


Figura 2.12: Estructura de multiplexación generalizada

2.2.1.1 Contenedor

El *C* (contenedor) es una unidad de carga útil dimensionada para poder transportar las señales de la *SDH*. Cada una de estas unidades es capaz de contener un solo tipo de señal que bien puede ser de origen plesiócrono, células *ATM* (módulo de transferencia asíncrono), canales de *N*x64 Kbps, etc.

2.2.1.2 Contenedor virtual

Desde el punto de vista de la *SDH*, la carga útil está constituida por subtramas de 4° orden (*VC-4*) y eventualmente de 3er (*VC-3*) y 1er orden (*VC-12*). Tanto las de primer como las de tercer orden, son asignadas directamente en espacios de carga resultantes de la subdivisión de la subtrama de cuarto orden (*VC-4*). Por otro lado la subtrama de cuarto orden es asignada directamente en el espacio de carga de la trama *STM-1* (módulo de transporte síncrono de nivel 1), que es la señal base en *SDH*. Estas subtramas son conjuntos de octetos que se encuentran en posiciones definidas de la trama donde son transportados, pero no corresponden a interfaces eléctricos reales. Es decir, no hay interfaces eléctricos a velocidades inferiores a 155,520 Mbps en *SDH*. Por esta razón, dichas subtramas se denominan virtuales.

El *VC* (contenedor virtual) es la unidad resultante de completar un *C* con información de gestión de trayecto (*POH*). Un *VC* se puede formar también añadiendo a un *TUG* la correspondiente *POH*.

Actualmente hay definidos dos tipos de *VC*:

1. *VC* de orden inferior: *VC-N* (*N*=l, 2, 3). Está formado por un solo *C-n* (*n*=l, 2, 3) más la *POH* correspondiente.

2. *VC* de orden superior: *VC-N* (*N*=3, 4). Está formado por un solo *C-n* (*n*=3, 4) o por *TUG*'s (*TUG-2* o *TUG3*), junto con la *POH* correspondiente. El *VC* de nivel 3 (*VC-3*) está considerado de orden inferior cuando el camino de multiplexación elegido es el *AU-4*, y de orden superior cuando éste sigue el camino del *AU-3*.

2.2.1.3 Unidades

Se conoce como *U* (unidad) a un grupo de octetos en posiciones fijas y conocidas dentro de una trama o subtrama de *SDH*, destinado al transporte de un *VC*.

Cada *U* posee un puntero que se encuentra en una posición fija en la trama. El puntero indica la distancia en octetos a la que se encuentra el primer octeto del *VC*. En consecuencia, el *VC* puede "flotar" dentro del área de carga que le ha sido asignada.

Unidad tributaria

Una *TU* (unidad tributaria) es una subdivisión del espacio de carga de un *VC* de orden superior y consta de:

- Un contenido útil de información (el *VC* de orden inferior).

- Un puntero de *TU*. Éste se encuentra en una posición fija de la trama o subtrama de orden superior donde reside. Señala el desplazamiento del comienzo de la trama de contenido útil (*VC* inferior), con relación al comienzo de la trama del *VC* de orden superior.

La *TU-n* (*n*=l, 2, 3) consta de un *VC* y de un puntero de *TU*.

Grupo de unidades tributarias

Se denomina *TUG* (grupo de unidades tributarias) a una o más *TU*'s que ocupen posiciones fijas y definidas en un contenido útil de *VC* de orden superior. Una *TUG-2* consta del ensamblado homogéneo de *TU-1*'s idénticas o de una sola *TU-2*. Una *TUG-3* consta de un ensamblado homogéneo de *TUG-2*'s o de una sola *TU-3*.

Unidad administrativa (AU)

Una *AU* (unidad administrativa) es una subdivisión del espacio de carga de la trama de línea. Consta de un contenido útil de información (el *VC* de orden superior) y un puntero de *AU* que señala el desplazamiento del comienzo de la trama de contenido útil, con relación de la trama *STM-N*.

Se han definido dos tipos de *AU*'s, *AU-4* y *AU-3*, aunque solamente la *AU-4* será la que se utilice en la jerarquía europea.

- *AU-4*: consta de un *VC-4* más un puntero de *AU* que indica el alineamiento de fase del *VC-4* con respecto a la trama del *STM-N*.

- *AU-3*: consta de un *VC-3* más un puntero de *AU*.

En cada caso la ubicación del puntero es fija respecto a la trama *STM-N*.

Grupo de unidades administrativas

Se denomina *AUG* (grupo de unidades administrativas) a una o más *AU* que ocupan posiciones fijas y definidas en una trama *STM*.

Una trama *STM-N* contiene un *AUG* de igual orden. Por ejemplo, una trama *STM-1* posee un *AUG* que puede ser de una sola *AU-4* o de tres *AU-3*'s. La configuración con *AU-4* es utilizada por la jerarquía europea (*ETSI*) para el transporte de un *VC-4*. La opción con tres *AU-3*'s es utilizada por la jerarquía norteamericana (*ANSI*), para el transporte de *VC-3*, sin utilizar nunca el *VC-4*.

2.2.1.4 Sección

Una sección se define como aquella parte del trayecto en la que se mantiene la integridad de la señal *STM-N*. Es decir, la multiplexación/demultiplexación se efectúa solamente en los extremos. Es, por lo tanto, un tramo de la red de transmisión delimitado por equipos terminales. En los extremos se extrae o inserta la *SOH*.

2.2.1.5 Trayecto

Es el tramo de la red de transmisión comprendido entre los puntos de ensamblado y desensamblado de los *VC*'s. Puede discurrir a través de más de una sección.

2.2.1.6 Taras o redundancias

Son octetos reservados para la información del propio sistema. Parte de ellos se asignan a los *VC*'s y otros son asignados a la propia señal *STM*. La información contenida en las taras se utiliza para, entre otras, las siguientes funciones:

- Monitorización de la calidad.
- Detección de fallos.
- Gestión de alarmas.
- Canales de comunicación.
- Canales de datos.
- Protección automática.

Según se asigne a los *VC* o a la señal *STM*, se denominan:

- *POH* (tara de trayecto): Se asigna al contenido útil al multiplexarse en el contenedor, permaneciendo con éste hasta que sea demultiplexada la carga útil. Esto define el significado de trayecto en *SDH*.
- *SOH* (tara de sección): Forma parte de la trama *STM*. Una sección (de multiplexación) puede estar formada por varias secciones de regeneración, por lo tanto la *SOH* está dividida en dos partes:
  - *MSOH* (tara de sección de multiplexación).
  - *RSOH* (Tara de sección de regeneración).

En los regeneradores sólo se tiene acceso a la *RSOH*. En la figura 2.13 se representan las partes en que puede dividirse una red de transmisión a efectos de tratamiento de las taras.

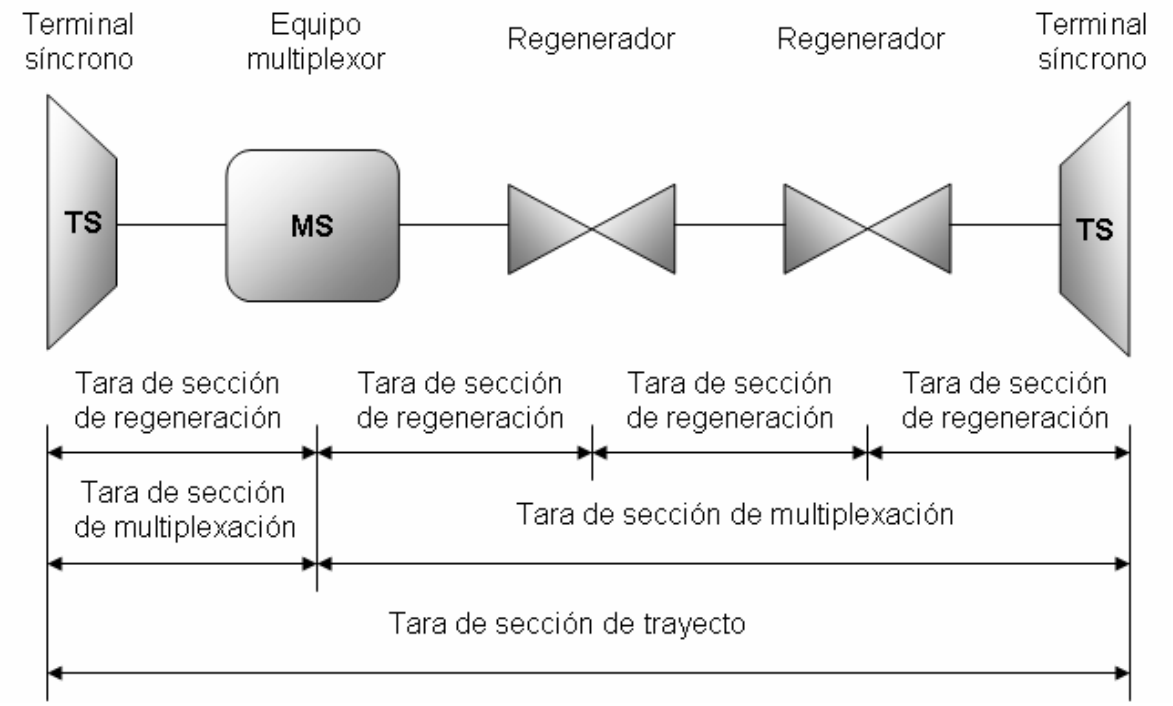


Figura 2.13: Segmentos en que puede dividirse una red

2.2.1.7 Punteros

El puntero es un número binario, situado en posiciones fijas, que va a indicar a qué distancia se encuentra el primer octeto del *VC*.

Los punteros desempeñan dos funciones:

1. Identifican la posición de los *VC*'s en la trama o subtrama correspondiente, que será una *AU* o *TU*. Esto permite asignar de forma flexible y dinámica la carga útil (*VC*) dentro de la trama *AU* o *TU*.

2. Adaptan la velocidad binaria de los *VC*'s a la velocidad binaria del canal de transmisión (*AU* o *TU*). Es decir, mediante un mecanismo de justificación (positiva/nula/negativa) permiten absorber las diferencias de frecuencia entre las diferentes señales que componen un *STM-N*.

2.2.1.8 Mapeado

Es el proceso mediante el cual se adaptan, dentro de los *VC*'s, las señales plesiócronas, las células *ATM* u otras señales. Cada señal se coloca en un *C* de tamaño adecuado y se completa con la *POH*.

2.2.1.9 Alineamiento

Procedimiento basado en la técnica de punteros, mediante el cual se permite identificar las posiciones de los *VC*'s en la trama que los contiene.

2.2.1.10 Multiplexación

Consiste en combinar, por medio del entrelazado de octetos, varias *TU*'s, para obtener un *TUG*, o varios *TUG*'s para formar otro *TUG* o un *VC* de orden superior. Es un proceso completamente síncrono.

2.2.1.11 Concatenación

La *SDH*, con el fin de prestar soporte a servicios de ancho de banda intermedios, ha previsto la posibilidad de concatenar varias *AU*'s o *TU*'s dentro de un *STM-N*, es decir, combinar *N AU*'s o *TU*'s, para formar un *C* de tamaño *N* veces el tamaño del *C* englobado

por una *AU* o *TU*. Por ejemplo, las *AU-4* pueden concatenarse para formar una *AU-4-Nc* que puede transportar contenidos útiles que requieren una capacidad superior a un *C-4*. La capacidad disponible para el establecimiento de la correspondencia es *N* veces la capacidad del *C-4*, el contenido útil de múltiples *C-4*'s transportados en un solo *VC-4-Nc*.

2.2.2 Comparación entre *SONET* y *SDH*

SONET		***SDH***	
Señal	**Velocidad binaria (Mbps)**	**Señal**	**Velocidad binaria (Mbps)**
STS-1[*]	51,840		
STS-3	155,520	STM-1[**]	155,520
STS-9	466,560		
STS-12	622,080	STM-4	622,080
STS-18	933,120		
STS-24	1244,160		
STS-36	1866,240		
STS-48	2488,320	STM-16	2488,320
STS-192	9953,280	STM-64	9953,280

[*] Señal primaria para la *SONET* [**] Señal primaria para la *SDH*

Tabla 2.4: Velocidades binarias de las redes *SONET* y *SDH*

Ambas redes son muy parecidas, utilizan la misma estructura de trama, ponen en práctica la misma filosofía de multiplexado, etc. No obstante, la diferencia estriba en que sus velocidades nominales no son iguales. Esto se debe a que ambas redes pertenecen a jerarquías diferentes. La *SONET* se utiliza ampliamente en EEUU mientras que la *SDH* se ha difundido por Europa. De todas formas, la adaptación S*ONET / SDH* y *SDH / SONET* se puede realizar sin problemas.

A pesar de que la *SONET* ha sido la precursora de la *SDH*, ha quedado englobada dentro del conjunto de normas que forman la *SDH* como una particularización de la misma y en corto plazo, se augura la desaparición de la *SONET* y la consolidación de la *SDH* como una red de ámbito mundial.

2.2.2.1 Tipos de señales

En *SONET* existen dos tipos de señales:

- *STS-N*: Señal de transporte síncrono de nivel *N*.
- *OC-N*: Portadora óptica de nivel *N*.

Conceptualmente se trata de la misma señal, solo que cuando es eléctrica se llama *STS* y cuando es óptica *OC*.

En *SDH*, en cambio, sólo existe un nombre para designar a la señal eléctrica y a la óptica, *STM* o módulo de transporte síncrono.

2.2.2.2 Equivalencias

Como ya se ha mencionado con anterioridad, ambas redes (*SONET* y *SDH*) utilizan las mismas técnicas de multiplexado, así como también la misma estructura de trama, por lo tanto su adaptación es sencilla.

Si se observa la tabla 2.4 se pueden ver aquellas señales que son equivalentes en cuanto a tamaño de trama y por lo tanto, a velocidad. Conviene resaltar que puede pasarse de un nivel a otro a través del entrelazado a nivel de octeto de un cierto número de señales primarias. Como puede verse, para formar una *STS-3* es necesario multiplexar 3 señales *STS-1*'s. Para obtener una *STM-16* habrá que multiplexar 16 *STM-1*'s. Nótese que la velocidad de las señales primarias de ambas redes no coinciden entre sí y que la velocidad de señal primaria en *SDH* es equivalente a la de la señal de nivel 3 en *SONET*.
Sin más, puede decirse que:

Velocidad de STM-N = N x (Velocidad de STS - 3)

2.2.3 Estructura de la trama

2.2.3.1 Estructura de una *STM-1*

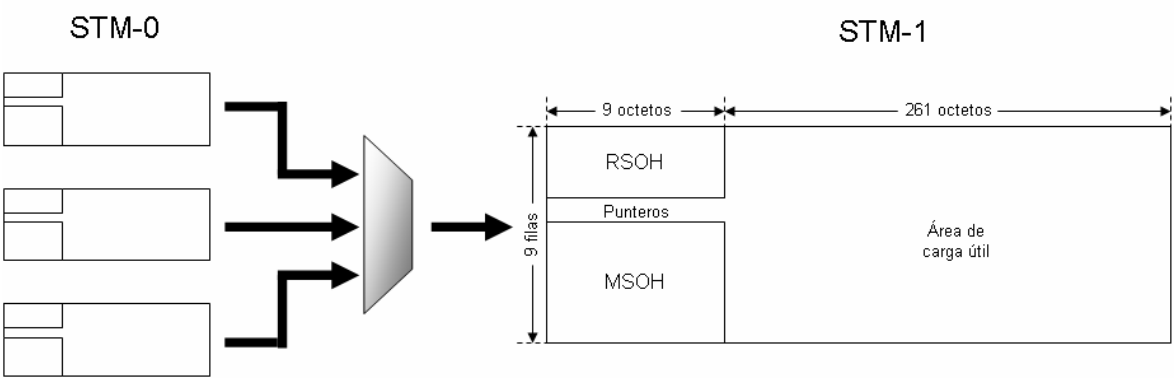


Figura 2.14: Estructura de la trama básica *STM-1*

Como ya se ha mencionado la *STM-1* es la señal base de la *SDH*. Conceptualmente hablando, puede decirse que una *STM-1* está formada por el multiplexado a nivel de octeto de tres señales *STM-0* (señales imaginarias) cuya velocidad es equivalente a la de una trama *STS-1*.

El multiplexado se realiza de tal forma que el primer byte de la trama *STM-1* es el primer byte de la primera trama *STM-0*, el segundo byte de la trama *STM-1* es el primer byte de la segunda trama *STM-0* y el tercer byte de la trama *STM-1* es el primer byte de la tercera trama *STM-0*, y así sucesivamente hasta obtener el multiplexado completo de las tres *STM-0*. El resultado final es el que se muestra en la parte inferior de la figura 2.14, una trama básica *STM-1* tres veces más rápida que una trama *STS-1* (trama básica en *SONET*).

La *STM-1*, o módulo de transporte síncrono de nivel 1, está constituida por 2430 octetos y posee un tiempo de duración de 125 μs. Tanto en *SONET* como en *SDH*, el tiempo de duración de trama se mantiene para los diferentes niveles cambiando el número de octetos contenidos en cada trama. A mayor nivel, mayor número de octetos.

Cada octeto representa un canal con una capacidad de transporte de 64 Kbps. Si se requiere un canal con una capacidad *N*x64 Kbps, entonces se tomarán *N* octetos dentro de la misma trama.

El primer octeto transmitido es el que ocupa la esquina superior izquierda, el sentido de la transmisión continúa barriendo las filas de izquierda a derecha, de tal forma que el último octeto en ser transmitido sea el que ocupa la esquina inferior derecha, a su vez, cada octeto será transmitido desde su bit más significativo hasta el menos significativo.

En toda trama *STM-1* se diferencian, como puede verse en la figura 2.14, cuatro regiones:

1. *RSOH* (tara de sección de regeneración): Está constituida por los primeros 9 octetos de las filas 1 a 3. Su contenido es examinado y puede ser modificado, no sólo por las estaciones terminales de una Sección de Multiplexación, sino también por los regeneradores de línea. Contiene, entre otras cosas:

    - Señal de alineamiento de trama.
    - Etiquetas.
    - Información de gestión.
    - Supervisión de errores de la señal de línea (sección de regeneración).
    - Canales de servicio (analógicos y digitales).

2. Región del puntero *AU-4*: Forma parte del área de carga y está formada por los 9 primeros octetos de la cuarta fila. En *SONET*, este espacio es ocupado por 3 punteros *AU-3*'s. Recuérdese que en *SONET* ésta sería una trama *STS-3*, fruto del entrelazado de tres *STS-1*'s. En cualquier caso, el puntero ocupa un lugar fijo en la trama. Una vez se

haya recuperado e interpretado el puntero, se sabrá donde comienza la carga útil (*VC-4*). El puntero de *AU-4* identifica el comienzo de un *VC-4*.

3. *MSOH* (tara de sección de multiplexación): Está compuesta por los 9 primeros octetos de las filas 5 a 9 y sólo pueden ser accedidos en los nodos de red terminales de la sección de múltiplexación. La *MSOH* la forman, entre otras cosas:

   - Supervisión de errores de la sección de multiplexación.
   - Canales de control para la conmutación de protección.
   - Canales de servicio.

4. Área de carga útil: Está formada por los octetos del 10 al 2430, de la filas 1 a la 9. También se incluye como carga útil la región del puntero *AU-4*.

2.2.3.2 Construcción de una *STM-1*

En el área de carga útil mostrada en la figura 2.14, irá situado el *VC-4*. No obstante, el *VC-4* podrá ir situado a partir de cualquier posición dentro de dicho área siendo el puntero de *AU-4* el que indique esta posición.

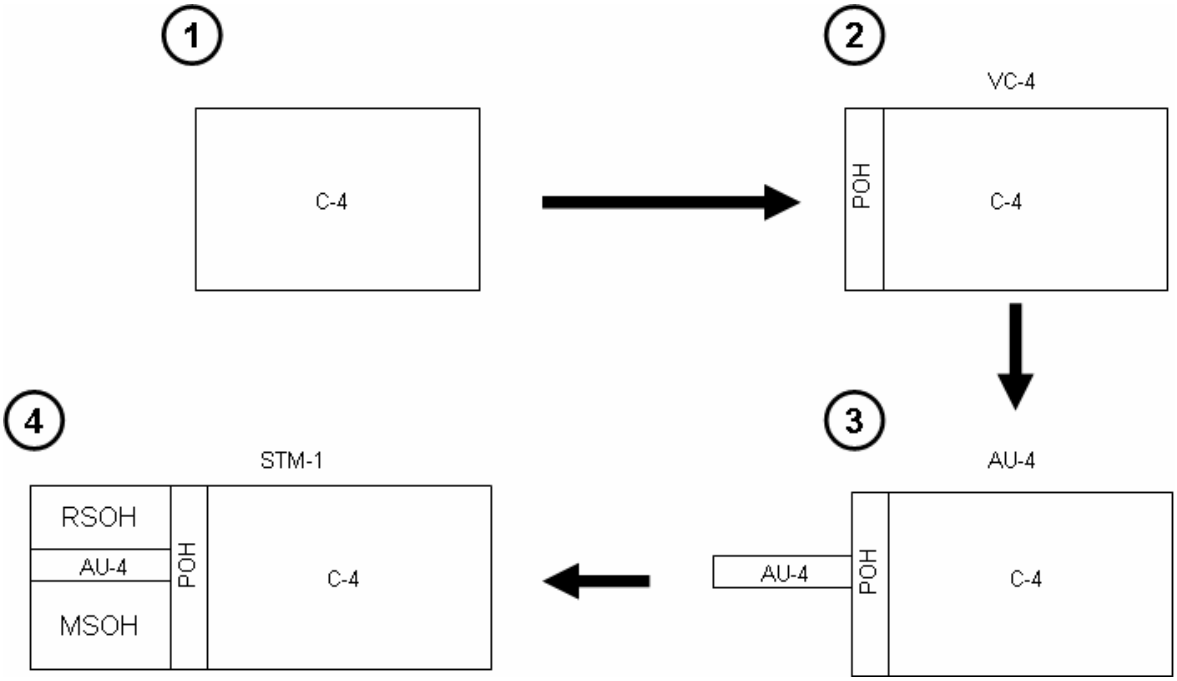


Figura 2.15: Creación de una trama *STM-1*

La figura 2.15 refleja los pasos a seguir para la creación de una trama *STM-1*, partiendo de la figura superior izquierda y siguiendo el sentido de las flechas:

1. Se toma un contenedor *C-4*, el cual posee datos.

2. Se le asigna una tara de trayecto, *POH* (se explicará más adelante), convirtiéndolo en un *VC-4*.

3. Se comprueba la posición que va a ocupar dentro del área útil de la trama *STM-1* y se le asigna un puntero. El conjunto se llama ahora *AU-4*.

4. Finalmente se le añade la *SOH* (se explicará más adelante).

El contenedor *C-4*, como ya se ha dicho anteriormente, puede contener señales, o lo que es lo mismo, tributarios de diferentes velocidades y de diferente naturaleza, como *ATM*,

señales plesiócronas, etc., intercalados a nivel de octeto en la trama, en subdivisiones consecutivas del área de carga, de forma muy ordenada. En la figura 2.16 se representa esta subdivisión, según el esquema general definido por la *UIT-T*. En este proyecto se contempla el tratamiento de *VC*'s de orden superior (resaltado en la figura 2.16), por lo tanto, en base a lo dicho, se omitirá la explicación del multiplexado de los *C*'s de más bajo nivel.

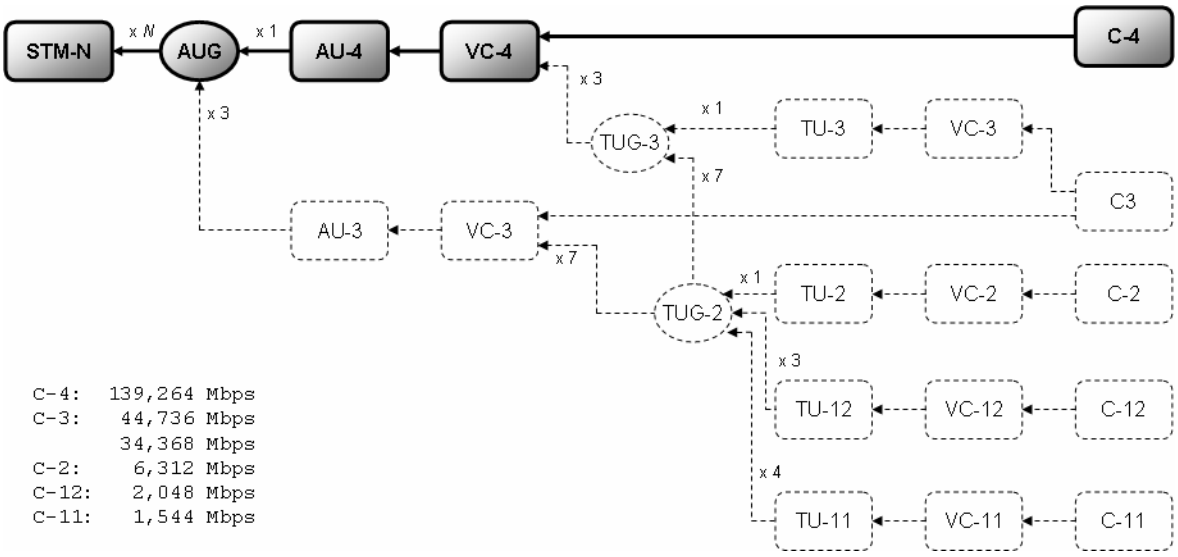


Figura 2.16: Estructura de multiplexación de la *UIT-T*

2.2.4 Taras de sección y de trayecto

Las taras (en inglés *overhead*, *OH*) son las encargadas de transportar la información necesaria para desempeñar las labores de operación, administración y mantenimiento. Se puede decir que son las responsables de transmitir la información necesaria para que la capa física de la red pueda realizar su función. Dichas taras no son más que conjuntos de octetos situados en unas posiciones fijas dentro de la trama. Existen dos tipos de taras:

- *SOH* (tara de sección)
- *POH* (tara de trayecto)

1. *SOH*. Permite realizar, entre otras, las funciones de:

    - Alineación de trama.
    - Monitorización de la calidad.
    - Detección de fallos.
    - Gestión de alarmas.
    - Canales de comunicación.
    - Canales de datos.

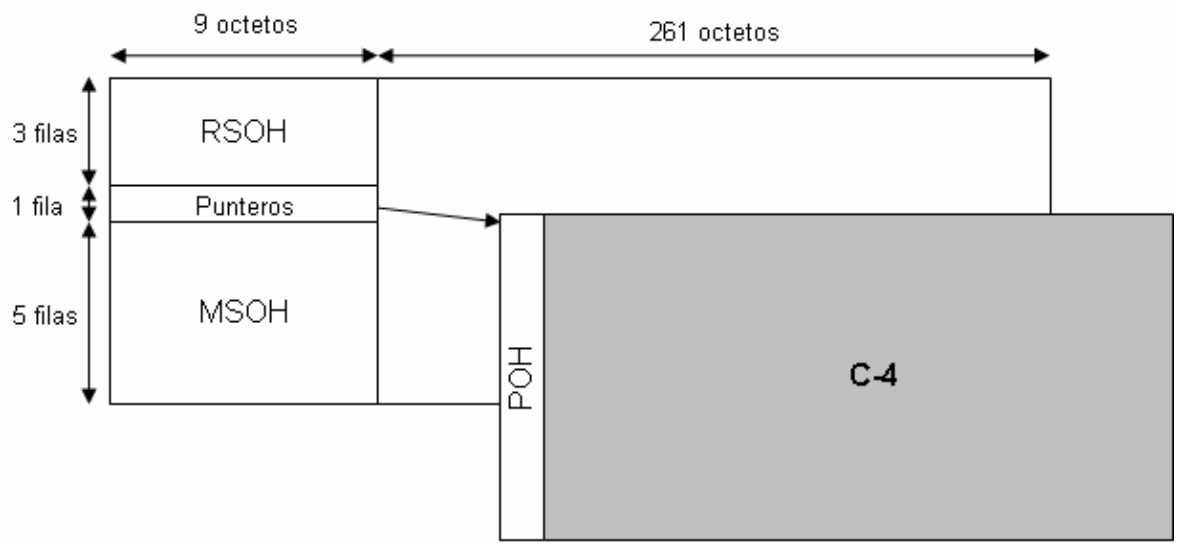


Figura 2.17: Ubicación de la *SOH* y *POH* en la trama *STM-1*

En la figura 2.17 pueden diferenciarse dos categorías de Tara de sección:

(a) *RSOH* (tara de sección de regeneración): Se procesa en los regeneradores y en los equipos terminales y puede extraerse o añadirse y modificarse, sin necesidad de desensamblar el *STM*.

(b) *MSOH* (tara de sección de multiplexación): Pasa de forma transparente a través de los regeneradores y se procesa en los equipos terminales, en los puntos donde se ensamblan y desensamblan las *AU*'s.

2. *POH* (tara de trayecto). Forma parte de los *VC*'s, permitiendo la integridad y gestión de la información, entre el punto de ensamblado de un *VC* y el punto de desensamblado. Se utiliza básicamente para:

   - Monitorización de la calidad de trayecto.
   - Mantenimiento.
   - Detección de fallos.
   - Canales de órdenes.
   - Canales de comunicación.

En la figura 2.13 se representan los segmentos en los que se divide una red y a qué segmento corresponde cada una de las taras existentes.

2.2.4.1 Tara de sección

La *SOH* está ubicada en las primeras 9 columnas de cada trama. Las tres primeras filas corresponden a la *RSOH* y las cinco últimas a la *MSOH*. Como ya se verá más adelante, la primera fila de la *RSOH* es la única en toda la trama que no es aleatorizada.

RSOH

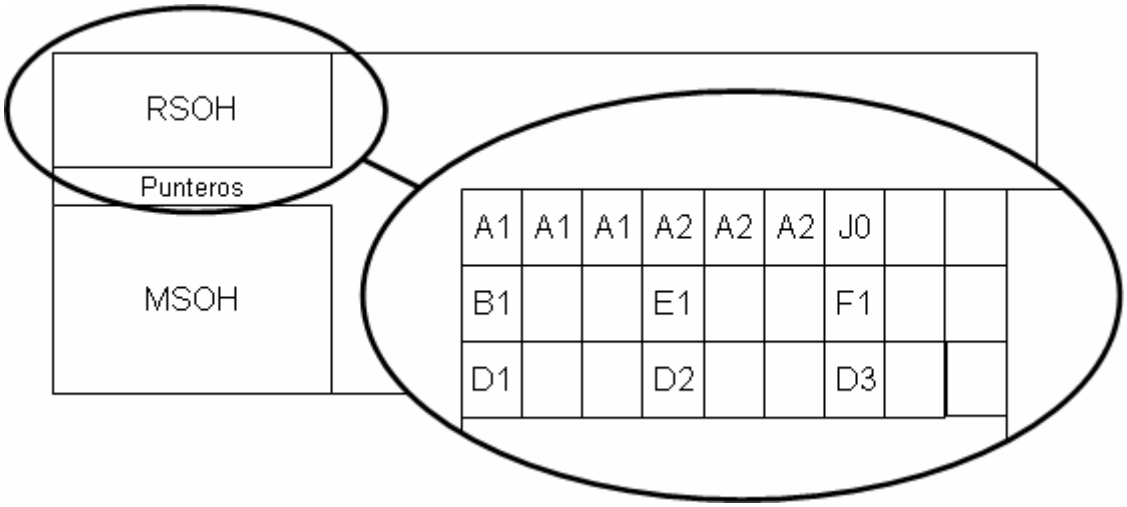


Figura 2.18: Composición de la *RSOH*

Todos los equipos de línea, ya sean terminales de una sección o regeneradores intermedios, tienen acceso a la *RSOH*. En la figura 2.18 se muestran los octetos que la forman y que se describen a continuación. Cada uno de estos octetos ocupa una posición fija en la trama.

A1, A2: Señal de alineamiento de trama

Indican el comienzo de una trama, siendo sus valores fijos:

Al: 11110110b=F6h
A2: 00101000b=28h

Estos octetos son analizados en el inicio de cada trama para verificar la alineación correcta. Cada *STM-1*, dentro de una señal *STM-N*, mantiene su propia señal de alineamiento. No son aleatorizados.

JO: Identificador de Sección de Regeneración

Este octeto se utiliza para transmitir de forma repetitiva un "Identificador de Punto de Acceso de Sección" , de tal modo que el receptor pueda verificar la continuidad de su conexión con el transmisor pretendido. Permite validar el correcto establecimiento de la sección y localizar los errores de la configuración de la red.

B1: Control de errores

Se utiliza para la monitorización de errores en una sección de regeneración. El método empleado se denomina paridad con entrelazado de 8 bits (*BIP-8*).

El *BIP-8* se calcula en base a todos los bits de la trama *STM-N* anterior y el resultado del cálculo se almacena en el octeto *B1*. En el extremo receptor se recalcula el *BIP-8* y se compara el valor calculado con el octeto *B1* recibido. Si hay diferencia entre ambos es que se ha producido algún error durante la transmisión.

Sea cual fuere el orden *N* de una trama *STM-N*, sólo hay un único octeto *B1*, siempre en la posición correspondiente al *B1* del primer tributario síncrono de dicha señal. Todos los octetos de cada trama (después de ser aleatorizada) pasan por un registro que, al final de la misma, calcula un código de 8 bits, que es el que se introduce en el octeto *B1*.

El código de paridad con entrelazado de 8 bits (*BIP-8*) se define como un método de supervisión de errores con paridad par (Ecuación 3.1). El equipo transmisor genera un código de 8 bits en una parte específica de la señal, de manera que el primer bit del código proporciona paridad par en los primeros bits de todos los octetos, el segundo bit del código proporciona paridad par en los segundos bits de todos los octetos, etc.

$$\sum_{1}^{270\times 9\times N} 1^{er}\,bit \quad \sum_{1}^{270\times 9\times N} 2^{o}\,bit \quad \sum_{1}^{270\times 9\times N} 3^{er}\,bit \quad \sum_{1}^{270\times 9\times N} 4^{o}\,bit \quad \sum_{1}^{270\times 9\times N} 5^{o}\,bit \quad \sum_{1}^{270\times 9\times N} 6^{o}\,bit \quad \sum_{1}^{270\times 9\times N} 7^{o}\,bit \quad \sum_{1}^{270\times 9\times N} 8^{o}\,bit \qquad (3.1)$$

El octeto *B1* es calculado y analizado por todos los elementos de red, incluidos los regeneradores intermedios.

E1: Circuito de órdenes

Este octeto proporciona un canal a 64 Kbps entre terminales y/o regeneradores, se puede utilizar como canal de servicio para comunicaciones de voz por parte del personal de mantenimiento.

F1: Canal de usuario

Es similar al octeto *E1*, constituye un canal de usuario para aplicaciones especiales tales como la realización de conexiones temporales de voz o datos para fines de mantenimiento.

D1 a D3: Canal de comunicación de datos

Deben ser considerados como un único canal de 192 Kbps (64 Kbps x 3) para alarmas, mantenimiento, control, monitorización, administración y otras necesidades de comunicación entre equipos terminales de sección.

Este canal está disponible para mensajes específicos generados internamente, externamente y del fabricante.

Es usado como canal de conmutación empotrado (*ECC*) para *SDH* y como canal de operaciones (*EOC*).

Los octetos de alineamiento de trama se repiten de tres en tres para mantener la compatibilidad con la jerarquía *SONET* americana. La trama básica *SONET* es de 51,840 Mbps, que corresponde a la tercera parte de la trama *STM-1*. Esta es la razón por la cual hay tantos octetos vacíos en la redundancia de la señal *STM-1*.

MSOH

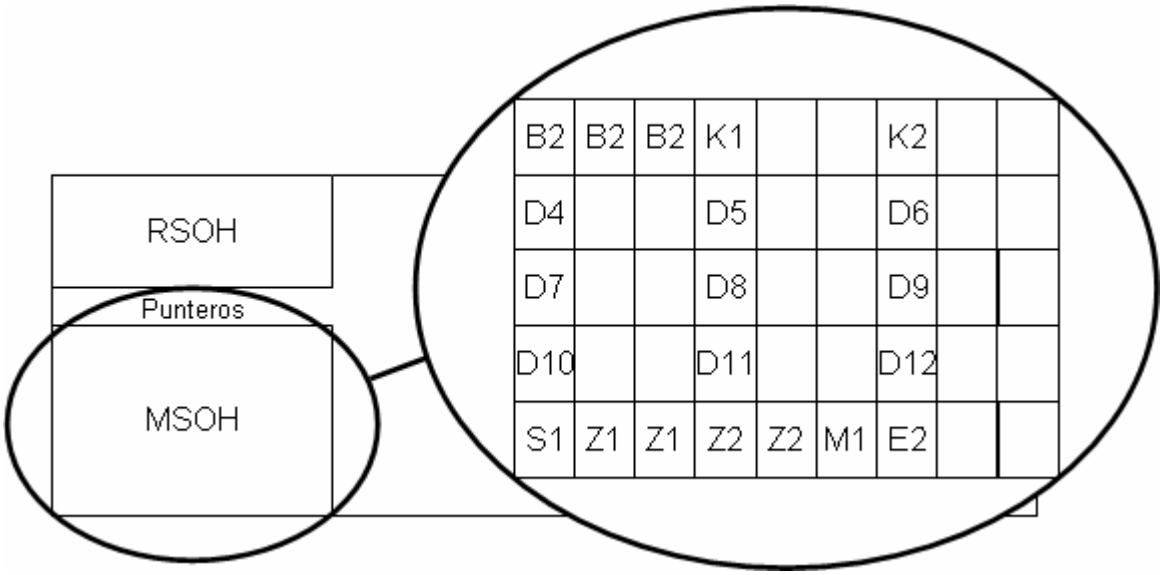


Figura 2.19: Composición de la *MSOH*

Tendrán derecho a acceder a los octetos que forman parte de la *MSOH* sólo aquellos equipos que sean terminales de sección. En la figura 2.19 se muestran los octetos que forman la *MSOH* y que se describen a continuación. Cada uno de estos octetos ocupa una posición fija en la trama.

B2: Control de errores

Existen diferencias con el cálculo de *B1*. En primer lugar, existen tres *B2* en cada *STM-1* y se utilizan para monitorización de errores dentro de una sección de multiplexación por el método de paridad con entrelazado de 24 bits (*BIP-24*).En segundo lugar, como el contenido de las tres primeras filas de la tara de sección correspondiente a la *RSOH*, pueden ser alteradas por los regeneradores intermedios y *B2* sólo es analizado en los extremos de una sección, la porción de señal que cubre el cálculo de *B2*, comprende toda el área de carga más las filas 4 a 9 de la redundancia, es decir, toda la señal *STM-1* (antes de ser aleatorizada) excepto la *RSOH*.

Debido a que para cada *STM-1* existen 3 octetos *B2*, en una señal *STM-N* el código que se utiliza para su cálculo es el *BIP-24* x *N*, siendo *N* el nivel de la señal *STM*. Es decir, se dividirá todo el conjunto de octetos sobre los que actúa el *B2* en grupos de *3xN* octetos y se realizarán las mismas operaciones que las explicadas para el octeto *B1*, solo que la palabra pasa de ser de 8 a 3x8x*N* bits.

Analizando la información que contienen los octetos *B1* y *B2*, se pueden identificar fallos, es decir, determinar en qué tramo en concreto de la red se están produciendo errores.

Kl, K2: Señalización para la conmutación de protección automática (APS)

Estos octetos se utilizan para el control de la protección por conmutación automática (*APS*) de una sección de multiplexación. Mediante el envío de mensajes, con protocolos orientados a bit, por los canales que proporcionan estos octetos y el establecimiento del correspondiente algoritmo de protección, es posible efectuar la conmutación de una señal desde un canal de servicio que falla a un canal de protección y su posterior restablecimiento.

Los octetos *K1* y *K2* se emplean en el control de protección en redes con diferentes estructuras. En cualquier caso, los requerimientos para ejecutar la conmutación de protección pueden ser iniciados externa o automáticamente.

Los requerimientos externos son iniciados en un elemento de red por el Sistema de Operación y pueden ser transmitidos al elemento de red apropiado mediante comandos que usan octetos *APS* (*K1*, *K2*) o mediante comandos que no son señalizados por éstos, y que pueden ser transmitidos sobre los *DCC*'s (canales de comunicación de datos de la *MSOH*).

Los comandos iniciados automáticamente requieren los octetos *APS* y están basados en condiciones de fallo de las secciones de multiplexación en servicio y de protección, detectadas por el elemento de red mediante la monitorización continuada de las mismas. Estas condiciones de fallo pueden ser:

- Fallo de señal (*SF*).

- Degradación de señal (*SD*).

La asignación de los octetos *K1* y *K2* puede diferir, en función de la arquitectura de la red a la que se aplique.

El octeto *K1* indica una petición de un canal para una acción de conmutación.

El octeto *K2* indica el estado del puente en el conmutador de la función de protección de la sección de multiplexación en los bits 4 a 7, aunque en el caso de estructura de anillo, pueden estar asignados al identificador del nodo de origen. El bit 3 indica el tipo de la arquitectura *MSP*: a nivel alto, indica arquitectura l:n, y a nivel bajo indica arquitectura 1+1.

Los bits 2 a 0 del octeto *K2* se reservan para uso futuro. No obstante, los códigos *111b* y *110b* son usados en todas las señales de línea *STM-N*, para transmitir señales de mantenimiento de la sección de multiplexación:

111b: *MS-AIS* Señal de indicación de alarma de sección de multiplexación.

110b: *MS-RDI* Indicación de defecto remoto de la sección de multiplexación.

La *MS-AIS* es una señal que se envía hacia el destino como indicación de que se ha detectado un fallo hacia el origen y se ha enviado una alarma.

El *MS-RDI* se utiliza para devolver una indicación al extremo transmisor de que el receptor ha detectado un fallo de sección entrante, o se está recibiendo una *MS-AIS*.

D4 a D12: Canal de comunicación de datos (DCC)

Debe considerarse como un canal de 576 Kbps (9x64 Kbps) para alarmas, mantenimiento, control, monitorización, administración y otras necesidades de comunicación entre entidades de línea.

Z1, Z2: Octetos sin especificar

Estos octetos están reservados para uso futuro.

S1: Estado de sincronización (Marcador de sincronismo)

El octeto *S1* se define como un canal que describe la calidad y el estado de sincronización de un nodo que genera una señal *STM-N*.

Los mensajes contenidos en el octeto *S1*, permiten a los equipos determinar el nivel de calidad de la fuente de temporización que reciben, e informar de la que envían.

M1: Indicador de errores remotos (MS-REI)

Este octeto se utiliza como comunicador de errores remotos de la sección de multiplexación.

Cuando el receptor realiza el cálculo de *B2* de una trama *STM-N* (*n*), espera a la llegada del *B2* de la trama *STM-N* (*n+1*) y entonces compara ambos. En caso de error, se contabilizan las diferencias encontradas entre ambas palabras y se envía, mediante el octeto *M1*, al extremo distante (llamado TERMINAL "A" en la figura 2.20) de la sección de multiplexación.

Por lo tanto se puede decir que en una señal *STM-N*, este octeto lleva la cuenta (en la gama de 0 a 255) de los bloques de bits entrelazados que han sido detectados como erróneos por el *BIP 24xN* (*B2*). Para velocidades de *STM-16* y superiores este valor será truncado a 255.

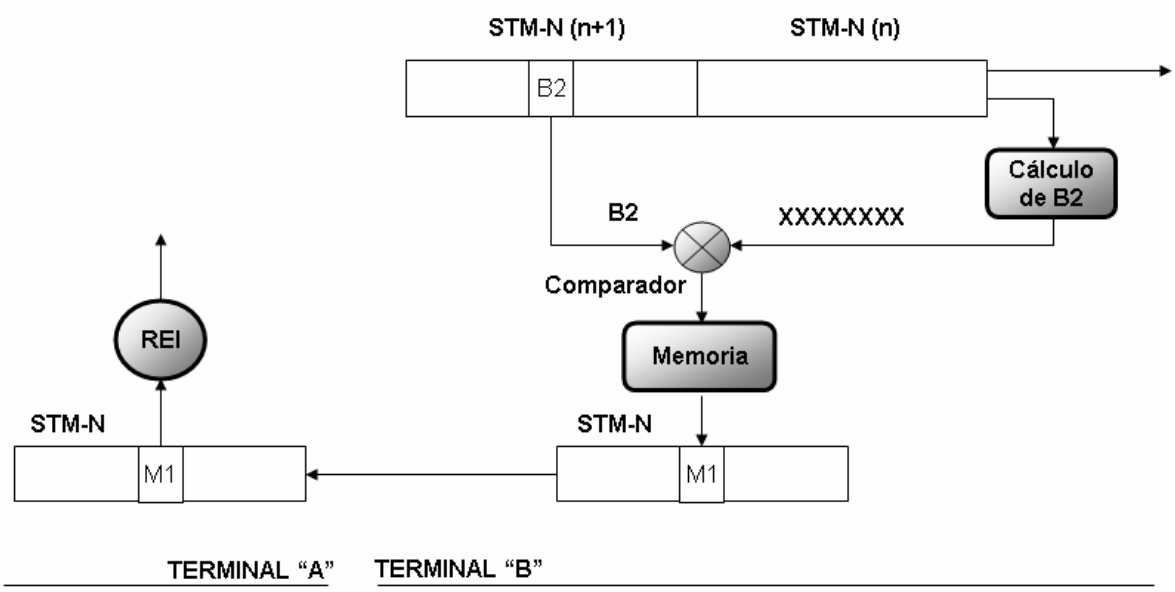


Figura 2.20: Esquema para la indicación de errores remotos (*RDI*)

El valor del octeto se interpretará de la siguiente manera:

Código M1[7:0]	Interpretación del código
00000000	Ninguna violación de *BIP*
00000001	1 violación de *BIP*
00000010	2 violaciones de *BIP*
. . .	. . .
11111111	255 o más violaciones de *BIP*

Tabla 2.5: Codificación de *M1*

E2: Circuito de órdenes

Este octeto es similar al octeto *E1* de la *RSOH*, proporcionando un canal de comunicación de voz, pero accesible solamente entre equipos multiplexores.

Interfaz con funcionalidades de *SOH* reducidas

En algunas aplicaciones, puede utilizarse una interfaz con funcionalidades de *SOH* reducidas. En la tabla 2.6 se muestran los octetos que deben utilizase en dicha interfaz.

Para la interpretación de dicha tabla se debe tener en cuenta que la condición "Requerido" indica que la señalización correspondiente contendrá la información tal y como se ha definido en el apartado 2.2.4.1. La condición "Opcional" permite que la señal correspondiente pueda tener o no información válida, por lo que la utilización de estas señales será una decisión local. Por último, la condición de "No Aplicable" se refiere a que esta función no está definida en la interfaz.

Octetos de *SOH*	Interfaz óptica	Interfaz eléctrica
A1, *A2*	Requerida	Requerida
J0-Z0/*C1*	Nota 1	Nota 1
B1	No aplicable	No aplicable
E1	Opcional	Opcional
F1	No aplicable	No aplicable
D1-D3	No aplicable	No aplicable
B2	Requerida	Requerida
K1, *K2* (*APS*)	Opcional	No aplicable
K2 (*MS-AIS*)	Nota 2	Nota 2
K2 (*MS-RDI*)	Requerida	Requerida
D4-D12	No aplicable	No aplicable
S1	Nota 2	Nota 2
M1	Nota 2	Nota 2
E2	No aplicable	No aplicable
Otros octetos	No aplicable	No aplicable

Notas:

1. Para equipos que implementan la función del identificador de *STM* (*C1*), el uso de este octeto es opcional para *STM-1* y requerido para *STM-4/16*. Para equipos que implementan la traza de sección de regeneración (*J1*) el uso de este octeto queda en estudio.
2. Queda en estudio.

Tabla 2.6: Interfaz con funcionalidad de *SOH* reducida

2.2.4.2 Tara de trayecto

El *VC-4* presenta una estructura de 9 filas por trama de 125 μs, si bien la longitud de la fila es diferente en cada caso, 260 x *N* octetos, donde *N* representa el nivel de la señal *STM-N*.

En cualquier caso, la *POH* del *VC* de orden superior ocupa 9 filas de la primera columna, es decir, consta de 9 octetos.

Esta misma *POH* es aplicable, en toda su extensión, al *VC* concatenado *VC-4-Nc*, que se verá más adelante.

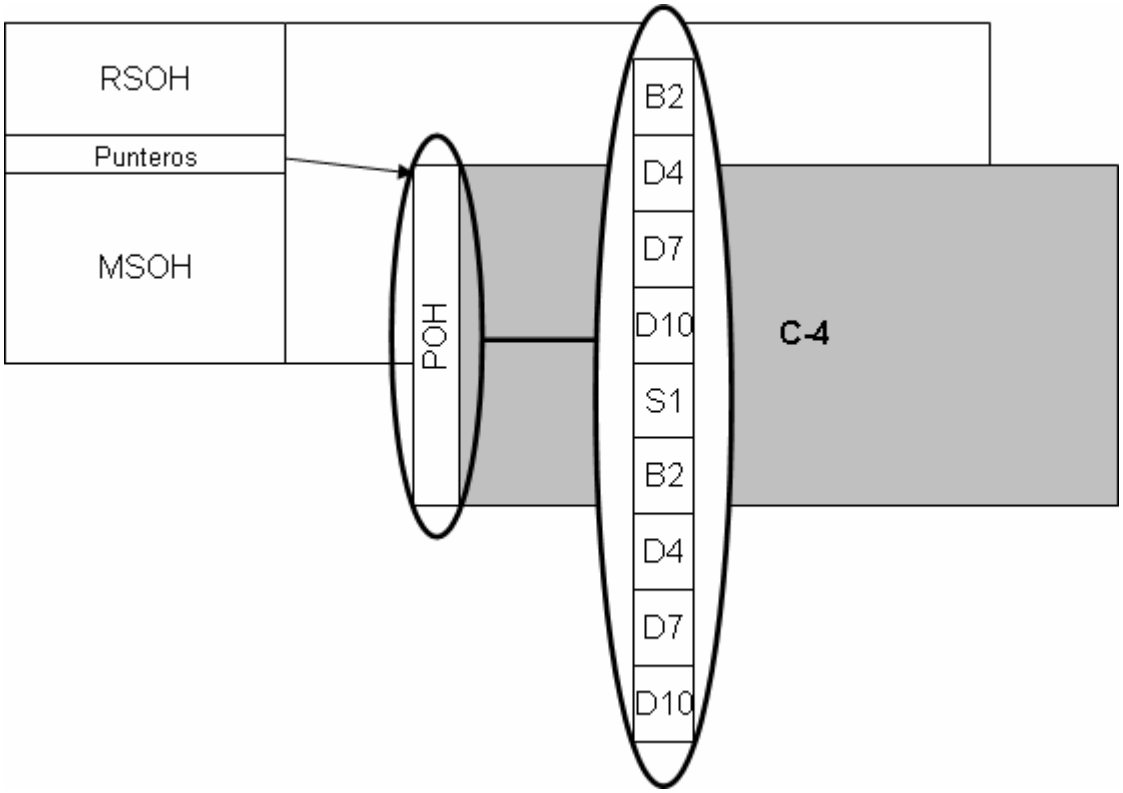


Figura 2.21: *POH* en una *STM-1* para un *VC-4*

Los 9 octetos de la *POH* especificados actualmente, están clasificados como sigue:

- Octetos o bits usados para la comunicación extremo a extremo, con independencia de la función de la carga útil: *J1*, *B3*, *C2*, *G1*, *K3* (b_7 a b_4).

- Octetos específicos de la carga útil: *H4*, *F2*, *F3*.

- Octetos para futura normalización internacional: *K3* (b_5 a b_0).

- Octetos que pueden sobreescribirse dentro del dominio del operador: *N1*.

J1: Identificador de trayecto (traza de trayecto)

Éste es el primer octeto de la trama del *VC*, su ubicación está señalada por medio del puntero asociado (de *AU-4*, en este caso) y es utilizado para transmitir de una forma repetitiva un identificador de punto de acceso de trayecto de orden superior, de tal forma que el receptor pueda verificar su continua conexión al transmisor pretendido.

Dentro de una red nacional o dentro del dominio de un solo operador, el identificador de punto de acceso de trayecto puede utilizar una cadena de formato libre de 64 octetos o el formato de identificador de punto de acceso definido en la Recomendación G.831 de la *UIT-T*. En las fronteras internacionales, o en las fronteras entre las redes de diferentes operadores, se utilizará el formato definido en dicha Recomendación, a menos que los operadores que proporcionan el transporte convengan otra cosa mutuamente.

La Recomendación *G.831*, define una trama de 16 octetos para la transmisión del identificador del punto de acceso de trayecto, de forma idéntica a la trama de 16 octetos descrita en el apartado 2.2.4.1, para el octeto *J0* de la tara de sección de regeneración.

B3: Control de errores de trayecto

Se asigna un octeto en cada *VC-4-Nc*/*VC-4*/*VC-3* para una función de supervisión de errores de trayecto. Esta función es un código de *BIP-8* que utiliza paridad par. El *BIP-*

8 de trayecto se calcula en base a todos los bits del *VC-4-Nc*/*VC-4*/*VC-3* anterior antes de la aleatorización.

En el extremo receptor, se calcula el *BIP-8* sobre la misma trama y el resultado se compara con el octeto *B3* recibido en la trama siguiente. Si hay diferencia entre el valor calculado y el valor recibido, es que se ha producido algún error en el trayecto.

C2: Etiqueta de la señal

Indica la composición del *VC*, la correspondencia con redes *MAN*, *FDDI*, *ATM*, .. . y comprueba si se han equipado. Además existen códigos de reserva que se dejan para uso futuro.

El valor *0* en el octeto *C2* indica trayecto de *VC-4-Nc*/*VC-4*/*VC-3* no equipado. Este valor se origina si la sección está completa, pero no hay equipo que origine el trayecto *VC-4-Nc*/*VC-4*/*VC-3*. Significa que no hay una conexión configurada.

El valor *1* solamente se utiliza en los casos en que un código de entramado o mapeado no está definido en la tabla de codificación que utiliza el equipo. Para la interoperabilidad con un equipo antiguo se aplican las siguientes condiciones:

- Para compatibilidad hacia atrás, el equipo antiguo interpretará cualquier valor recibido, distinto del valor *0*, como una condición de equipado.

- Para compatibilidad hacia delante, cuando reciba el valor 1 del equipo antiguo, el nuevo equipo no generará una alarma de desadaptación de etiqueta de señal.

G1: Estado de trayecto

Este octeto se utiliza para devolver al origen del trayecto de *VC-4 / VC-3* el estado de la categoría y funcionamiento de la terminación de trayecto. Esta característica permite el control dúplex del trayecto que es monitorizado, en ambos extremos, o en cualquier punto a lo largo del trayecto.

Los bits 7 a 4 del octeto *G1* se utilizan para devolver la indicación de errores remotos (*REI*). Llevan la cuenta de los bloques de bits entrelazados que han sido detectados con error por la aplicación del código *BIP-8* del trayecto (*B3*).

Los bits 3, 2 y 1 proporcionan un código para devolver una indicación de defecto remoto (*RDI*). Esta indicación permite la diferenciación de defectos entre la carga útil, conectividad y servicio. Facilita, además, la compatibilidad hacia atrás con versiones anteriores en las que sólo se asignaba el bit 3 para dicha indicación. La recepción de los bits 2 y 1 como (0, 0) o (1, 1), implica que solamente el bit 3 será interpretado como *RDI*. Si los bits 2 y 1 se reciben como (0, 1) o (1, 0), entonces los bits 3 y 2 serán interpretados como *RDI*.

El bit 0 queda reservado para uso futuro. No tiene ningún valor por lo que el receptor debe ignorar su contenido.

F2: Canal de usuario de trayecto

Canal a 64 Kbps a disposición del usuario para la comunicación entre elementos de trayecto. Similar al octeto *F1* de la *SOH* visto anteriormente.

H4: Identificador de posición

Este octeto proporciona un indicador de posición generalizado para cargas útiles y puede ser específico de la carga útil.

F3: Canal de usuario

Este canal es idéntico al *F2*.

K3: Señalización para conmutación automática (APS)

Este octeto es similar a los octetos *K1* y *K2* de la tara de sección. Se usa para el control de la protección por conmutación automática (*APS*), pero a nivel de trayecto, en lugar de a nivel de sección.

De momento sólo se utilizan para este fin los bits 7, 6, 5 y 4 de este octeto. Los bits 3 a 0 se dejan en reserva para uso futuro. No tiene actualmente ningún valor, por lo que el receptor debe ignorar su contenido.

N1: Octeto de operador de red

Este octeto se asigna para proporcionar una función de supervisión de conexión en cascada (*TCM*). En los Anexos C y D de la Recomendación G.707 se dan los detalles relativos a dos posibles implementaciones de la función *HO-TCM*.

2.2.5 Punteros, fuente de alineamiento

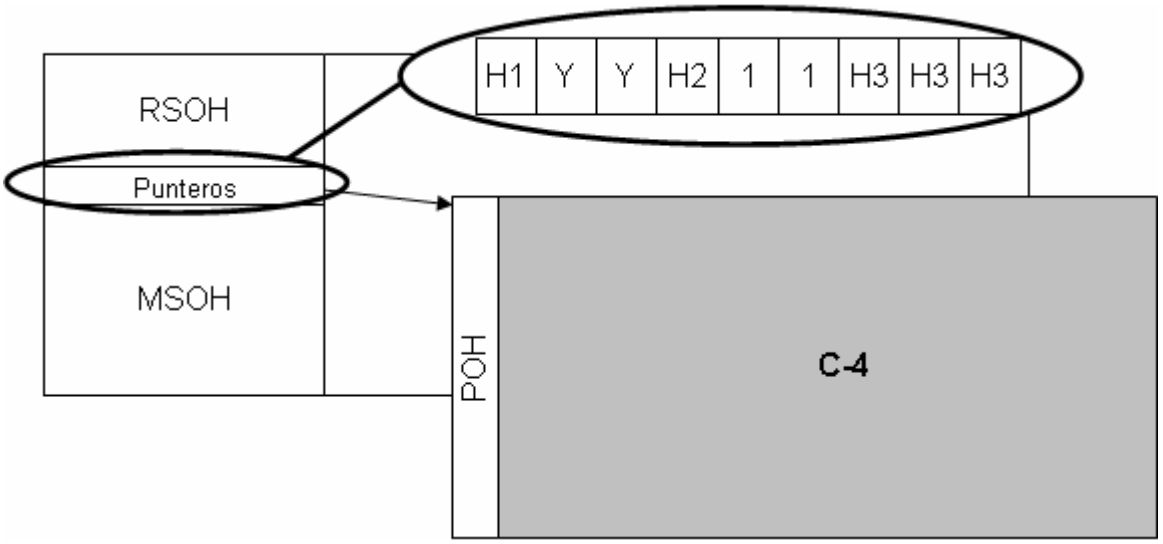


Figura 2.22: Unidad administrativa de orden 4

2.2.5.1 Alineamiento

El alineamiento consiste en asociar un puntero a un *VC* para obtener una *AU* o una *TU*. Los procedimientos de alineación recomendados por la *ETSI* son:

1. *AU-4*: Como puede observarse en la figura 2.22, el puntero de *AU-4* está constituido principalmente por los octetos Hl, H2 y H3.

2. *TU-3*.

3. *TU-12*.

En este proyecto se trata exclusivamente con unidades administrativas de orden superior (*AU-4*).

2.2.5.2 Punteros

Concepto de puntero

En una red síncrona pueden aparecer problemas derivados de la pérdida de sincronismo en algún punto de ésta. En principio, todos los nodos de red van a estar sincronizados por la distribución de una señal de reloj común. Si se interrumpe dicha distribución de reloj a un nodo, éste pasa a sincronizarse mediante un reloj local.

Dicho esto, se puede deducir que habrá problemas si el contenido multiplexado de una *STM-1*, que posee una cierta frecuencia, se transfiere a otra *STM-1* con diferente frecuencia. Esto se evita con la utilización del puntero. Si el reloj de la nueva trama *STM-1* tiene una frecuencia más baja que la del reloj utilizado para multiplexar la carga en algún lugar de la red, el resultado será que la memoria de transferencia se desbordará. En consecuencia deberá hacerse una justificación negativa, la cual se definirá más adelante.

Por el contrario, si el reloj tiene una frecuencia más elevada que la carga entrante, la memoria tenderá a vaciarse. Antes de que esto suceda deberá hacerse una justificación positiva, también explicada en esta misma sección.

Además, los punteros pueden ser necesarios en los terminales de línea. Así, en caso de que se pretenda multiplexar diferentes *STM-1* que llegan con diferente retardo, puede hacerse mediante el ajuste de punteros sin la utilización de memorias, asignando de manera independiente las cargas entrantes en la trama saliente.

En resumen, puede decirse que la utilización de punteros trae consigo dos claras ventajas:

1. El equipo se hace insensible a pequeñas desviaciones en frecuencia y fase entre las señales en la red.

2. Se reduce considerablemente la necesidad de memoria en los equipos.

Funciones de los punteros

Los punteros, como ya se ha visto, son octetos en posiciones fijas dentro de la trama, que realizan dos funciones:

1. Permiten asignar de forma flexible e independiente a los distintos *VC*'s dentro del área de carga. Esto significa que los *VC*'s pueden "flotar" dentro de la trama. Si una trama está formada por *N VC*'s, existirán *N* punteros, uno por cada *VC*. Así, cada *VC* puede situarse en una posición diferente dentro de la trama.

2. Permiten absorber, mediante un sistema de justificación positiva/nula/negativa, las diferencias de frecuencia entre las diferentes señales que van a constituir la *STM-N*.

2.2.5.3 El puntero de *AU-4*

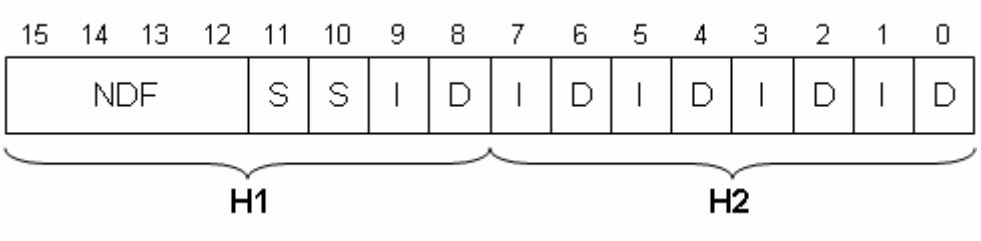


Figura 2.23: Formato del puntero de *AU-4*

El puntero de *AU-4* está contenido en los nueve primeros octetos de la cuarta línea de la trama *STM-1*. El puntero está constituido por los octetos *H1* y *H2* formando así una palabra de 16 bits, cuyo formato puede verse en la figura 2.23.

Los últimos diez bits (bits *I* y *D*) forman un número binario que designa la posición en el área de carga del *STM*, en donde se encuentra el inicio del *VC-4*.

De esta manera, el procesador del *VC* sólo tiene que contar posiciones a partir del primer octeto de la cuarta línea del área de carga (posición cero).

En el caso de *STM-1* las posiciones se cuentan cada tres octetos (la posición 0 es el primer octeto que le sigue al puntero, la posición 1 el cuarto, etc.). Como el área de carga contiene:

$$261 \times 9 = 2349 \text{ octetos,}$$

habrán en total:

$$2349 / 3 = 783 \text{ posiciones}$$

Por lo tanto, la gama de valores del puntero *AU-4* es un número binario de 0 a 782. Como ya se ha comentado con anterioridad, esto es debido a que en la jerarquía norteamericana hay tres punteros en lugar de un solo para cada *VC* de cada *STS-1*.

Sabiendo que en una *STM-1* las posiciones se cuentan cada 3 octetos, debido a la causa anteriormente mencionada, en una señal *STM* de nivel *N* (lleva multiplexadas *N STM-1*), las posiciones se contarán cada *N*x3 octetos:

$$\text{desplazamiento} = N \times 3 \times [\text{Valor puntero}] + 1$$

Esta fórmula da idea del número de octetos de carga útil, a partir del último *H3*, en el que se encuentra el primer octeto (*J1*) del *VC*.

Los octetos *H3* constituyen un área de carga extra, utilizada de forma eventual cuando hay necesidad de justificación negativa. La justificación positiva, por su parte, se efectúa dejando de enviar información en los 3x*N* octetos del área de carga que siguen al último *H3*.

La carga sólo puede ser localizada una vez se haya interpretado el puntero, por lo tanto, es lógico que la posición cero corresponda a la primera posición del área de carga siguiente al puntero. Esto quiere decir que el *VC* podrá estar localizado entre la posición 0 y la última posición de la tercera línea (correspondiente al valor 782) de la trama *STM* siguiente.

Codificación del puntero

En la tabla 3.5 se muestra la codificación del puntero, donde puede apreciarse que los cuatro primero bits del puntero, denominados *N*, constituyen la bandera de nuevos datos (*NDF*), que permite un cambio arbitrario del puntero. Normalmente está desactivada con la codificación *0110b*.

Octeto *H1*								Octeto *H2*							
15	14	13	12	11	10	9	8	7	6	5	7	3	2	1	0
N	N	N	N	S	S	I	D	I	D	I	D	I	D	I	D
NDF				Tipo		Valor del puntero *AU-4*									
0 1	1 0	1 0	0 1	1	0	0…782									
1	0	0	1	S	S	1	1	1	1	1	1	1	1	1	1
1	0	0	1	S	S	1	1	1	1	1	0	0	0	0	0
I: bit de incremento D: bit de decremento N: bit de bandera de nuevos datos (*NDF*) Cuando aparece la *AIS* el puntero pone a todos															

Tabla 2.7: Codificación del puntero de *AU-4*

Los bits *SS* para un puntero *AU-4* (excepto para la indicación de puntero concatenado) poseen el valor fijo 10.

La justificación negativa se indica invirtiendo la mayoría de los bits *D* del puntero, indicando así que el *VC* se encontrará una posición antes. En cuanto a la positiva, se indica invirtiendo la mayoría de los bits *I*, por lo que el *VC* estará una posición más allá que el anterior.

Justificación de frecuencia

Si hay diferencia entre relojes de los nodos de una red síncrona, implica que existe una diferencia entre la velocidad de la trama del *VC-4* y la de la *AU-4* en la que tiene que copiarse, por lo tanto, el valor del puntero aumentará o disminuirá, según la necesidad.

Las operaciones de puntero deben separarse, al menos, por tres tramas en las cuales el puntero permanezca constante.

Justificación negativa: Decremento del puntero

Si la velocidad de trama del *VC-N* es demasiado rápida con respecto a la del *AUG*, la alineación del *VC-N* debe avanzarse en el tiempo de forma periódica y el valor del

puntero debe disminuirse en uno. Esta operación se indica con la inversión de la mayoría de los bits *D* de la palabra de puntero, apareciendo octetos de justificación negativa (tres en el caso de una *STM-1*) en los *H3* de la trama de *AU-4* que contiene los bits *D* invertidos.

En la trama siguiente el puntero presentará el nuevo valor, una unidad menor que el anterior. Si el valor del anterior era 0, el nuevo valor será 782, pues es la posición anterior a 0. Esto indica que el inicio del nuevo *VC* ocupará la posición inmediatamente anterior a la que venía ocupando y permanecerá en ella hasta que se produzca la próxima justificación, o se reciba un puntero con la bandera de nuevos datos activada (*NFS*=100l). Véanse las figuras 2.24 y 2.25.

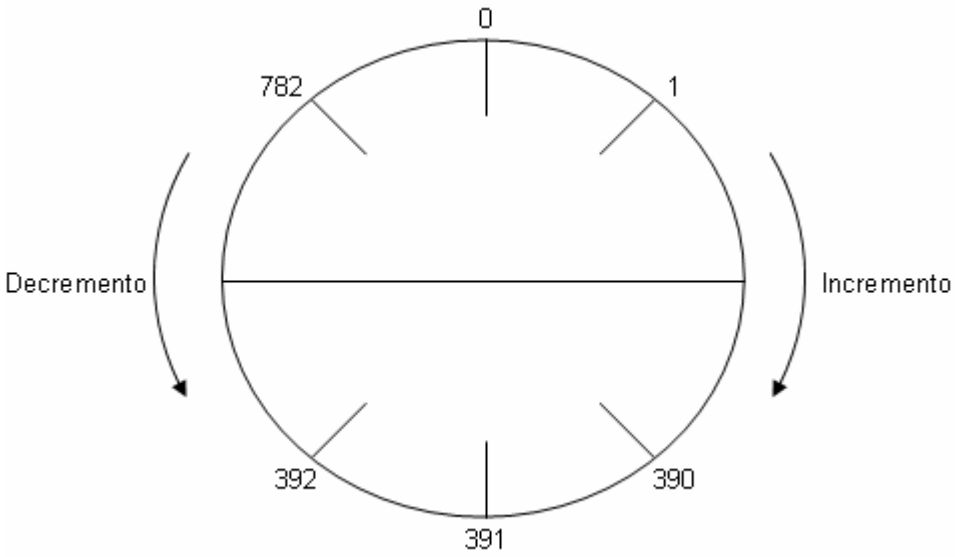


Figura 2.24: Incremento y decremento de los valores del puntero *AU-4*

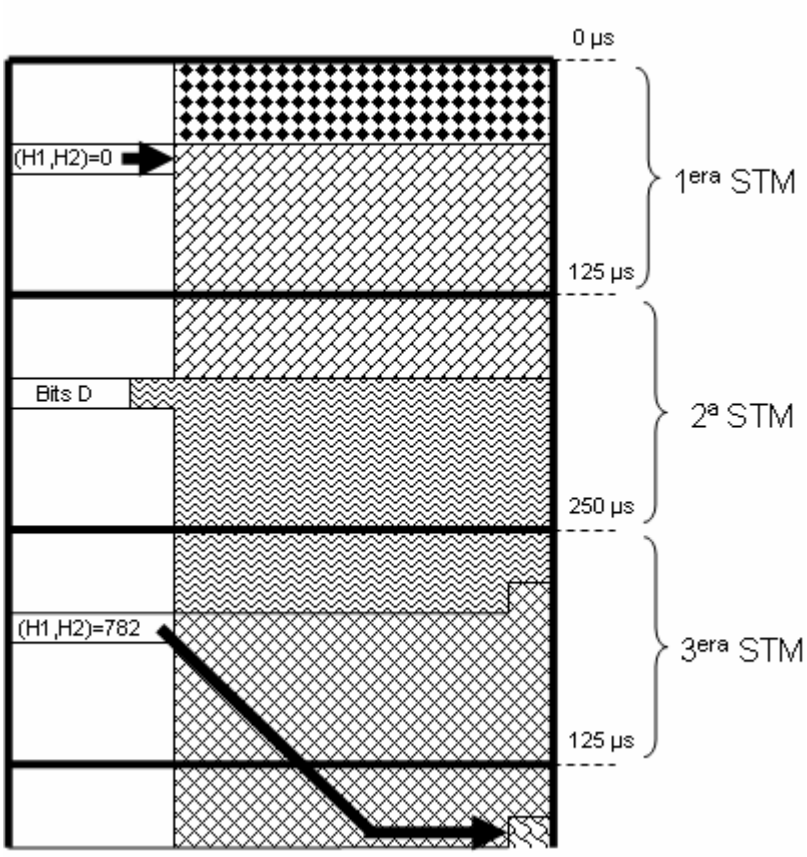


Figura 2.25: Caso especial en la operación de decremento de un puntero

Justificación positiva: Incremento del puntero

Si la velocidad de trama del *VC-N* es demasiado lenta con respecto a la del *AUG*, la alineación del *VC-N* debe retroceder en el tiempo de forma periódica y el valor del puntero debe aumentarse en uno, Esta operación se indica con la inversión de la mayoría de los bits *I* de la palabra del puntero, apareciendo octetos de justificación positiva (son octetos de relleno, el receptor los ignora) inmediatamente después del último octeto *H3* de la trama de *AU-4* que contiene los bits *I* invertidos.

En la trama *STM* inmediatamente consecutiva, el inicio del próximo *VC* va a estar desplazado una posición (3 octetos en el caso de *STM-1*) hacia delante, permaneciendo en esa nueva posición hasta una nueva justificación, o hasta que se reciba un puntero con la bandera de nuevos datos activada (*NDF=1001b*). Si el valor anterior era 782, el

nuevo valor será 0, pues esa es la posición que sigue a 782. Véanse las figuras 2.24 y 2.26.

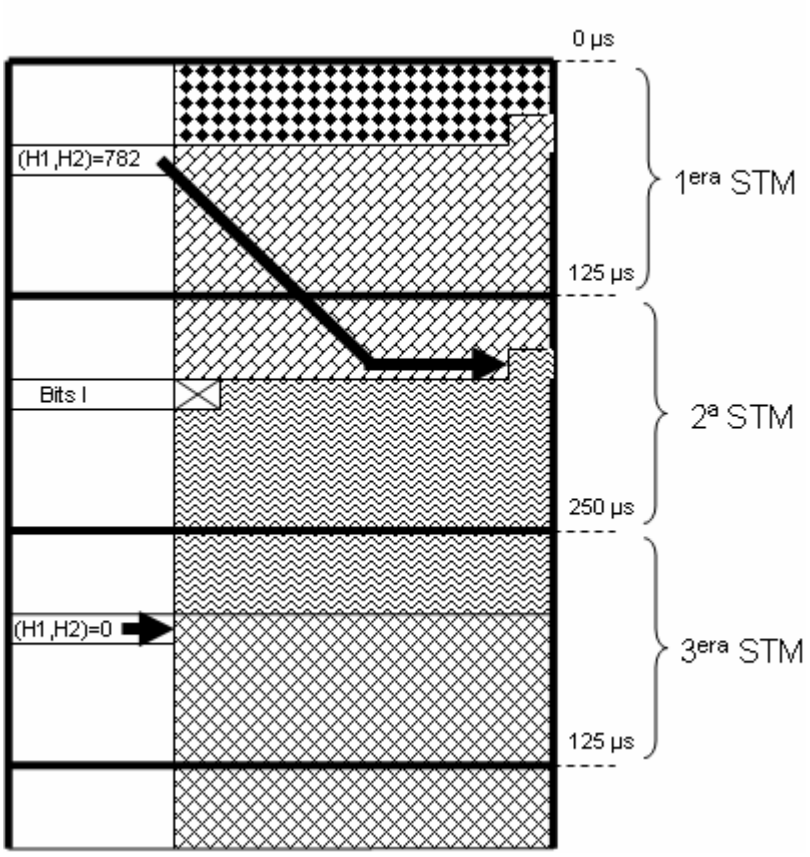


Figura 2.26: Caso especial en la operación de incremento de un puntero

Bandera de nuevos datos

Como ya se ha visto, los bits 15 a 12 (bits *N*) de la palabra del puntero llevan una *NDF*, que permite una modificación arbitraria del valor del puntero.

Se asignan 4 bits a la bandera para permitir la corrección de errores. La operación normal se indica con un código de "0110" en los bits *N*. La *NDF* se indica por inversión de los bits *N* a "1001". Una *NDF* debe interpretarse como activada cuando tres o más de los cuatro bits coinciden con el esquema" 1001" y debe interpretarse como desactivada cuando tres o más de los cuatro bits coinciden con el esquema "0110". Los demás valores (es decir, "0000", "0011", "0101", "1010", "1100" Y "1111") deben interpretarse como no válidos.

La nueva alineación se indica con el valor del puntero que acompaña a la *NDF* y tiene efecto cuando se produce el desplazamiento indicado.

Cambios en los punteros

Un cambio en un puntero se traduce en un cambio de posición del *VC*, al que éste apunta, dentro del espacio de carga de la trama *STM*. Se han podido ver dos formas de modificar un puntero:

1. A través de una justificación de frecuencia, ya sea positiva o negativa. El puntero sólo podrá ser modificado en una unidad.

2. Mediante la activación de la *NDF*. En este caso el puntero puede tomar cualquier valor.

No obstante, existe una tercera forma de modificar un puntero. Cuando se reciben (caso del receptor), tres punteros consecutivos normales con la *NDF* desactivada y con el mismo valor, indicará que el *VC* se ha desplazado a la posición que indica el puntero. El receptor tomará este valor como válido.

Generación del puntero

La siguiente lista resume las reglas para generar los punteros de *AU-4*:

1. Durante la operación normal, el puntero localiza el comienzo del *VC-4* dentro de la trama de la *AU-4*. La *NDF* se establece a *0110b*.

2. El valor del puntero solamente puede ser modificado por las operaciones 3, 4 o 5.

3. Si se requiere una justificación positiva, el valor vigente del puntero se envía con los bits *I* invertidos, y la oportunidad de justificación positiva subsiguiente se llena con

información ficticia. Los punteros subsiguientes contienen el valor del puntero previo aumentado en uno. No se permite ninguna operación subsiguiente de aumento o disminución hasta pasadas por lo menos tres tramas después de esta operación.

4. Si se requiere una justificación negativa, el valor vigente del puntero se envía con los bits *D* invertidos, y la oportunidad de justificación negativa subsiguiente se reescribe con datos reales. Los punteros subsiguientes contienen el valor del puntero previo disminuido en uno. No se permite ninguna operación subsiguiente de aumento o disminución hasta pasadas por lo menos tres tramas después de esta operación.

5. Si la alineación del *VC-4* cambia por cualquier razón distinta de las reglas 3 ó 4, se envía el nuevo valor del puntero acompañado de la *NDF* puesta a *1001b*. La *NDF* aparece solamente en la primera trama que contiene los nuevos valores. La nueva ubicación del *VC-4* comienza en la primera aparición del desplazamiento indicado por el nuevo puntero. No se permite ninguna operación subsiguiente de aumento o disminución hasta pasadas por lo menos tres tramas después de esta operación.

Interpretación del puntero

A continuación se resumen las reglas para interpretar los punteros de la *AU-4*:

1. Durante la operación normal, el puntero localiza el comienzo del *VC-4* dentro de la trama de la *AU-4*.

2. Cualquier variación del valor del puntero vigente se ignora, a no ser que se reciba tres veces consecutivas un mismo valor nuevo o que vaya precedido por una de las reglas 3, 4 ó 5. Cualquier valor nuevo recibido tres veces consecutivas tiene prioridad sobre las reglas 3 ó 4.

3. Si la mayoría de los bits *I* de la palabra del puntero están invertidos, se indica una operación de justificación positiva. Los valores subsiguientes del puntero aumentarán en uno.

4. Si la mayoría de los bits *D* de la palabra del puntero están invertidos, se indica una operación de justificación negativa. Los valores subsiguientes del puntero disminuirán en uno.

5. Si la *NDF* es interpretada como activada, el valor del puntero coincidente reemplazará el valor vigente cuando se produzca el desplazamiento indicado por el nuevo valor del puntero, a no ser que el receptor esté en un estado que corresponda a una pérdida de puntero.

Algoritmo para la detección de punteros

El algoritmo de procesamiento de puntero puede ser modelado por una máquina de estados finitos. Dentro del algoritmo de interpretación de puntero, se definen tres estados representados en la figura 2.27:

- NORM: estado *normal*.
- AIS: estado *AIS*.
- LOP: estado *LOP*

Las transiciones entre los estados serán eventos consecutivos (indicaciones), de forma que, por ejemplo, son necesarias tres indicaciones *AIS* consecutivas para pasar del estado normal al estado *AIS*. La única transición en un evento es la de un estado *AIS* al estado normal después de recibir una *NDF* habilitada con un valor de puntero válido.

Cabe señalar que, como el algoritmo sólo contiene transiciones basadas en indicaciones consecutivas, esto conlleva que las indicaciones no válidas recibidas no consecutivamente no activan las transiciones al estado *LOP*.

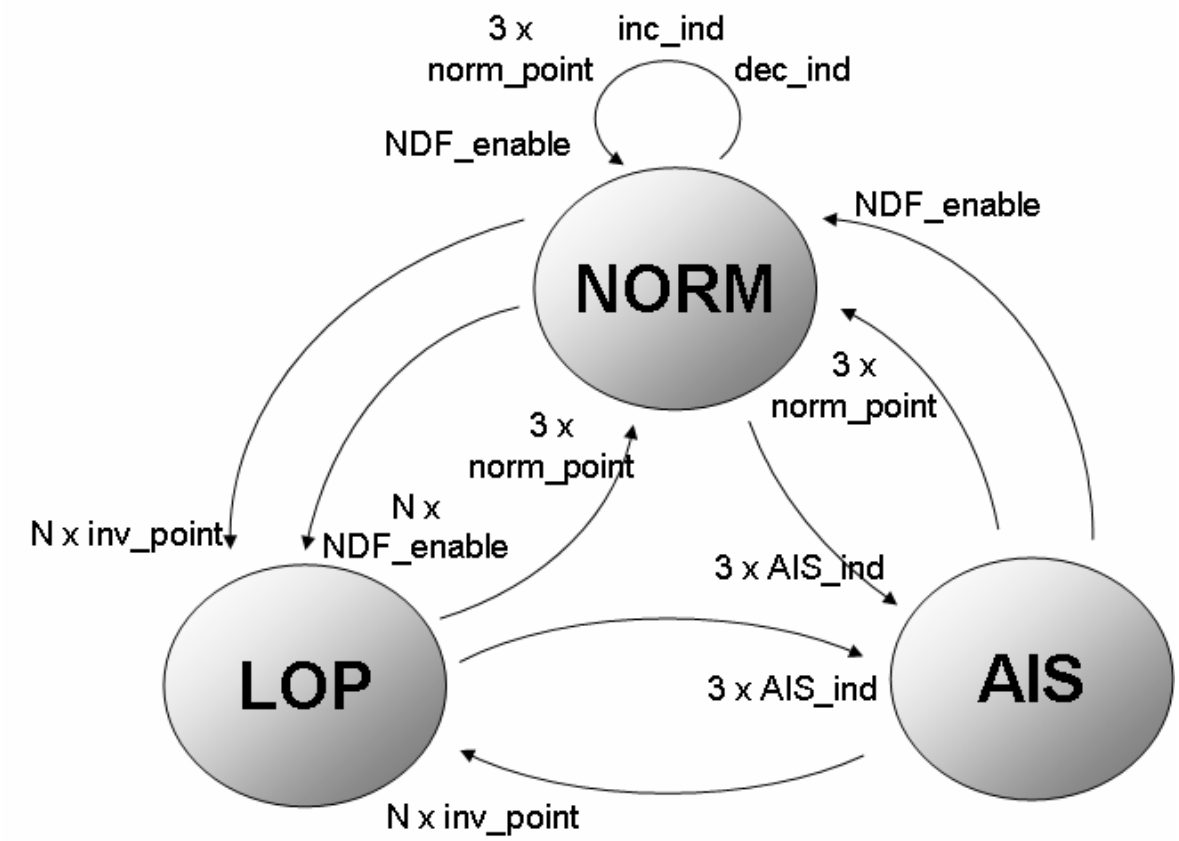


Figura 2.27: Diagrama de estados de interpretación de punteros

Se definen los siguientes eventos (indicaciones):

- *norm_point*: *NDF* normal. El valor del desplazamiento debe estar entre 0-782. Los bits *SS* deben poseer el valor *10b*.

- *NDF_enable*: *NDF* habilitada. El valor del desplazamiento debe estar entre 0 y 782. Los bits *SS* deben poseer el valor *10b*.

- *AIS_ind*: *11111111_11111111b*.

- *inc_Ind*: *NDF* normal. Los bits *SS* deben poseer el valor *10b*. La mayoría de los bits *I* deben estar invertidos. No debe haber mayoría de bits *D* invertidos.

- *dec_ind*: *NDF* normal. Los bits *SS* deben poseer el valor *10b*. La mayoría de los bits *D* deben estar invertidos. No debe haber mayoría de bits *I* invertidos.

- *inv _point*: Cualquier otro, o bien, *norm_point* con valor de desplazamiento fuera del margen de valores 0-782.

Las transiciones indicadas en el diagrama de estados se definen como sigue:

- *inc_ind/dec_ind*: Ajuste de desplazamiento (indicación de incremento o decremento).

- *3 x norm_point*: Tres indicaciones *norm_point* iguales consecutivas.

- *NDF_enable*: Una indicación *NDF_enable*.

- *3 x AIS_ind*: Tres indicaciones *AIS* consecutivas.

- *N x inv_point*: *N inv_point* ($8 \leq N \leq 10$) consecutivas.

- *NDF_enable*: *N NDF_enable* ($8 \leq N \leq 10$) consecutivas.

2.2.6 Multiplexación síncrona

En apartados anteriores, se ha dicho que a partir de *N* señales *STM-1* se puede obtener una señal *STM* de nivel *N*. Esto no es del todo correcto, en realidad lo que se combinan son *N AUG*'s para formar una *STM-N*. Si se trata con unidades administrativas de orden superior (*AU-4*), entonces hablar de *AUG* o *AU-4* es equivalente.

La multiplexación de dichas *AUG*'s tiene lugar a nivel de octeto. El paso siguiente es añadir la *SOH* correspondiente.

La creación de una trama *STM-N* lleva implícita la construcción de un *VC* capaz de transportar a *N VC-4*'s. Existen dos tipos de *VC*, que se verán en los subsiguientes apartados.

2.2.6.1 *VC-4-N*

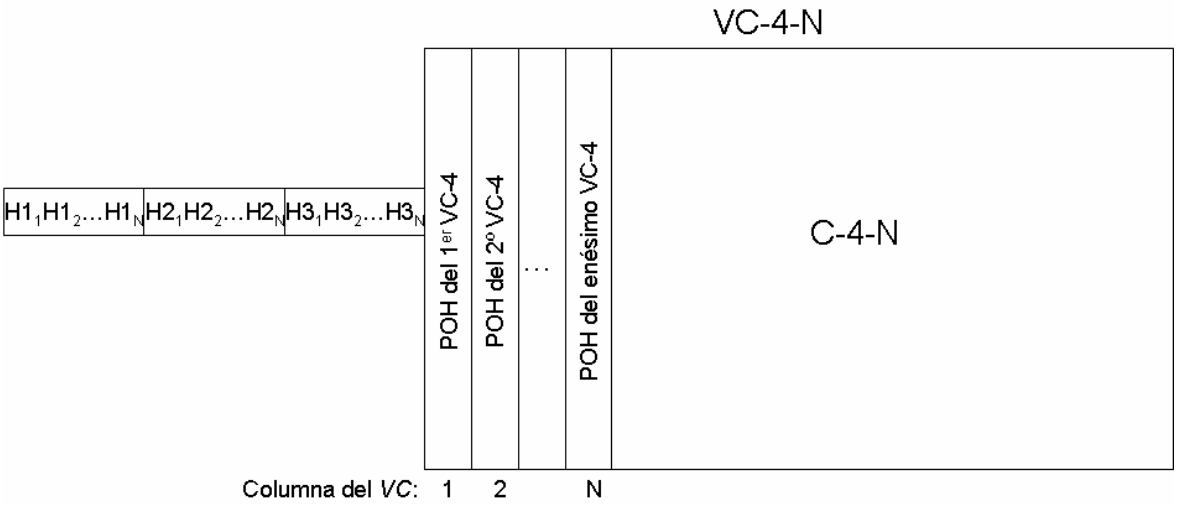


Figura 2.28: Estructura de un *VC-4-N* y punteros *AU-4* asociados

Una trama *STM-N*, formada a partir de un *VC-4-N* (*N VC-4* multiplexados), poseerá *N* parejas de punteros (*H1*, *H2*).

Cada una de estas parejas apunta a un *VC-4*, de tal forma que la primera pareja de punteros señale la posición del primer *VC-4* y así consecutivamente, hasta llegar a la enésima pareja que apunta al enésimo *VC-4*.

Las primeras *N* columnas del *VC-4-N* estarán ocupadas por las taras de trayecto de los *N VC-4*'s, véase figura 2.28.

2.2.6.2 *VC-4-Nc*

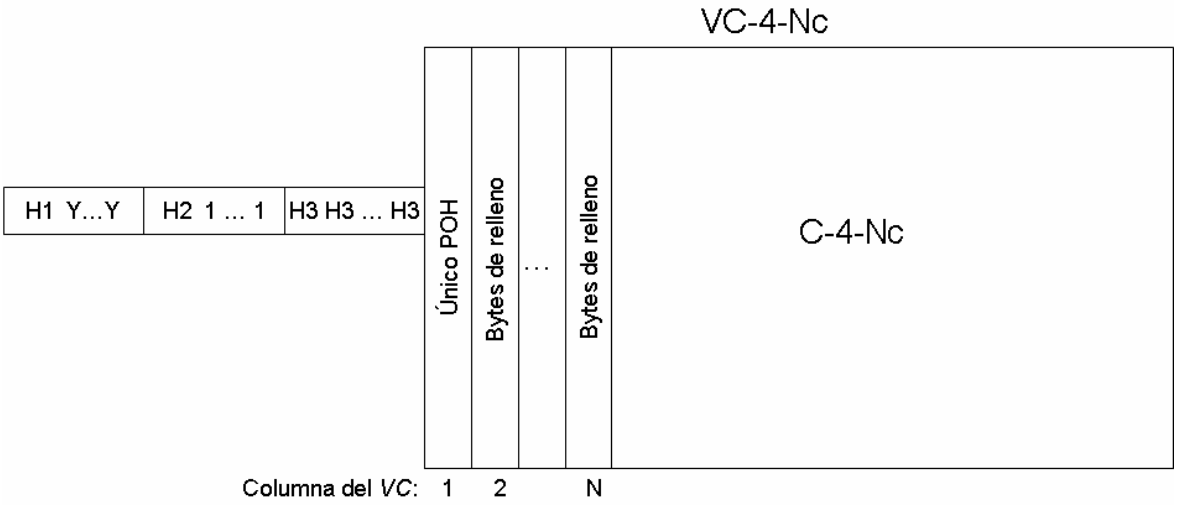


Figura 2.29: Estructura de un *VC-4-Nc* y punteros asociados

Existen aplicaciones en las que se requiere transportar una señal a alta velocidad. Es entonces cuando se hace necesaria la existencia de un contenedor de gran capacidad capaz de soportarla.

Se podría utilizar perfectamente la estructura del apartado anterior. No obstante, en este caso no conviene multiplexar *N AU-4*'s, ya que esto supondría tener que dividir la señal e introducirla en cada *C-4*. Además, supondría una carga computacional innecesaria, pues habría que interpretar los diferentes punteros para localizar a los *VC-4*'s correspondientes.

Lo que se necesitaría es un *VC* continuo que pueda ser localizado mediante una sola pareja de punteros (*H1*, *H2*). Dicho *VC* puede ser construido mediante la concatenación de *N VC-4*, pasando a llamarse *VC-4-Nc* (la "*c*" indica concatenación).

Esta clase de *VC* posee una sola *POH*, lo que supone una sola pareja de punteros asociados. No obstante, las *N-1* columnas del *VC-4-Nc* siguientes a la *POH*, poseerán bytes de relleno (ver figura 2.29) para mantener compatibilidad entre los sistemas. Por lo tanto, la primera *AU-4* de una *AU-4-Nc* tendrá un margen normal de valores del puntero. Todas las *AU-4* siguientes de la *AU-4-Nc* tendrán sus punteros puestos a indicación de

concatenación *1001b* en los bits 15 a 12. Los bits 11 y 10 estarán sin especificar y los bits del 9 al 0 estarán todos a nivel alto. La indicación de concatenación señala que los procesadores de puntero realizarán las mismas operaciones que las realizadas en la primera *AU-4* de la *AU-4-Nc*. En la figura 2.30 puede verse la estructura de punteros y su codificación.

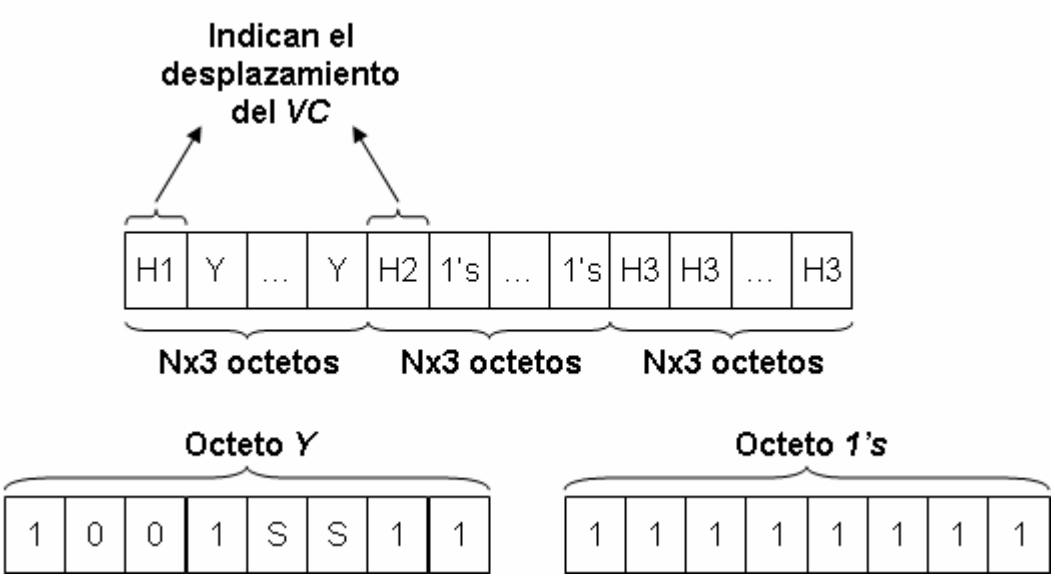


Figura 2.30: Codificación de los punteros en una *AU-4-Nc*

Algoritmo para la detección de punteros concatenados

Este algoritmo es similar al mostrado en el apartado 2.2.5.3, solo que en este caso, el algoritmo actuará solamente sobre los punteros que indiquen concatenación. Sobre la primera pareja de punteros (*H1*, *H2*) actuará el algoritmo presentado en el apartado ya mencionado. Pueden diferenciarse tres estados:

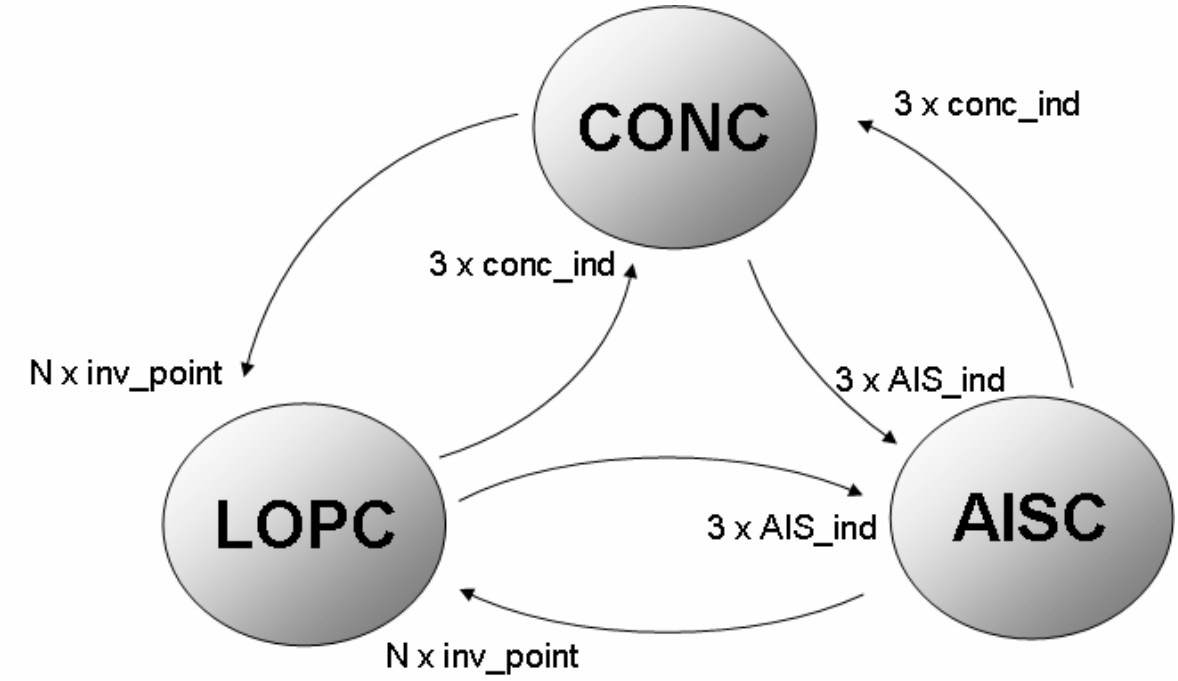


Figura 2.31: Diagrama de estados de interpretación de punteros concatenados

- *CONC*: estado *CONC*.
- *LOPC*: estado *LOPC*.
- *AISC*: estado *AISC*.

Se han definido los siguientes eventos:

- *conc_ind*. Todos los punteros deben presentar este patrón: *NDF* habilitada + *SS* + *1111111111b*.
- *AIS_ind*. Todos los punteros deben presentar este patrón: *1111111_11111111b*.
- *inv _point*: Si algún puntero presenta cualquier otro.

Los bits *SS* no están especificados en la Recomendación G.707 y por consiguiente no cuentan para el algoritmo.

Las transiciones indicadas en el diagrama de estados se definen como sigue:

- 3 x *AIS_ind*: Tres indicaciones *AIS* consecutivas.
- *N* x *inv_point*: *N inv_point* ($8 \leq N \leq 10$) consecutivas.
- 3 x *conc_ind*: Tres *conc_ind* consecutivas.

Se puede informar de dos tipos de defectos en las *AU* con la cabida útil concatenada:

- Pérdida de puntero (*LOPC*).
- *AIS* de trayecto (*AISC*).

El defecto pérdida de puntero (señal de error *AU-LOP*) se ha definido como una transición del intérprete de puntero del estado normal al estado *LOP* o al estado *AIS*, o una transición del estado *CONC* al estado *LOPC* o al estado *AISC*. En caso de que el intérprete de puntero esté en el estado *AIS* y el indicador de concatenación de las *AU* concatenadas esté en el estado *AISC*, se informará de un defecto *AU-AIS*.

2.2.7 Puntualizaciones en la transmisión/recepción de una señal *STM*

Conviene darse cuenta de que a pesar de que la trama *SDH* suele ser representada de forma rectangular, la transmisión/recepción se realiza serie, es decir, bit a bit, de tal forma que en realidad lo que se tiene es una ristra de unos y ceros en serie.

Como ya se ha indicado en el apartado 2.2.3.1, el primer octeto transmitido/recibido es el que ocupa la esquina superior izquierda, el sentido de la transmisión/recepción continúa barriendo las filas de izquierda a derecha, de tal forma que el último octeto en ser transmitido/recibido sea el que ocupa la esquina inferior derecha. Véase figura 2.14.

A nivel de octeto el primer bit en ser transmitido/recibido es el bit 7, que es el de mayor peso y el último es el bit 0, el de menor peso.

2.2.7.1 Aleatorización en una señal *STM*

Para que exista una buena distribución de reloj, es importante que la señal que se transmite posea suficientes transiciones. Para asegurar esto, se realizará la aleatorización de toda la señal *STM*, a excepción de la primera fila de la *RSOH*, previa a ser transmitida. Debido a esto, los octetos de la *RSOH* que se hallan a la derecha de *J0* deben ponerse a un valor que sigue el patrón fijo *10101010b*.

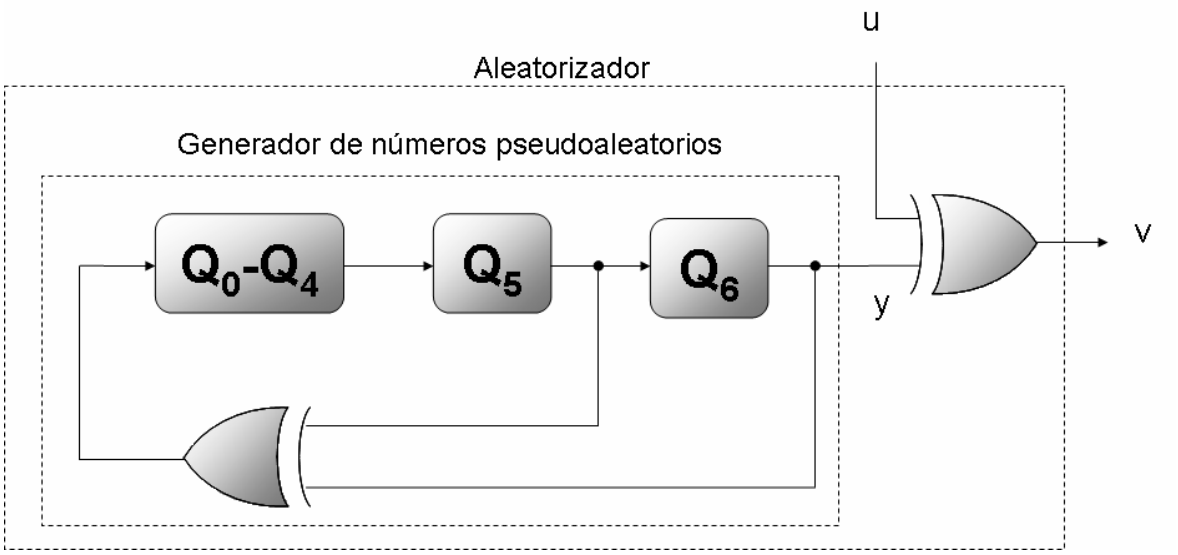


Figura 2.32: Aleatorizador

En la figura 2.32 se muestra el aleatorizador. Consta de un generador de números pseudoaleatorios[1] que generará, bit a bit, la secuencia de aleatorización:

```
1111111000000100000110000101000
0111100100010110011101010011111
1010000111000100100110110101100
1111011000110100101110111001100
0101010111111100000010000011000
0010100011110010001011001110100
                .
                .
                .
```

Para aleatorizar la trama se realizará la *or*-exclusiva entre la salida del generador (*y*) y los bits de la trama (*u*).

El generador se inicializará con el valor *1111111b* en el bit más significativo del octeto que sigue al último octeto de la primera fila de la *RSOH*.

[1] Para más información sobre los generadores de números pseudoaleatorios consultar las referencias [5] y [6] de la bibliografía.

En recepción, se aplicará el mismo polinomio para recuperar la señal. El proceso de desaleatorización tendrá lugar en toda la señal *STM* a excepción de la primera fila de la *RSOH*.

2.2.8 Particularizando para una *STM-16*

Como se mencionó en la introducción, el objeto del presente proyecto es un receptor *SDH* que maneja señales *STM-16* que transportan *VC*'s del tipo *VC-4-16c*, mostrado en la figura 2.33. El receptor procesará la trama de 32 en 32 bits, siendo el paralelizador (que no forma parte de este proyecto) el encargado de agrupar los bits recibidos. Por lo tanto, se define una palabra como la agrupación de 32 bits de la trama, o lo que es lo mismo, de 4 octetos.

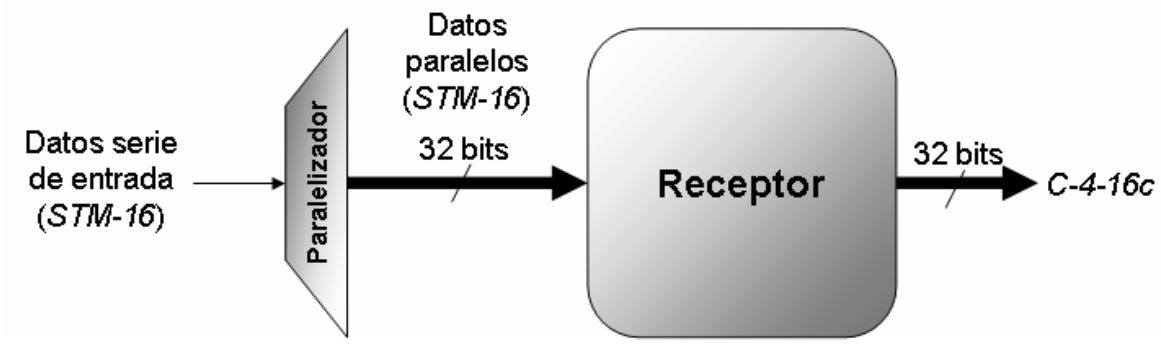


Figura 2.33: Receptor de tramas *STM-16*

La trama *STM-16* está compuesta de 38880 octetos o, lo que es lo mismo, 9720 palabras. Cada palabra está numerada de 0 a 9719 (véase figura 2.34), comenzando en los primeros 4 octetos de la primera fila (los 4 primeros *A1*'s) para, avanzando de izquierda a derecha y de arriba abajo, finalizar en los cuatro último octetos de la carga útil. Su duración es de 125 μs y su velocidad de transmisión es de 2,48832 Gbps.

El *VC-4-16c* consta de 9396 palabras (4176 octetos) numeradas de 0 a 9395 comenzando por la esquina superior izquierda y avanzado de izquierda a derecha, de arriba abajo.

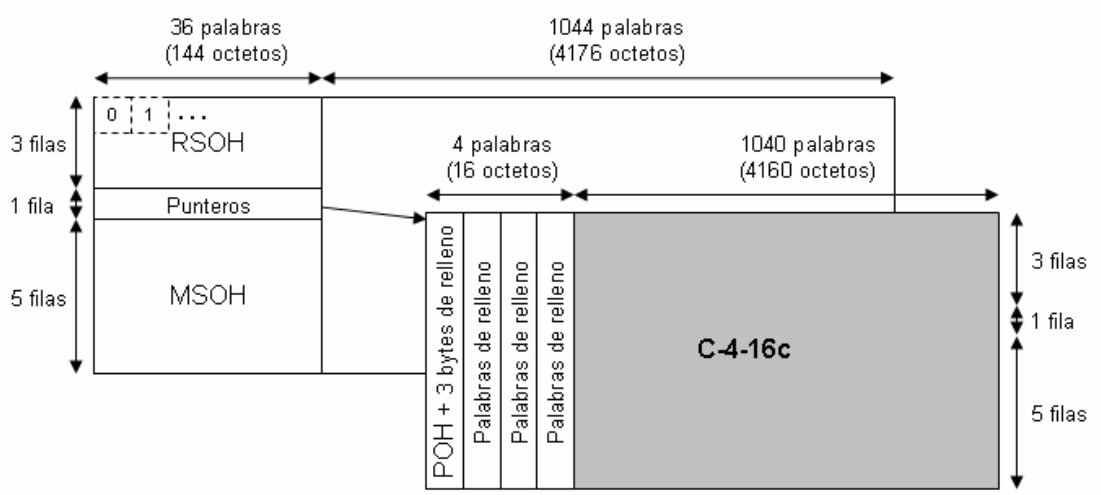


Figura 2.34: Estructura de una trama *STM-16*

La posición de cada octeto de la *SOH* y de los punteros en la trama, y de la *POH* en el *VC-4-16c* se muestra en las tablas 2.8 y 2.9, respectivamente.

Octetos de *SOH* y punteros	Palabra/-s	Bits
A1	0-11	[31:0]
A2	12-23	[31:0]
J0	24-27	[31:0]
B1	1080	[31:24]
E1	1092	[31:24]
F1	1104	[31:24]
D1	2160	[31:24]
D2	2172	[31:24]
D3	2184	[31:24]
B2	4320-4331	[31:0]
K1	4332	[31:24]
K2	4344	[31:24]
D4	5400	[31:24]
D5	5412	[31:24]
D6	5424	[31:24]
D7	6480	[31:24]
D8	6492	[31:24]
D9	6504	[31:24]
D10	7560	[31:24]
D11	7572	[31:24]
D12	7584	[31:24]

Tabla 2.8: Posición de cada octeto de la *SOH* y de los punteros en la trama

Octetos de *SOH* y punteros	Palabra/-s	Bits
S1	8640-8643	[31:0]
M1	8660-8663	[31:0]
E2	8664	[31:24]
H1	3240	[31:24]
H2	3252	[31:24]
H1 concatenados	3240 3241-3251	[23:0] [31:0]
H2 concatenados	3252 3253-3263	[23:0] [31:0]
H3	3264-3275	[31:0]

Tabla 2.8: Posición de cada octeto de la *SOH* y de los punteros en la trama (continuación)

Octetos de *POH*	Palabra/-s	Bits
J1	0	[31:24]
B3	1044	[31:24]
C2	2088	[31:24]
G1	3132	[31:24]
F2	4176	[31:24]
H4	5220	[31:24]
F3	6264	[31:24]
K3	7308	[31:24]
N1	8352	[31:24]

Tabla 2.9: Posición de cada octeto de la *POH* en el *VC-4-16c*

En el próximo capítulo se describe el método de diseño que se aplica al estándar *SDH* estudiado en este tema.

Capítulo 3

Dispositivos FPGA

Los *PLD*'s (dispositivos lógicos programables) son circuitos integrados utilizados por los diseñadores de *hardware* para realizar funciones lógicas específicas. Contienen un gran número de funciones lógicas básicas, mayormente puertas y biestables, en las que el usuario, mediante software de aplicación y una placa con zócalo para conectar los dispositivos, puede definir su interconexión, tanto a nivel de realizar el circuito interno, como la asignación de los terminales de entrada y salida.

Utilizando dispositivos reprogramables es posible realizar la depuración hardware del diseño, realizando tantos intentos como sea necesario, para la corrección de errores de funcionamiento y mejora del diseño. De esta forma se consigue una reducción de tiempo y coste en la verificación hardware.

Es posible encontrar *PLD*'s de distinto coste, arquitectura, velocidad y capacidad de integración, desde los *SPLD*'s (dispositivos lógicos programables sencillos), de poca

capacidad y con una arquitectura simple, hasta las *FPGA*'s (redes de puertas programables), que ofrecen normalmente la mayor capacidad y complejidad.

En este capítulo se describe la *FPGA Spartan-3* de *Xilinx*, en concreto la *XC3S1000*, que es la utilizada en el presente proyecto, y la placa *Diligent Spartan-3 Starter Board*, mediante la cual se realiza la programación de la *FPGA* y la verificación hardware del diseño, profundizando en los recursos utilizados para la realización del proyecto. Para más información pueden consultarse las referencias [11] y [12] de la bibliografía.

3.1 La serie *Spartan-3* de *Xilinx*

En los siguientes apartados se presentan las características y arquitectura de la *FPGA XC3S1000*, así como su programación.

3.1.1 Características

En la tabla 3.1 se resumen las características de la *FPGA XC3S1000*.

Puertas equivalentes		1 Millón
Celdas lógicas equivalentes		17.280
Matriz de *CLB*'s	Filas	48
	Columnas	40
	Total de *CLB*'s	1.920
RAM distribuida		120 Kbits
Bloques *RAM*		432 Kbits
Multiplicadores dedicados		24
DCM's		4
Nº máximo de entradas y salidas de usuario		391
Nº máximo de entradas y salidas diferenciales		175

Tabla 3.1: Características de la *FPGA XC3S1000*

3.1.2 Arquitectura

La *FPGA XC3S1000* consta de 5 elementos fundamentales:

- *CLB*'s (bloques lógicos configurables).
- *IOB*'s (bloques de entrada y salida).
- Bloques *RAM* (memoria de acceso aleatorio).
- Bloques multiplicadores.
- *DCM*'s (controladores de reloj digital).

Estos elementos están organizados como se muestra en la figura 3.1. Un anillo de *IOB*'s rodea a una matriz de *CLB*'s con dos columnas de bloques *RAM* empotradas en la misma. Cada bloque *RAM* tiene un multiplicador dedicado. Los *DCM*'s se sitúan en los extremos de cada columna.

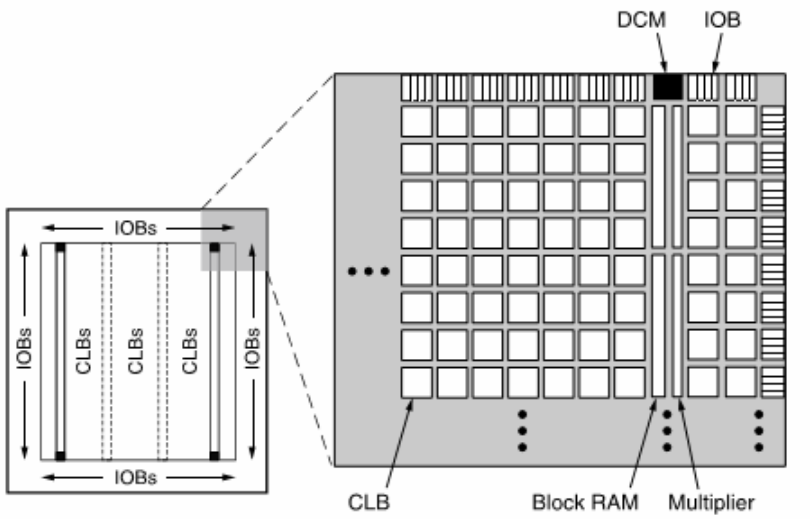


Figura 3.1: Disposición de los elementos de la *FPGA*

3.1.2.1 *IOB*

Los *IOB*'s proporcionan una interfaz bidireccional programable entre un pin de entrada y salida del encapsulado de la *FPGA* y la lógica interna de la misma.

En la figura 3.2 se muestra un diagrama simplificado de la estructura interna de un *IOB*. Hay tres caminos principales en el *IOB* con su propio par de elementos de almacenamiento:

- Camino de entrada: lleva la señal desde el pin hasta la línea *I* a través de un elemento de retardo opcional. Hay dos rutas alternativas a través de un par de elementos de almacenamiento hacia las líneas *IQ1* e *IQ2*. Las salidas *I*, *IQ1* e *IQ2* llegan a la lógica interna de la *FPGA*.

- Camino de salida: lleva las líneas de entrada *O1* y *O2*, procedentes de la lógica interna de la *FPGA* al pin, a través de un multiplexor y un driver triestado. Además de este camino directo, dispone de dos elementos de almacenamiento opcionales.

- Camino triestado: determina cuando el driver triestado de salida está en alta impedancia. Las líneas *T1* y *T2* llevan el dato desde la lógica interna de la *FPGA*, a través de un multiplexor, hasta el driver de salida. Además de este camino directo, dispone de dos elementos de almacenamiento opcionales. Cuando las líneas *T1* o *T2* se encuentran a nivel alto, el driver de salida está en alta impedancia.

Los elementos de almacenamiento pueden actuar como *Flip-Flops* tipo *D* activos por flanco de subida o como *Latches* tipo *D* activos por nivel. También pueden ser usados junto con un multiplexor *DDR* (doble velocidad de dato) para duplicar la velocidad de transmisión de los datos. Esto se consigue tomando los datos de cada registro en los flancos de subida y de bajada de la señal de reloj.

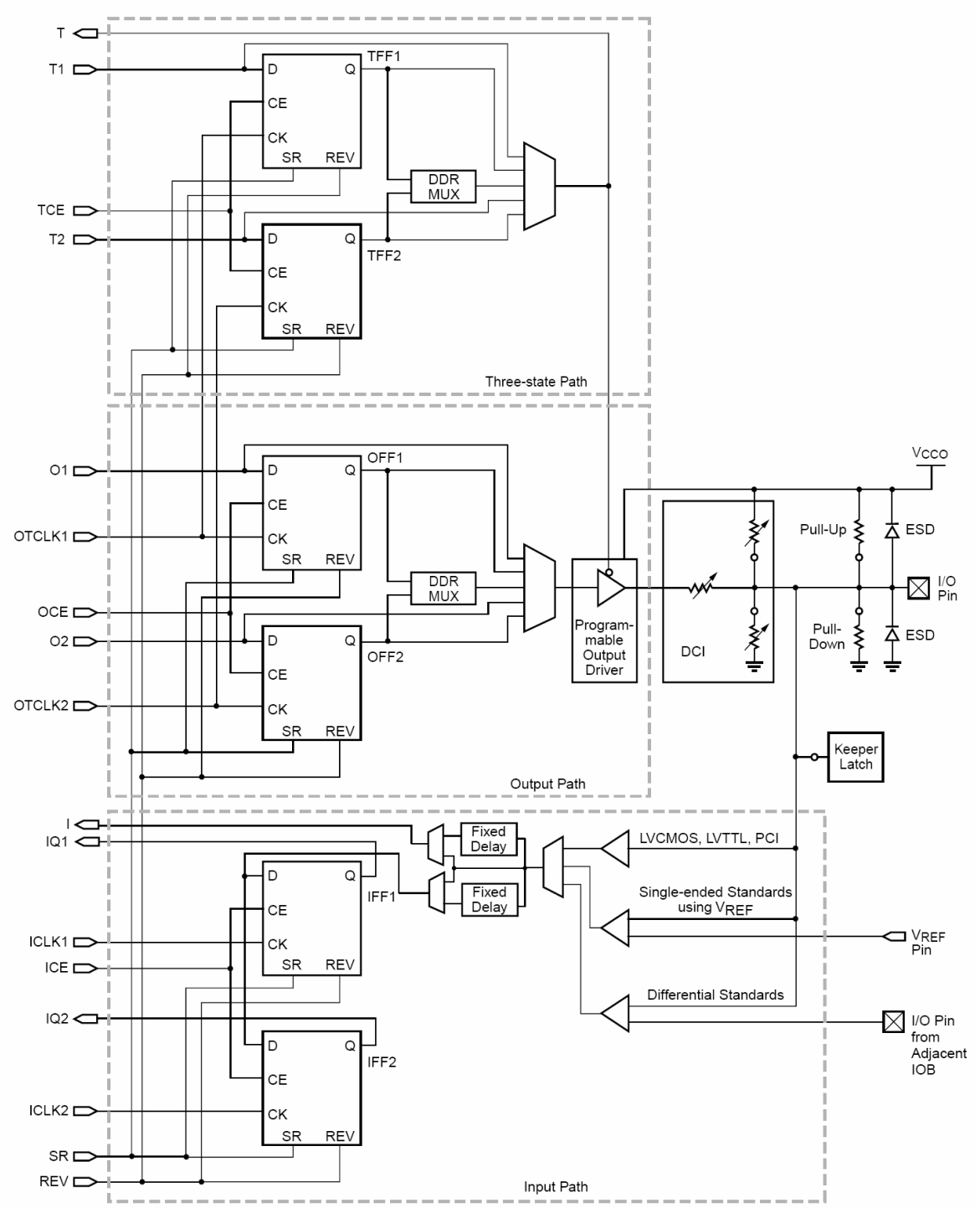


Figura 3.2: Diagrama de un *IOB*

3.1.2.2 *CLB*

Los *CLB*'s pueden ser programados para implementar una gran variedad de funciones lógicas así como para almacenar datos. Cada uno consta de 4 partes, como se muestra en la figura 3.3, denominadas *slices*. Dichas *slices* se agrupan en pares organizados en dos columnas con acarreo independiente.

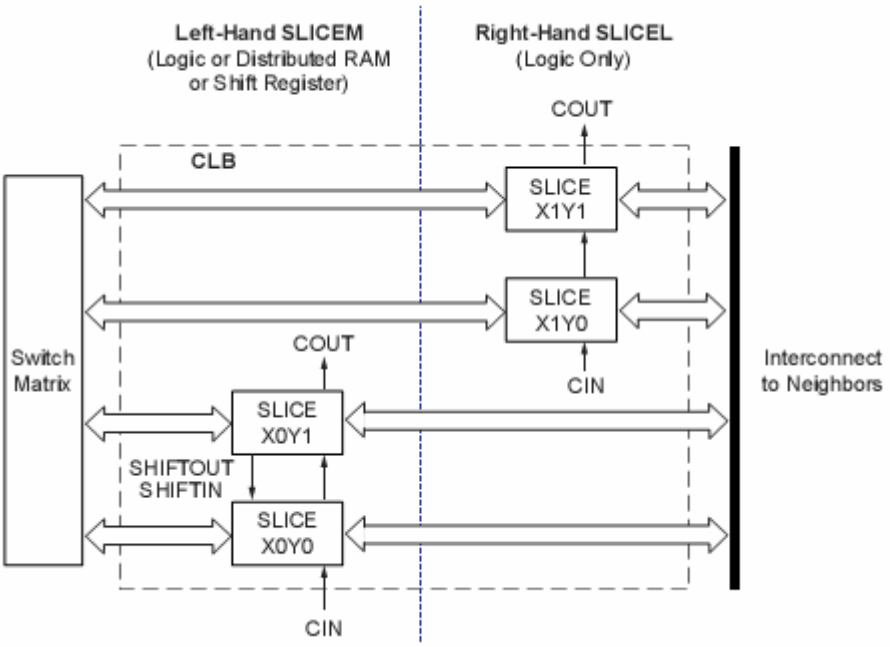


Figura 3.3: Disposición de las *slices* en el *CLB*

Todas las *slices* tienen elementos comunes: dos generadores de funciones lógicas, dos elementos de almacenamiento, multiplexores, lógica de acarreo y puertas aritméticas. Estos elementos proporcionan funciones lógicas, aritméticas y de *ROM* (memoria de sólo lectura), inicializada en el momento de la configuración de la *FPGA*.

Además de los elementos comunes, el par de *slices* de la izquierda soporta dos funciones adicionales: almacenamiento de datos utilizando *RAM* distribuida y desplazamiento de datos con registros de 16 bits. En la figura 3.4 se muestra el diagrama de una de las *slices* de la izquierda.

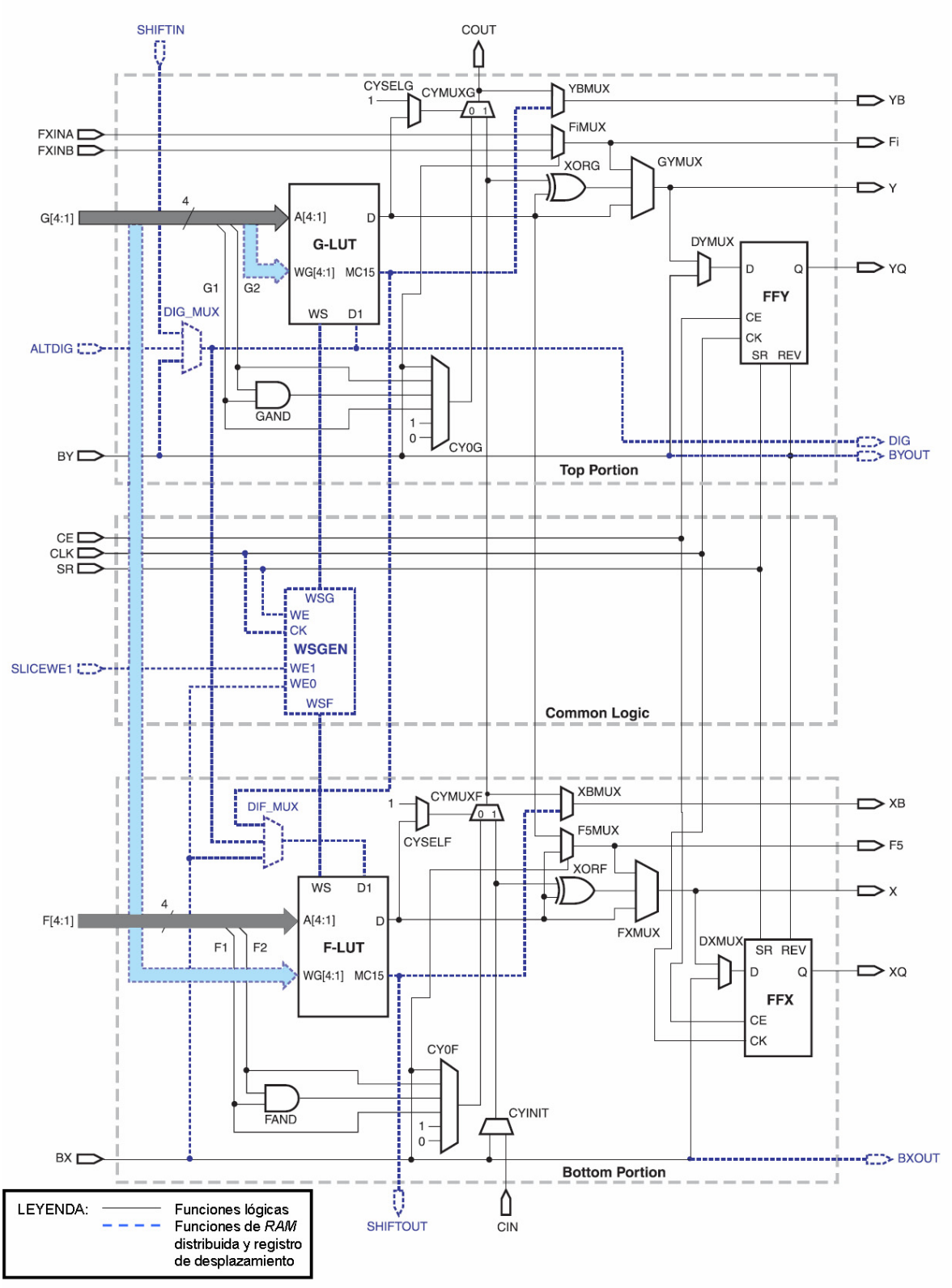


Figura 3.4: Diagrama simplificado de una *slice*

El generador de funciones lógicas, conocido también como *LUT* (tabla de verdad), es el recurso principal para implementar funciones lógicas. Además, cada *LUT* en cada par de *slices* de la izquierda del *CLB* puede ser configurado como *RAM* distribuida o como registro de desplazamiento de 16 bits.

Los elementos de almacenamiento, que pueden programarse para ser utilizados como registros tipo *D* activos por nivel o por flanco de subida, proporcionan un medio para sincronizar datos a una señal de reloj.

Los multiplexores combinan *LUT*'s, en la misma o en diferentes *slices*, para obtener operaciones lógicas más complejas.

La cadena de acarreo, junto con varias puertas de lógica aritmética dedicadas, proporciona implementaciones de operaciones matemáticas rápidas y eficientes.

El centro de operación de cada *slice* son dos caminos de datos casi idénticos, denominados *top* y *bottom*. La siguiente descripción utiliza los nombres asociados al camino *bottom* (las del *top* aparecen entre paréntesis). 4 líneas, *F1-F4* (*G1-G4*) entran en el *slice* y se conectan directamente al *LUT*. La salida de datos del *LUT* ofrece 5 posibilidades:

- Salir del *slice* a través de la línea *X* (*Y*).

- Servir como entrada al multiplexor *DXMUX* (*DYMUX*) que alimenta la entrada de datos *D* del elemento de almacenamiento *FFX* (*FFY*). La salida *Q* del elemento de almacenamiento sale del *slice*.

- Controlar el multiplexor *CYMUXF* (*CYMUXG*) en la cadena de acarreo.

- Con la cadena de acarreo, servir como entrada a la puerta *OR*-exclusiva *XORF* (*XORG*) que realiza operaciones aritméticas, dando su resultado en *X* (*Y*).

- Alimentar al multiplexor *F5MUX* para implementar funciones lógicas de más de 4 bits. Las salidas de ambos *LUT*'s sirven de entrada de datos para este multiplexor.

Además de estos dos caminos principales, existen otros dos que entran en el *slice* como *BX* y *BY*, con las siguientes posibilidades:

- Desviarse al *LUT* y al elemento de almacenamiento y salir como *BXOUT* (*BYOUT*) del *slice*.

- Desviarse al *LUT*, después pasar al elemento de almacenamiento por la entrada *D* antes de salir como *XQ* (*YQ*).

- Controlar el multiplexor *F5MUX* (*F6MUX*).

- A través de los multiplexores, servir de entrada para la cadena de acarreo.

- Alimentar a la entrada *D1* del *LUT*.

- *BY* puede controlar las entradas *REV* de los elementos de almacenamiento *FFY* y *FFX*.

- El multiplexor *DIG_MUX* puede conmutar *BY* a la línea *DIG*, que sale del *slice*.

3.1.2.3 Bloques *RAM*

La *Spartan-3* dispone de bloques *RAM* organizados en bloques síncronos de puerto dual de 18 Kbits. Estos bloques almacenan cantidades relativamente grandes de datos de forma más eficiente que la *RAM* distribuida vista en el apartado anterior. Tanto el ancho como la profundidad de la memoria pueden ser configurados.

Los bloques *RAM* y los multiplicadores están interconectados para permitir operaciones simultáneas. En este caso, como los multiplicadores almacenan sus entradas en las memorias, el ancho máximo de los datos de las memorias será de 18 bits.

En la figura 3.5 se muestra la estructura interna de un bloque *RAM*. Consta de dos puertos *A* y *B* que acceden de forma independiente al mismo bloque. Cada puerto tiene sus propias líneas de control, datos y reloj para realizar operaciones síncronas de lectura y escritura. Existen 4 caminos de datos básicos:

1. Lectura y escritura en el puerto *A*.

2. Lectura y escritura en el puerto *B*.

3. Transferencia de datos del puerto *A* al *B*.

4. Transferencia de datos del puerto *B* al *A*.

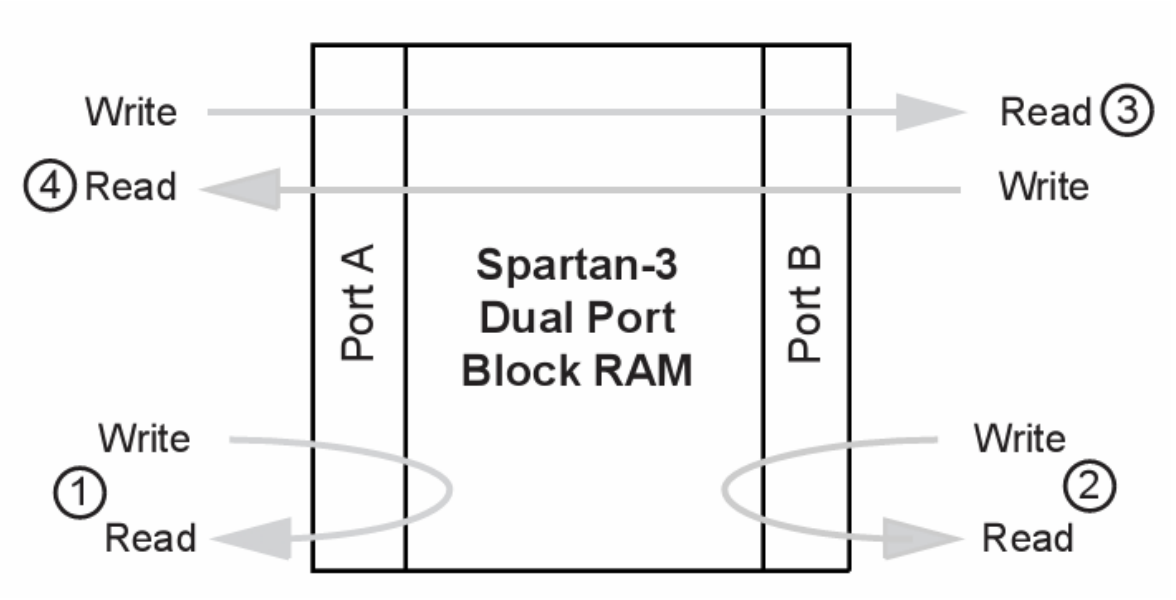


Figura 3.5: Estructura interna de un bloque *RAM*

3.1.2.4 Bloques multiplicadores

Los multiplicadores realizan el producto de sus dos entradas, que pueden ser de 18 bits, para operaciones con signo, y 17 bits para operaciones sin signo. El resultado es mostrado en su salida de 36 bits. Se pueden utilizar multiplicadores en cascada, permitiendo así operandos de más de 18 bits.

3.1.2.5 *DCM*

Los bloques funcionales de un *DCM* se muestran en la figura 3.6. Las funciones que realiza son las siguientes:

- Eliminación de las fluctuaciones de fase del reloj: gracias al *PLL* (bucle enganchado en fase) del que dispone.

- Síntesis de frecuencias: con el sintetizador de frecuencias digital se pueden generar frecuencias de reloj a partir de la señal de reloj de entrada. Las frecuencias que se pueden conseguir son la de la señal de reloj de entrada multiplicada por cualquier entero comprendido entre 2 y 32, ambos incluidos, o dividida por cualquier entero comprendido entre 1 y 32, ambos incluidos.

- Desplazamiento de fase: el *PLL* proporciona 9 salidas con distintas señales de reloj desfasadas. Además, el *DCM* también dispone de desplazadores de fase que permiten desplazamientos de fase más precisos.

Además de los elementos ya descritos los *DCM*'s constan de un cuarto componente, la lógica de estado. La función de este elemento es proporcionar el estado en que se encuentra el *DCM*, así como resetear el mismo a un estado inicial conocido.

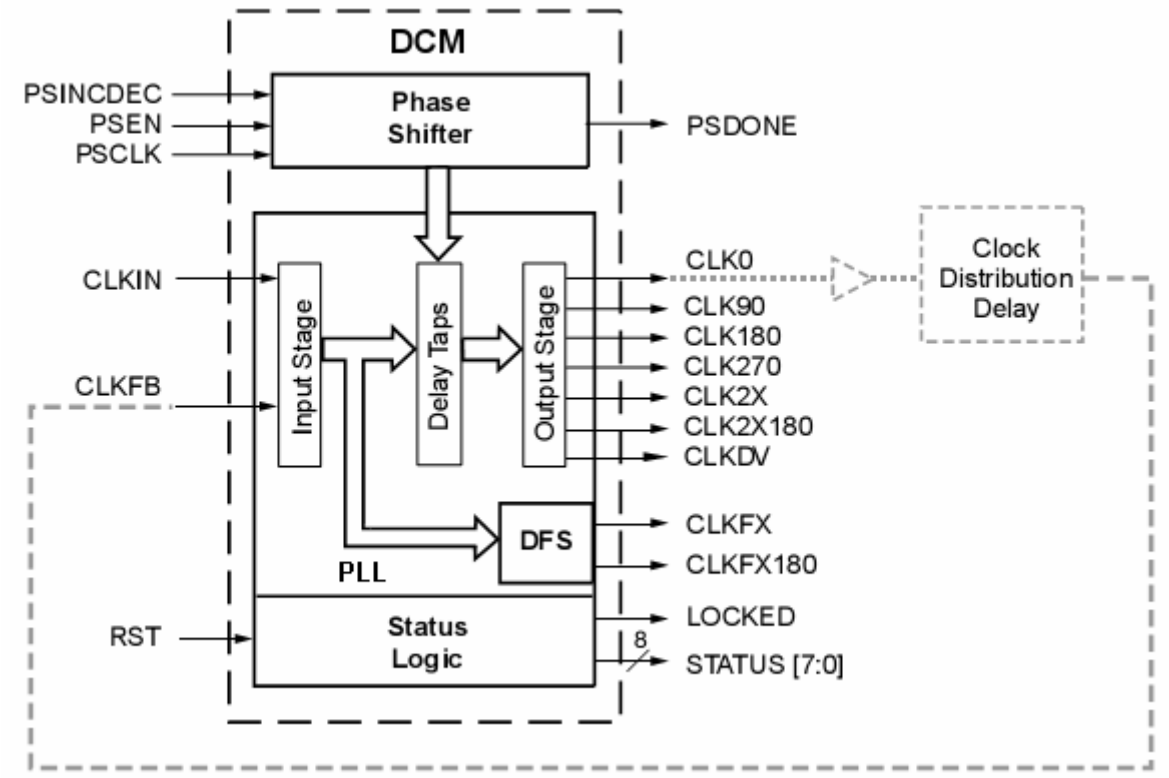


Figura 3.6: Bloques funcionales de un *DCM*

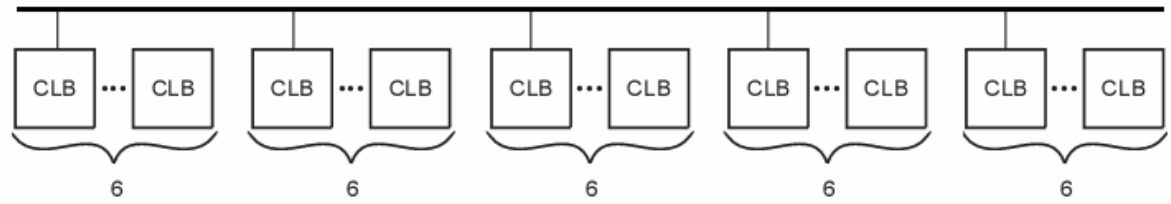
3.1.2.6 Ruteado

El ruteado transporta las señales entre los distintos elementos de la *FPGA*. Hay cuatro tipos de ruteado:

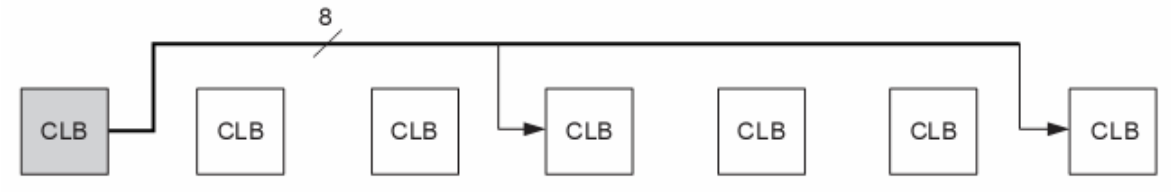
- Líneas *Long* (figura 3.7 a)): conectan uno de cada seis *CLB*'s. Debido a su baja capacidad, son las adecuadas para transportar señales de alta frecuencia con los mínimos efectos de carga, como por ejemplo las fluctuaciones de fase.

- Líneas *Hex* (figura 3.7 b)): conectan uno de cada tres *CLB*'s. Estas líneas están entre las *Long* y las *Double*, ya que alcanzan características de alta frecuencia y un mayor grado de conectividad.

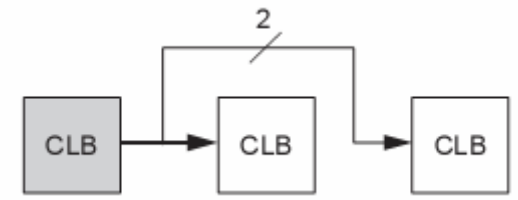
- Líneas *Double* (figura 3.7 c)): conectan cada *CLB*. Comparadas con las vistas anteriormente, permiten un mayor grado de flexibilidad al hacer las conexiones.



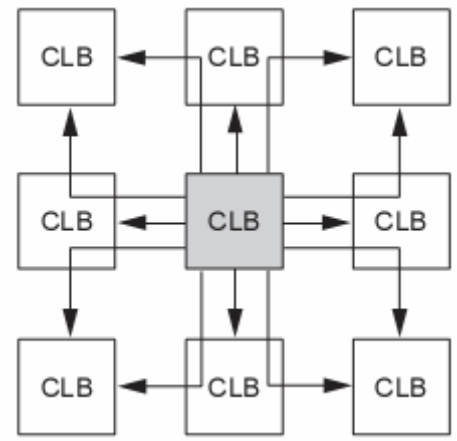
(a)



(b)



(c)



(d)

Figura 3.7: a) Líneas *Long*. b) Líneas *Hex*. c) Líneas *Double*. d) Líneas *Direct*

- Líneas *Direct* (figura 3.7 d)): permiten que cualquier *CLB* acceda directamente a los *CLB*'s vecinos. Estas líneas son las más ampliamente usadas.

- Líneas *Long* (figura 3.7 a)): conectan uno de cada seis *CLB*'s. Debido a su baja capacidad, son las adecuadas para transportar señales de alta frecuencia con los mínimos efectos de carga, como por ejemplo las fluctuaciones de fase.

- Líneas *Hex* (figura 3.7 b)): conectan uno de cada tres *CLB*'s. Estas líneas están entre las *Long* y las *Double*, ya que alcanzan características de alta frecuencia y un mayor grado de conectividad.

- Líneas *Double* (figura 3.7 c)): conectan cada *CLB*. Comparadas con las vistas anteriormente, permiten un mayor grado de flexibilidad al hacer las conexiones.

- Líneas *Direct* (figura 3.7 d)): permiten que cualquier *CLB* acceda directamente a los *CLB*'s vecinos. Estas líneas son las más ampliamente usadas.

3.1.3 Configuración

La *FPGA Spartan-3* se programa cargando los datos de configuración de la aplicación específica en la memoria interna de configuración, que controla todos los elementos funcionales y su conexión, a través de unos pines concretos. Estos pines de configuración son de dos tipos: los dedicados a esta función específica y los de doble propósito, pudiendo ser usados estos últimos como pines de entrada y salida de propósito general por el usuario una vez finalizada la configuración. Estos datos están disponibles en una memoria externa o en cualquier otro medio no volátil en o fuera de la placa donde está situada la *FPGA*.

El proceso de configuración se inicia cuando se alimenta la *FPGA* o cuando se activa el pin *PROG_B* a nivel bajo y de forma asíncrona. Dicho proceso consta de tres fases:

1. Se limpia la memoria de configuración.

2. Se cargan los datos de configuración.

3. Se activa la lógica mediante un proceso de inicialización.

La *FPGA* dispone de cinco modos de configuración, seleccionados a través de los pines de configuración *M0*, *M1* y *M2*:

- Modo serie esclavo: la *FPGA* recibe los datos de configuración de bit en bit por el pin *DIN* en los flancos de subida de la señal de reloj introducida por el pin *CCLK*, configurado como entrada.

- Modo serie maestro: es similar al modo serie esclavo, excepto en que es la propia *FPGA* la que proporciona la señal de reloj por el pin *CCLK*. Cuando el esclavo recibe la señal de reloj proporciona los datos en los flancos de subida.

- Modo paralelo esclavo: una fuente externa proporciona los datos de 8 bits (pines *DIN*/*D0-D7*), la señal de reloj (pin *CCLK*), una señal de selección activa a nivel bajo (pin *CS_B*) y una señal de escritura activa a nivel bajo (pin *RDWR_B*). La *FPGA* puede activar la señal *BUSY* (pin *DOUT* /*BUSY*), a nivel alto, para que el maestro mantenga el dato hasta que esté preparada para poder leerlo.

- Modo paralelo maestro: es similar al modo paralelo esclavo, solo que en este caso la *FPGA* suministra la señal de reloj por el pin *CCLK*.

- Modo *Boudary-Scan* (*JTAG*): obedece al estándar *IEEE 1149.1-1993* y al *IEEE 1532*. La configuración se realiza por el puerto *TAP* (puerto de acceso de test).

3.2 *Diligent Spartan-3 Starter Board*

La placa *Diligent Spartan-3 Starter Board* (figura 3.8) proporciona una plataforma de desarrollo y evaluación de diseños para la *FPGA Spartan-3* fácil de usar.

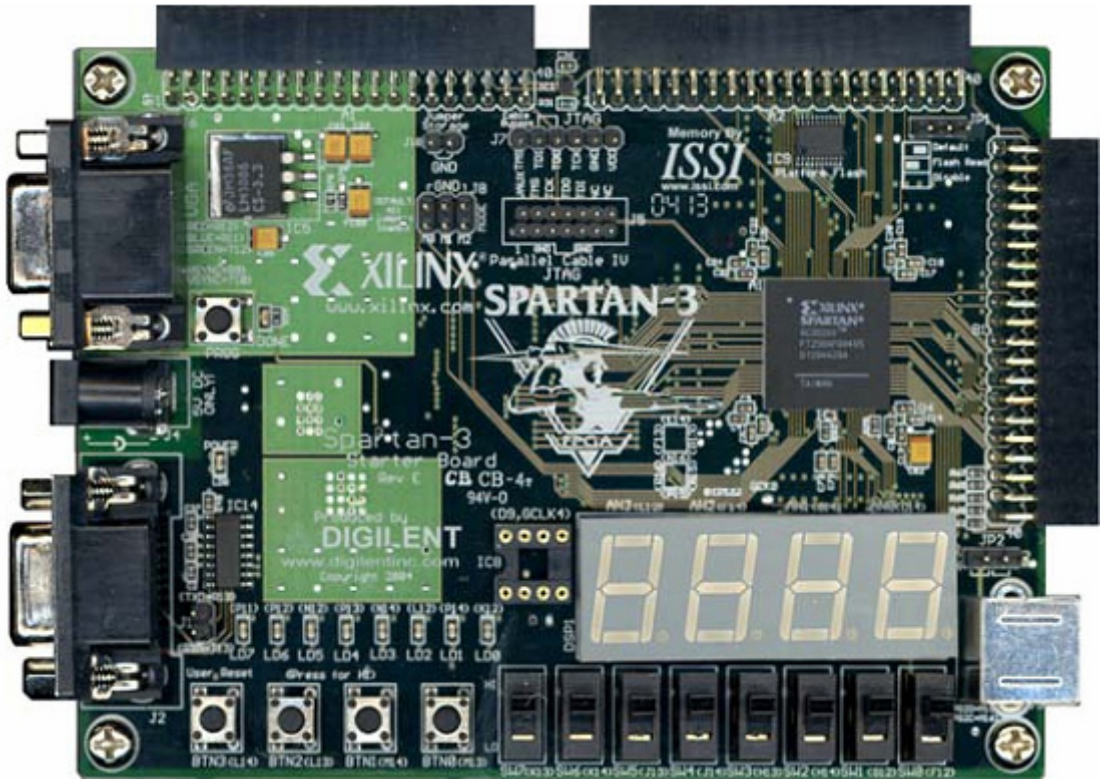


Figura 3.8: *Diligent Spartan-3 Starter Board*

3.2.1 Componentes

Las figuras 3.9 y 3.10 muestran las caras superior e inferior, respectivamente, de la placa, que incluye los siguientes componentes:

- La *FPGA XC3S1000* de *Xilinx* de 1.000.000 de puertas con encapsulado *FT256* (1).
- Una *PROM* (memoria de sólo lectura programable) *Flash XCF02S* de *Xilinx* (2).
- 1 MByte de memoria *RAM* (4).
- Un puerto *VGA* de 8 colores y 3 bits (5).

- Un puerto serie *RS-232* de 9 pines (6).
- Un puerto *PS/2* (9).
- Un display de 4 caracteres de 7 segmentos cada uno (10).
- 8 conmutadores (11).
- 8 diodos *LED*'s (12).
- 4 pulsadores (13).
- Una fuente de reloj de oscilador de cristal de 50 MHz (14).
- Un socket para una fuente de reloj de oscilador de cristal auxiliar (15).
- Jumpers de selección del modo de configuración de la *FPGA* (16).
- Un pulsador para forzar a la *FPGA* a reconfigurarse (17).
- Diodos *LED*'s que indican cuando la *FPGA* está correctamente configurada (18).
- 3 puertos de expansión de 40 pines para extender y mejorar la placa (19) (20) (21).
- Un puerto *JTAG* para la programación de la *FPGA* (22).
- Un puerto *JTAG* para la programación y depuración de la *FPGA* (24).
- Un conector de entrada de alimentación de 5 *VDC* (25).
- Un diodo *LED* indicador de encendido (26).
- Reguladores de 3,3V (27), 2,5V (28) y 1,2V (29).

En los siguientes apartados se profundizará un poco más en los componentes de la placa utilizados en este proyecto.

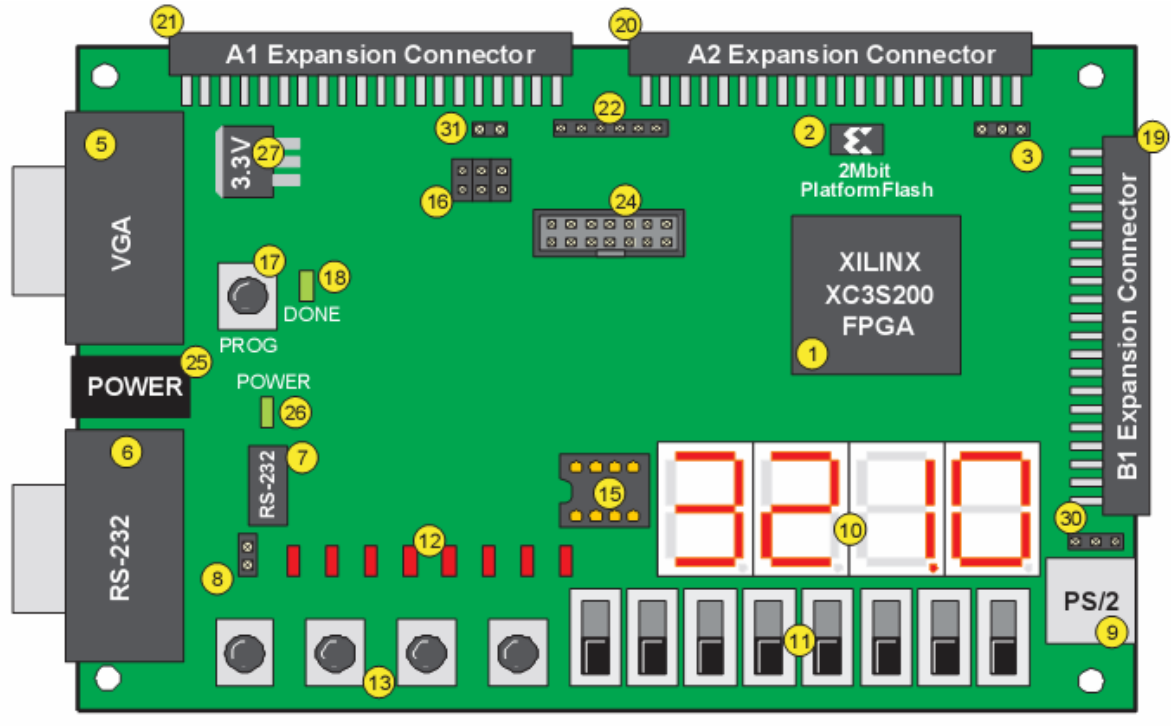


Figura 3.9: Cara superior de la placa

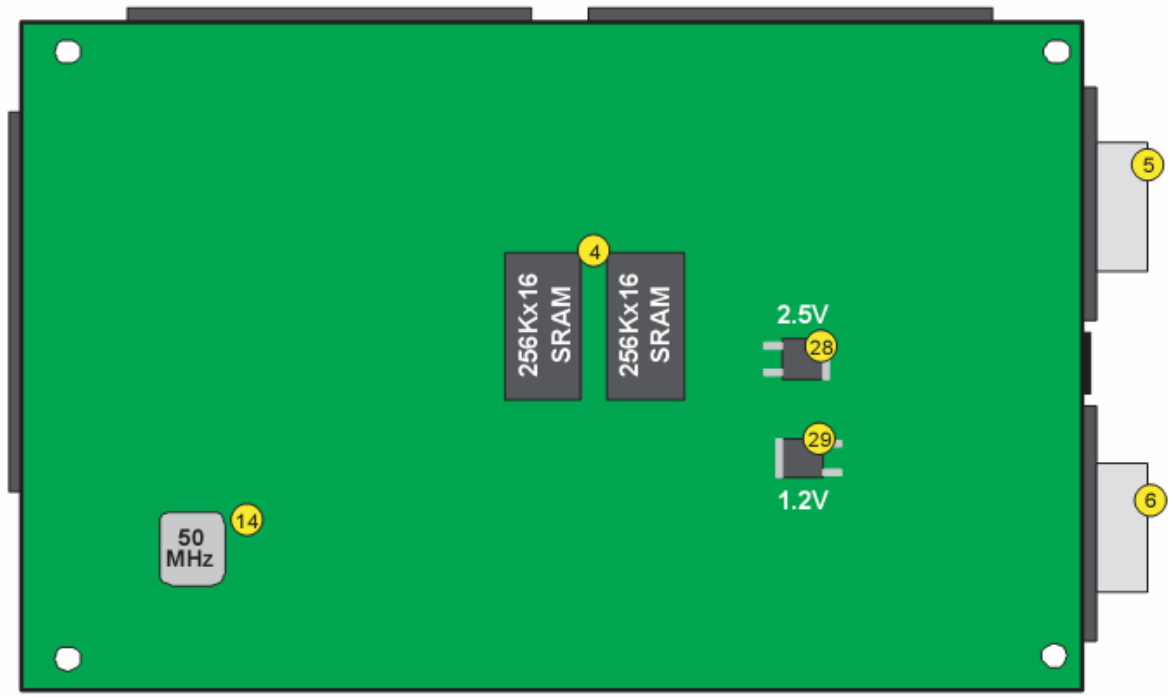


Figura 3.10: Cara inferior de la placa

3.2.2 Pulsadores

La placa dispone de cuatro pulsadores, cada uno de los cuales tiene un pin asociado de la *FPGA*, como se muestra en la tabla 3.2. Al presionar un pulsador genera un nivel lógico alto.

Pulsador	Pin de la *FPGA*
BTN3	L14
BTN2	L13
BTN1	M14
BTN0	M13

Tabla 3.2: Pulsadores y pines asociados

3.2.3 Conmutadores

La placa dispone de ocho pulsadores, cada uno de los cuales tiene un pin asociado de la *FPGA*, como se muestra en la tabla 3.3. Cuando un pulsador se encuentra en la posición *ON* se está generando un nivel lógico alto y cuando está en la posición *OFF* un nivel lógico bajo.

Pulsador	Pin de la *FPGA*
SW7	K13
SW6	K14
SW5	J13
SW4	J14
SW3	H13
SW2	H14
SW1	G12
SW0	F12

Tabla 3.3: Conmutadores y pines asociados

3.2.4 Puerto *VGA*

La placa dispone de un puerto *VGA* y un conector *DB15* para poder conectar directamente la mayoría de monitores de *PC* utilizando un cable estándar. La *FPGA* controla las cinco señales del puerto *VGA*: *R* (rojo), *G* (verde), *B* (azul), *HS* (sincronismo horizontal) y *VS* (sincronismo vertical). En la tabla 3.4 se muestran los pines de la *FPGA* y las señales que controlan. Con las señales *R*, *G* y *B* se pueden generar ocho posibles colores, mostrados en la tabla 3.5.

Pin de la *FPGA*	Señal
R12	*R*
T12	*G*
R11	*B*
R9	*HS*
T10	*VS*

Tabla 3.4: Señales *VGA* y pines asociados

R	*G*	*B*	Color
0	0	0	Negro
0	0	1	Azul
0	1	0	Verde
0	1	1	Cyan
1	0	0	Rojo
1	0	1	Magenta
1	1	0	Amarillo
1	1	1	Blanco

Tabla 3.5: Colores generados con *R*, *G* y *B*

3.2.5 Establecer el modo de configuración de la *FPGA*

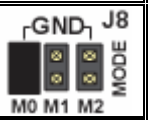
Modo ***<M0:M1:M2>***	***J8***	***JP1***	**Descripción**
Serie Maestro *<0:0:0>*			La *FPGA* se configura desde la memoria *FLASH*
			La *FPGA* intenta configurarse desde una fuente de configuración serie unida al puerto de expansión *A2* o *B1*
Serie Esclavo *<1:1:1>*			La *FPGA* intenta configurarse desde una fuente de configuración serie y de reloj unidas al puerto de expansión *A2* o *B1*
Paralelo Maestro *<1:1:0>*			La *FPGA* intenta configurarse desde una fuente de configuración paralela unida al puerto de expansión *B1*
Paralelo Esclavo *<0:1:1>*			La *FPGA* intenta configuras desde la una fuente de configuración paralela y de reloj unidas al puerto de expansión *B1*
JTAG *<1:0:1>*			La *FPGA* espera ser configurada a través del puerto *JTAG*

Tabla 3.6: Modos de configuración de la *FPGA*

A través del jumper *J8* se establece el modo de configuración de la *FPGA*, tal y como se indica en la tabla 3.6. El jumper *JP1* es necesario cuando se utiliza el modo de configuración serie maestro.

3.2.6 El puerto de programación *JTAG*

El puerto de programación *JTAG J7* permite la configuración de la *FPGA* a través de un cable *JTAG*-paralelo de bajo coste que se conecta a la placa como se indica en la figura 3.11. El otro extremo del cable se conecta al puerto paralelo de un *PC*.

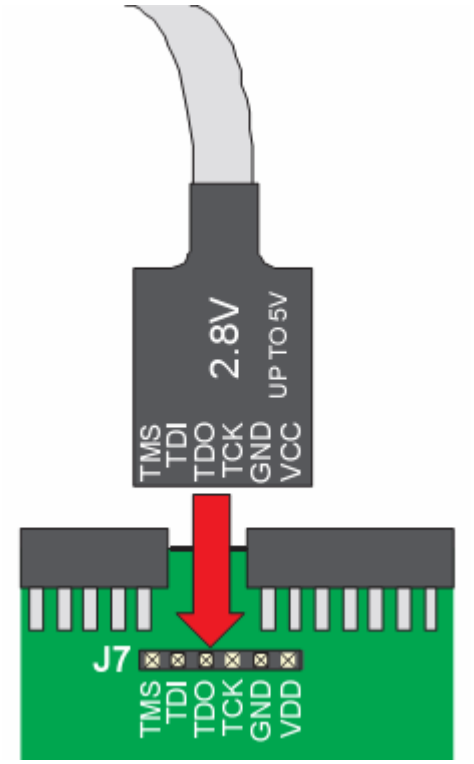


Figura 3.11: Conexión del cable de bajo coste *JTAG*-paralelo al conector *J7* de la placa

Capítulo 4

Metodología de diseño

Más allá de la teoría que se ha visto en los capítulos anteriores sobre las redes digitales, cualquier diseño debe seguir ciertas pautas a lo largo de todo el proceso para poder garantizar un funcionamiento adecuado. No basta con saber qué tareas se deben hacer sino cuál es el orden y el momento más adecuado para realizar cada una de ellas. Conjuntamente, debe existir un control de calidad que asegure que el diseño especificado y la implementación resultante son similares.

Para poder entender y garantizar la características de un dispositivo se va a explicar brevemente a lo largo de este capítulo los pasos que se siguen durante el desarrollo de un proyecto y la documentación que se genera en cada una de sus partes. Posteriormente, se detallan los aspectos más relevantes del lenguaje de programación *Verilog* y la manera más adecuada de utilizarlo para diseñar un dispositivo electrónico como el que se presenta en este proyecto.

4.1 Fases del diseño

A lo largo de un proyecto se definen un conjunto de fases. Cada una de ellas posee un objetivo determinado y está encaminada a obtener un resultado más cercano al producto final. En las siguientes secciones se explica de forma genérica cada una de estas fases y lo que se persigue.

4.1.1 Ruta de diseño

La ruta de diseño es el conjunto de tareas y sus relaciones para alcanzar el objetivo del proyecto. Se pretende aclarar cuáles son los hitos de cada una. En la figura 4.1 se muestran las fases más significativas.

Una vez realizado el estudio teórico y analizada la viabilidad del proyecto se puede empezar el desarrollo de éste.

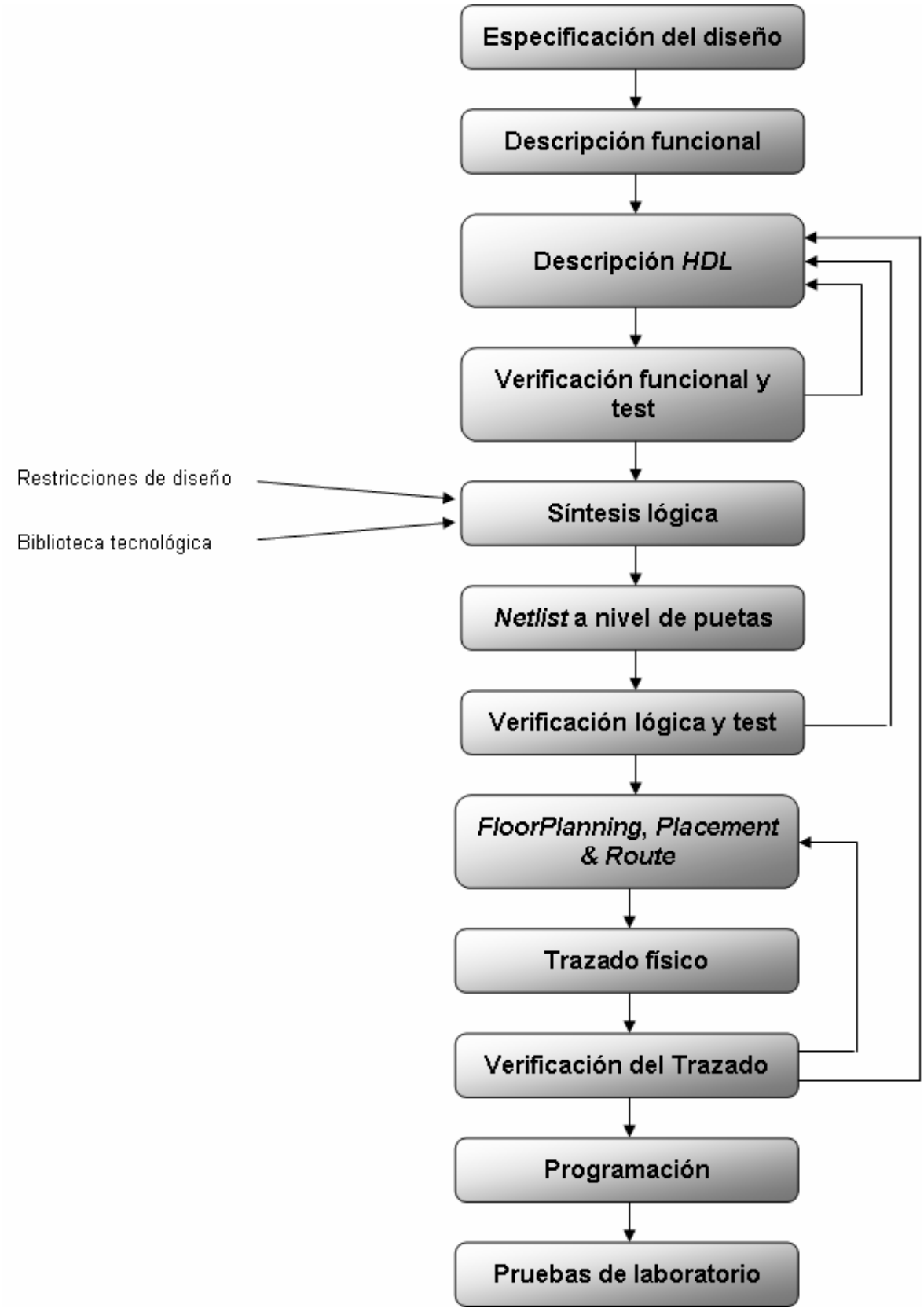


Figura 4.1: Ruta de diseño

4.1.1.1 Especificación y descripción funcional

Una vez que se ha tomado la decisión de desarrollar el proyecto, se deben determinar las partes o bloques que componen el sistema, qué funciones y cómo las realizan.

Se pueden definir en el sistema tres niveles de abstracción:

- Nivel sistema.

  Este enfoque del sistema se caracteriza por aportar información sobre la conectividad, funcionalidad y arquitectura del sistema. Esto significa que se describe un sistema con uno o varios integrados, sus interfaces, arquitectura interna, conectividad a nivel de chip o bloque,... No se aportan datos sobre la implementación ni restricciones temporales. Sólo se pretende entender cómo funciona el sistema. Normalmente, toda esta información se pone por escrito y se denomina especificación de diseño a nivel de arquitectura.

- Nivel cluster

  En todo sistema existe un nivel intermedio entre los bloques individuales y el sistema completo. Normalmente, se agrupan los primeros ya que están relacionados entre ellos para formar un subsistema.

  Las principales ventajas que aportan los cluster al sistema es la posibilidad de definir áreas de trabajo, es decir, los diseñadores disponen de un nivel intermedio entre los bloques y el sistema.

- Nivel bloque.

  El nivel inferior de un sistema son los bloques. Cada uno de ellos posee características independientes que permiten que un diseñador pueda definir en partes el sistema

teniendo en cuenta aspectos concretos pero sin la necesidad de conocer su funcionamiento global.

Evidentemente, es necesario dejar constancia de todas las decisiones tomadas por lo que se genera un documento en el que, típicamente, se incluyen los requerimientos del bloque, modos de funcionamiento, jerarquía (si es necesario), descripción de sus entradas y salidas, así como especificaciones temporales de las señales críticas.

Al documento generado con esta información se le denomina especificación del diseño o simplemente la descripción funcional.

Un denominador común en la especificación de un sistema es la discusión. Todas las decisiones se toman por varias personas. Un bloque no es una entidad independiente sino que forma parte de un sistema mayor. Esto implica que cualquier decisión afecta a los sistemas a los cuales se ha conectado por lo que siempre existe un encargado de definir los aspectos más relevantes y un conjunto de revisores que se encargan de definir aquellos aspectos que se puedan escapar o quedar pendientes por otras razones.

4.1.1.2 Codificación. Descripción *HDL*

Una vez documentados todos los detalles del sistema se codifica en un *HDL* (lenguaje de descripción *hardware*). Los *HDL*'s usados normalmente son el *VHDL* (lenguaje de descripción *hardware* para circuitos integrados de muy alta velocidad) y *Verilog*. En los proyectos universitarios es más popular el *VHDL* y en la industria el *Verilog*.

En concreto, en este proyecto se va a utilizar el lenguaje *Verilog*. Los aspectos más relevantes de este lenguaje se estudiarán en el apartado 4.2.

Además, como el flujo de diseño es iterativo, durante el proceso de codificación puede ser necesario volver a especificar algún detalle del sistema que, por lo general, sólo afectará a un módulo concreto.

4.1.1.3 Verificación y test funcional

Una vez terminada la codificación es necesario verificar el funcionamiento y realizar el test. Al igual que en la especificación, se definen tres niveles de abstracción en la verificación del sistema: módulos, cluster y chip.

Para cada nivel se definen a su vez unos pasos y unos documentos específicos:

- De forma similar y conjunta al documento de especificación se escribe el documento de verificación (*test-plan*).

  Normalmente este documento contiene unas notas del diseñador, un diagrama para aclarar cómo debe ser conectado el código del módulo a testear con un sistema externo que introduce los estímulos, los requerimientos o elementos que deben ser testeados y los casos de test que se programan. Evidentemente, un test puede verificar varios requerimientos y viceversa. Para testear un requerimiento pueden ser necesarios varios casos de test.

- A continuación se procede a programar una biblioteca (*testbench*) que contiene un conjunto de tareas y funciones que facilitan la verificación del diseño.

- La última fase de la verificación consiste en invocar las definiciones anteriores y comprobar los resultados.

Por último, la regla de oro de la verificación: los encargados de la verificación no deben participar en la codificación.

El principal motivo de seguir esta política es conseguir que el ingeniero encargado de verificar el diseño sea imparcial, por lo que generalmente, la verificación es más efectiva. Dentro del marco de este proyecto fin de carrera es inviable seguir esta política debido a que este trabajo se desarrolla de forma individual.

4.1.1.4 Síntesis lógica

La síntesis lógica es el proceso de convertir una descripción de alto nivel del diseño en una representación optimizada a nivel de puertas, dada por una biblioteca de células estándar y unas determinadas restricciones de diseño.

La aparición del *CAD* (diseño asistido por ordenador) ha automatizado el proceso de convertir la descripción de alto nivel a puertas lógicas. Normalmente la herramienta *CAD* iterará el proceso de diseño hasta alcanzar el resultado más acorde con las especificaciones dadas.

Los aspectos más relevantes que aportan las herramientas *CAD* al diseño electrónico son:

- El diseño es menos propenso a los errores humanos ya que los diseños son descritos con un nivel de abstracción alto.

- El diseño se realiza sin tener en cuenta las restricciones de espacio, tiempo y potencia. La síntesis lógica puede convertir el diseño en un *netlist* e intentar que todas las especificaciones se cumplan. En caso contrario, el diseñador modifica el diseño y repite el proceso hasta que el resultado sea óptimo.

- La conversión de alto nivel a nivel de puertas lógicas es rápida y automática.

- Si se modifica alguna restricción no es necesario rediseñar el sistema sino volver a ejecutar la síntesis lógica.

- Por último, aunque no menos importante, la síntesis lógica permite obtener un diseño independiente de la tecnología.

Para obtener los resultados anteriores, las herramientas de síntesis lógica siguen unos pasos que se describen a continuación de forma breve:

Descripción *HDL*

El diseñador describe el diseño a alto nivel usando estructuras *HDL*. El tiempo se invierte en la verificación funcional para asegurar que la descripción es correcta. Después de verificar la funcionalidad, la descripción obtenida es la entrada a la herramienta de síntesis lógica.

Traducción

La descripción es convertida a una representación intermedia no optimizada. La traducción es relativamente simple y usa técnicas similares a las que se estudiarán en el apartado 4.2.1.3. En este punto, la herramienta de síntesis lógica únicamente coloca los recursos internos sin tener en cuenta ninguna restricción.

Representación intermedia no-optimizada

El diseño es representado internamente por la herramienta de síntesis lógica en términos de estructuras de datos interna. Esta representación resulta incomprensible para el usuario.

Optimización lógica

La lógica se optimiza para remover las redundancias. Se utilizan técnicas booleanas independientes de la tecnología. Este paso es muy importante y produce una representación interna optimizada del diseño.

Mapeo y optimización

En este paso, la herramienta de síntesis toma la representación interna optimizada e implementa la representación a nivel de puertas usando las células proporcionadas en la biblioteca tecnológica. A este proceso se le denomina mapeo. Además, la implementación debería satisfacer las restricciones del diseño. En este caso, se realizan algunas

optimizaciones locales para alcanzar los mejores resultados con la tecnología elegida. A esta parte se le denomina optimización tecnológica u optimización dependiente de la tecnología.

La biblioteca tecnológica contiene básicamente el trazado, o en su defecto, las interconexiones, y las características de cada célula. Las características más importantes de cada célula son su función, el área que ocupan, las características temporales y la potencia. Toda esta información es necesaria para llevar a cabo la optimización.

Representación optimizada a nivel de puertas. *Netlist*

Una vez alcanzado este punto el diseñador obtiene la representación del diseño a nivel de puertas. En el caso de que el resultado no cumpla todas las restricciones, se debe modificar el código *HDL* y volver a realizar el proceso.

La figura 4.2 muestra gráficamente el flujo de diseño en la síntesis lógica.

4.1.1.5 Verificación y test lógico

Una vez que se ha obtenido el *netlist* se debe comprobar que existe una correspondencia entre éste y la descripción *HDL*, además de realizar las mismas tareas que la descripción funcional a partir de la cual ha sido sintetizado. Para conseguir tales objetivos, se vuelve a verificar el diseño utilizando los casos de test de la verificación funcional y esperando que los resultados obtenidos sean exactamente los mismos.

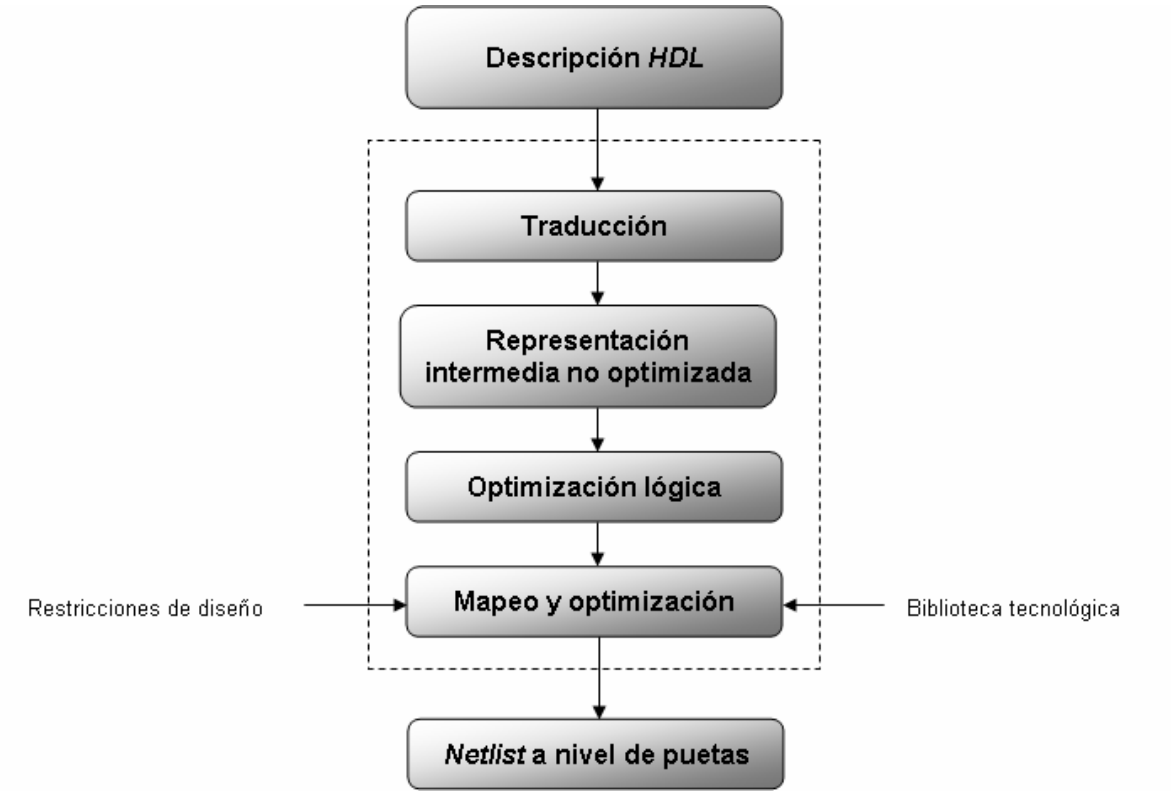


Figura 4.2: Síntesis lógica

4.1.1.6 *Floorplanning* y *Placement & Route*

En este punto se transforma el *netlist* en puertas físicas. Para ello se define la colocación de los elementos desde un nivel de abstracción alto (*FloorPlanning*) y se deja que la herramienta procese la información y genere el trazado colocando las puertas asociadas a cada módulo y conectándolas (*Placement & Route*).

4.1.1.7 Trazado físico

Normalmente el trazado obtenido de la etapa anterior se corresponde con partes digitales del diseño. Para completarlo es necesario añadir los *pads*, etapas recuperadoras del reloj, árboles de reloj, distribución de la alimentación, puertos de entrada/salida,. . . y obtener el trazado completo.

4.1.1.8 Verificación del trazado

Antes de la fabricación del dispositivo es necesario volver a verificar el trazado obtenido. En este caso, se buscan errores propios de la fase anterior y de las fases de *Placement & Route*.

4.1.1.9 Programación y pruebas de laboratorio

Por último, se fabrica el dispositivo y se realizan las pruebas de laboratorio para asegurar la correcta funcionalidad del sistema.

4.2 Importancia de los lenguajes *HDL. Verilog*

El diseño digital de circuitos integrados ha evolucionado vertiginosamente en los últimos años. La escala de integración de los dispositivos ha aumentando conjuntamente con los avances de la tecnología y la sociedad solicita nuevos servicios que requieren a su vez de dispositivos más complejos.

El aumento progresivo de la complejidad de los dispositivos y el ritmo con el cual se producen los cambios en la sociedad actual requieren que los nuevos servicios y los dispositivos que les dan soporte se adapten rápidamente. Para poder disponer de toda la electrónica necesaria entra en juego el *CAD*. Estas herramientas se caracterizan por permitir diseños más complejos en un menor tiempo y, lo que es más importante, verificados.

La combinación de las nuevas herramientas *software* con los lenguajes de descripción *hardware* presenta un conjunto de ventajas que se describen a continuación:

- Se puede describir un diseño usando un nivel de abstracción muy alto. Los diseñadores pueden escribir su descripción sin elegir una tecnología de fabricación específica. Las herramientas de síntesis lógica pueden convertir automáticamente el diseño a cualquier tecnología. Si aparece una nueva tecnología, los diseñadores no necesitan rediseñar el circuito sino simplemente volver a introducir la descripción *HDL* en la herramienta de síntesis lógica y crear una nueva *netlist*. La herramienta de síntesis lógica optimizará el circuito en área y retardo para la nueva tecnología.

- Describiendo los diseños en *HDL*, la verificación funcional del diseño puede hacerse rápidamente dentro del flujo de diseño. Los diseñadores pueden optimizar y modificar la descripción *HDL* hasta que cumpla la funcionalidad deseada. La mayor parte de los errores son eliminados en este momento haciendo que el tiempo dedicado al diseño se reduzca significativamente y que la probabilidad de encontrar un error posteriormente en el *netlist* o en el trazado se minimice.

- El diseño con *HDL* es análogo a la programación. Una descripción con comentarios es el camino más fácil para desarrollar y depurar los circuitos. También proporciona una representación concisa del diseño comparada con el esquemático a nivel de puertas. Estos últimos son casi incomprensibles para muchos diseños complejos.

Estas son las principales razones que han propiciado el crecimiento de los lenguajes *HDL*. En el caso concreto del lenguaje *Verilog*, éste ha evolucionado como un lenguaje de descripción hardware estándar que ofrece las siguientes características:

- Es un lenguaje de descripción hardware de propósito general que es fácil de aprender y usar. Su sintaxis es similar al lenguaje de programación *C* por lo que a muchos programadores les resulta sencillo aprender *Verilog*.

- Permite utilizar diferentes niveles de abstracción en un mismo modelo. De esta manera, un diseñador puede definir un modelo hardware en términos de conmutación, puertas lógicas, *RTL* (lógica de transferencia de registro), o comportamiento. Además, el

diseñador sólo necesita aprender un lenguaje para la aplicación de los estímulos y el diseño.

- Casi todas las herramientas de síntesis lógica lo soportan.

- Los fabricantes proporcionan su biblioteca para la simulación de los diseños en él.

- El interfaz del lenguaje de programación es una característica que permite a los usuarios escribir código en lenguaje *C* que interactúe con las estructuras de datos internas de *Verilog*. Es decir, los diseñadores pueden personalizar un simulador *Verilog HDL* en función de sus necesidades.

Quizás, el aspecto más interesante es aclarar el concepto asociado a los niveles de abstracción. Esta idea permite describir el sistema e interconectar sus módulos considerando cada módulo de manera independiente. Esto se traduce en la posibilidad de combinar registros, puertas lógicas,. . . A continuación se enumeran cada una de sus facetas:

Modelo a nivel de comportamiento

Este es el nivel más alto proporcionado por *Verilog HDL*. Se puede codificar un módulo en términos algorítmicos sin interesarse en detalles relativos a su implementación. El diseño en este nivel es muy similar a la programación en lenguaje *C*.

Modelo a nivel de flujo de datos

En este nivel el módulo es diseñado especificando el flujo de datos. El diseñador debe conocer el modo en que se transfieren los datos entre los registros y cómo se procesan en el diseño.

Modelo a nivel de puertas

El módulo se diseña con puertas lógicas y las interconexiones entre ellas. Diseñar en este nivel es muy similar a describir el diseño en términos de un diagrama lógico a nivel de puertas.

Modelo a nivel de conmutación

Este es el nivel más bajo de abstracción proporcionado por *Verilog*. Un módulo puede ser implementado en términos de conmutadores, nodos con capacidad de almacenamiento de carga, y las interconexiones entre ellos. El diseño a este nivel requiere conocimientos sobre los detalles de la conmutación.

Verilog permite al diseñador mezclar y/o unir los cuatro niveles de abstracción en un único diseño. En la comunidad de ingenieros dedicados al diseño electrónico el término *RTL* es usado frecuentemente en las descripciones que usan una combinación de construcciones de comportamiento y flujo de datos y que son aceptadas por las herramientas de síntesis lógica.

En la siguiente sección se explicará de forma breve y práctica el estilo más adecuado de codificación *RTL* y las construcciones más comunes, así como su equivalente *hardware*. En las referencias [7], [8], [9] y [10] se puede profundizar más sobre estas cuestiones.

4.2.1 Síntesis del lenguaje *Verilog*

Usualmente, los diseñadores se enfrentan a un problema cada vez que codifican en *Verilog RTL*. Simplemente se plantean qué ocurre con el código cuando se procesa posteriormente. La cuestión es realmente sencilla de plantear y compleja de resolver y entender ya que intervienen muchos factores.

En esta sección se pretende aconsejar cómo se debe describir un sistema en *Verilog* para evitar problemas indeseados, explicar los aspectos más típicos e interpretar las equivalencias más sencillas entre el código y la primera fase de la síntesis lógica.

4.2.1.1 Estilo del código *RTL*

El estilo de codificación del lenguaje *Verilog* afecta al *netlist* final generado en la fase de síntesis lógica. Esta fase puede generar un *netlist* eficiente o ineficiente dependiendo del estilo de la descripción *RTL*. De aquí que los diseñadores deban ser conscientes de las técnicas existentes para escribir descripciones eficientes. En los siguientes apartados se cubren los aspectos más importantes sobre este tema.

Uso de nombres con significado para las señales y variables

Los nombres de las señales y las variables deben tener significado para que el código sea más legible.

Evitar mezclar *flip-flops* activos por flanco de subida y bajada

La combinación de *flip-flops* activos por flanco de subida y bajada puede producir que la herramienta de síntesis lógica introduzca inversores y buffers en el árbol de reloj. Estos elementos pueden producir *skew* en la distribución del reloj.

Uso de bloques elementales frente a asignaciones continuas

Las declaraciones de asignaciones continuas son un camino muy conciso de representar la funcionalidad. Sin embargo, la estructura lógica final no es necesariamente simétrica. Los bloques elementales dan lugar a diseños simétricos y la herramienta de síntesis lógica es capaz de optimizar los módulos más pequeños de manera más efectiva. Sin embargo, no son el camino más claro para describir el diseño. Además, empeoran el rendimiento del simulador.

Uso de multiplexores frente a las sentencias *if-else* o *case*

Las instrucciones *if-else* y *case* son sintetizadas frecuentemente con multiplexores en *hardware*. Si es necesario un diseño estructurado es mejor implementar directamente un multiplexor antes que utilizar las instrucciones *if-else* y *case* ya que la síntesis puede generar lógica aleatoria. Un multiplexor proporciona mejor control y una síntesis más veloz, pero tiene la desventaja de depender de la tecnología y dar lugar a una descripción *RTL* más larga.

Uso de paréntesis para optimizar la estructura lógica

Un diseñador puede controlar la estructura final de la lógica usando paréntesis para agruparla. El uso de paréntesis también mejora la legibilidad de la descripción *Verilog*.

Uso de operadores aritméticos (*, /, %) frente a bloques diseñados específicamente

La multiplicación, la división y el módulo son operadores muy exigentes en términos de área y lógica. Sin embargo, estos operadores pueden ser usados para implementar la funcionalidad deseada de manera concisa e independiente de la tecnología. Sin embargo, los bloques diseñados a medida por el usuario requieren más tiempo en el diseño y pueden depender de la tecnología a utilizada.

Asignaciones múltiples a la misma variable

Las asignaciones múltiples a la misma variable pueden dar lugar a lógica no deseada. Normalmente, sólo se tendrá en cuenta la última asignación. En cualquier caso, es mejor evitar estas asignaciones.

Definición de *if-else* y *case* explícita

Las salidas para todas las posibles condiciones deben ser especificadas para las instrucciones *if-else* y *case*. De otra manera, la herramienta de síntesis puede colocar *latches* activos por nivel en vez de multiplexores.

4.2.1.2 Asignaciones bloqueantes y no bloqueantes

Las asignaciones actualizan los valores de las variables. El valor tomado por una variable permanece sin cambio hasta que otra asignación cambia la variable a un valor diferente. Hay dos tipos de asignaciones:

Asignación bloqueante

Las asignaciones bloqueantes son ejecutadas en el orden en el cual se especifican dentro de un bloque secuencial. Una asignación bloqueante no detiene la ejecución de las sentencias de los restantes bloques que se ejecutan en paralelo. Se usa el operador = para las asignaciones bloqueantes.

Asignación no bloqueante

Las asignaciones no bloqueantes permiten planificar las asignaciones sin detener la ejecución de las sentencias en un bloque secuencial. Se usa el operador <= para especificar las asignaciones no bloqueantes. Este operador tiene el mismo símbolo que el operador de relación menor o igual que. El operador <= se interpreta como un operador de relación en una expresión y como una asignación en el contexto de una asignación no bloqueante.

Las recomendaciones más importantes sobre el tipo de asignación a usar y el criterio a seguir en cada caso son:

1. Cuando se modela lógica secuencial se usan asignaciones no bloqueantes.

2. Cuando se modela un *latch* se usan asignaciones no bloqueantes.

3. Cuando se modela lógica combinacional dentro de un bloque *always* se usan asignaciones bloqueantes.

4. Cuando se modela lógica combinacional y secuencial dentro de un bloque *always* se usan asignaciones no bloqueantes.

5. No se deben combinar asignaciones bloqueantes y no bloqueantes dentro de un bloque *always*.

6. No se deben hacer asignaciones a la misma variable desde más de un bloque *always*.

7. Usar *$strobe* para visualizar valores que hayan sido asignados usando asignaciones no bloqueantes.

8. No se deben hacer asignaciones usando *#0 delays*.

Los problemas que se evitan siguiendo estos consejos son:

- Las carreras ocurren cuando dos o más sentencias que son ejecutadas en la misma unidad de tiempo pueden proporcionar diferentes resultados. Normalmente, si se modela adecuadamente la lógica combinacional y secuencial se evita este problema.

- Muchas veces el estilo *RTL* empleado conlleva que el diseño se pueda simular correctamente pero no así el *netlist* que se obtiene con la síntesis lógica. En otros casos se garantiza una correcta simulación del *netlist* pero no del código *RTL*. En ambos casos, se entiende de manera inequívoca que el estilo del código es de mala calidad.

- En muchas ocasiones las simulaciones de un sistema son correctas pero el *hardware* que genera el código *RTL* no modela el sistema que se desea.

4.2.1.3 Interpretación de algunas construcciones en *Verilog*

En la primera fase de la síntesis lógica se produce la traducción del código *RTL* a estructuras lógicas. En este apartado se pretenden enlazar los conceptos de programación con los conceptos de electrónica.

La sentencia *assign*

La sentencia *assign* es la construcción más básica usada para describir lógica combinacional con código *RTL*. A continuación se muestra un ejemplo:

```
assign out = (in1 & in2) | in3;
```

Esta puede ser una representación a nivel de puertas de esta sentencia:

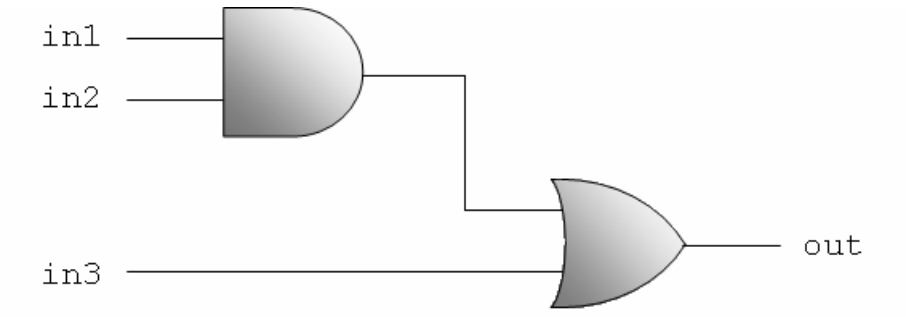


Figura 4.3: La sentencia *assign*

En caso de trabajar con vectores, se genera la lógica necesaria para cada bit de forma paralela e idéntica.

La sentencia *if-else*

Una sentencia *if-else* se traduce en un multiplexor donde la señal de control es la señal o variable en la cláusula *if*.

```
if(s)
      out = in1;
else
      out = in2;
```

También se interpreta como un multiplexor el operador condicional.

```
assign out = (s) ? in1 : in2;
```

En general, múltiples sentencias *if-else* anidadas no tienen porqué sintetizar un multiplexor mayor sino varios pequeños y anidados. Esto se debe tener en cuenta al escribir el código *RTL*.

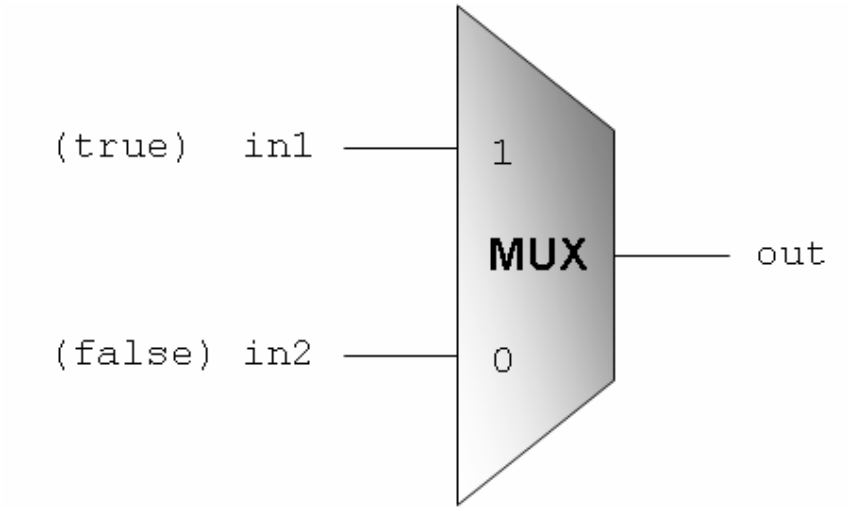


Figura 4.4: La sentencia *if-else*

La sentencia *if*

Cuando la sentencia *if* está incompleta el multiplexor se convierte en un *latch*. Cuando la expresión es verdadera se activa el *latch*. Si es falsa, no ocurre nada.

```
if(s)
      out = in;
```

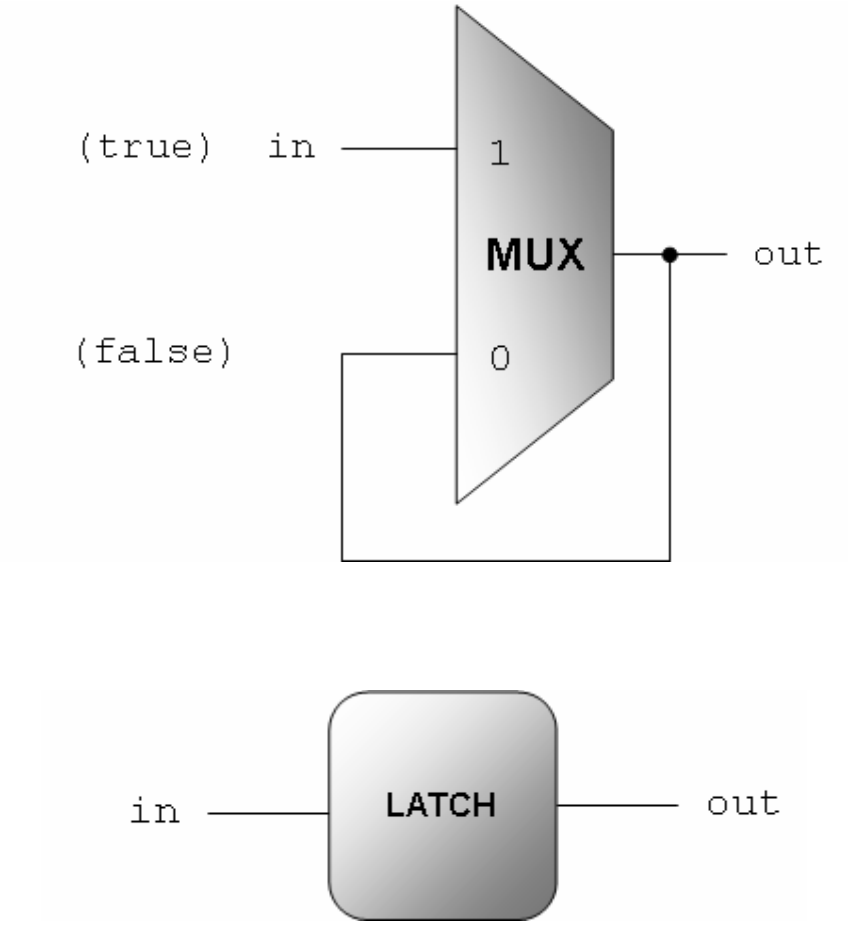


Figura 4.5: La sentencia *if*

La sentencia *case*

La sentencia *case* también puede ser usada para sintetizar multiplexores. Los multiplexores anteriores se pueden deducir de la siguiente descripción que usa la sentencia *case*.

```
case(s)
      1 'b0 : out = in1;
      1 'b1 : out = in2;
endcase
```

El uso de la sentencia *case* con numerosas opciones puede dar lugar a multiplexores grandes.

Bucles

Normalmente, cuando se diseña un dispositivo usando el estilo *RTL* se evita el uso de bucles. La mayor parte de las herramientas de síntesis aceptan la sentencia *for* porque tiene limitado el número de iteraciones. En este caso, interpreta la sentencia y repite la lógica encadenándola, por lo que acumula el retardo, lo cual ocasiona muchas veces que no se cumplan las especificaciones temporales, y generando una gran cantidad de lógica ya que sintetiza cada iteración por separado.

La sentencia *always*

La sentencia *always* puede ser usada para sintetizar lógica combinacional y secuencial. Para la lógica secuencial, la sentencia *always* debe ser controlada por el cambio en una señal de reloj.

Para la lógica combinacional, la sentencia *always* debe ser disparada por una señal distinta del reloj, reset o preset.

La sentencia *function*

Las funciones sintetizan bloques combinacionales con una variable de salida. La salida puede ser escalar o vectorial. Puede ser llamada cuantas veces quiera el diseñador sabiendo que en cada ocasión se genera lógica en paralelo.

Una vez visto el estándar *SDH* en el capítulo anterior y explicada la metodología en éste, sólo queda presentar el diseño.

4.3 Metodología de diseño empleada en este proyecto

Para la realización del presente proyecto se han empleado los siguientes medios:

- Toda la bibliografía incluida en la parte V de la memoria.
- Recursos hardware:
  - Estación de trabajo y servidor *SUN Microsystems*.
  - Placa de prototipado *Digilent Spartan 3 System Board*, que incluye la *FPGA Spartan 3* de un millón de puertas de *Xilinx*, así como, entre otros recursos, un puerto *VGA*, un puerto *JTAG*, 8 conmutadores y 4 pulsadores.
  - Un monitor *VGA*.
- Recursos software:
  - Editor de textos *Emacs 21.3.1*.
  - Compilador y simulador *Verilog-XL 05.10.003-s*.
  - Visor de formas de onda *SimVision 05.10-s12*.
  - Herramienta de síntesis *Xilinx ISE* (entorno *software* integrado) *8.1.03i*.
  - Herramienta de configuración *iMPACT 6.3i*.
  - Paquete *Office* de *Microsoft*.

A continuación se detallan los pasos seguidos para el desarrollo del proyecto:

1. Estudios previos

Aquí se incluye el estudio de las redes de comunicación digitales, del estándar *SDH*, del lenguaje *Verilog* y de las características de los dispositivos lógicos programables, así como del manejo de las distintas herramientas *software* empleadas en este proyecto y de los manuales de la *FPGA* y la placa de prototipado.

Para estos estudios se ha empleado la bibliografía incluida en la parte V de la memoria.

2. Especificaciones del sistema

Se determinan las especificaciones generales del sistema y se divide el mismo en subsistemas. Luego se determinan las funciones de cada subsistema y cómo las realizan.

3. Descripción funcional de los subsistemas y del sistema completo

Para la descripción de cada subsistema, así como del sistema completo, se ha utilizado el lenguaje *Verilog* y el editor de textos *Emacs* (figura 4.6).

Para cada subsistema se crea un módulo en *Verilog* y cada módulo se almacena en un fichero distinto, con el mismo nombre que el módulo y con extensión *.v*. La estructura de cada uno es la siguiente:

- Declaración del módulo: Se declara el módulo con su lista de puertos.

- Interfaz: Se declaran cada uno de los puertos de la lista como entrada, salida o bidireccional.

Figura 4.6: Editor de textos *Emacs*

- Registros internos: Se declaran los registros internos del módulo.

- Lógica combinacional: Aquí se incluyen las asignaciones bloqueantes para la posible lógica combinacional que pueda contener el módulo.

- Conexiones internas: Si el subsistema consta a su vez de otros subsistemas, éstos podrán estar conectados entre sí, por lo que habrá que declarar variables de tipo nodo.

- Módulos de nivel inferior: Se incluyen las instancias de los módulos de los posibles subsistemas de los que consta.

- Bloques: Aquí se incluyen todos los bloques de los que consta el módulo, formados por asignaciones no bloqueantes que se ejecutarán en cada flanco de subida de la señal de reloj o de la señal de reset del módulo.

4. Verificación funcional

Para verificar cada subsistema, se diseña un módulo que genere las señales de estímulo necesarias[2], y que incluye los siguientes bloques:

```
initial
  begin
     $shm_open("tb.shm");
     $shm_probe("AS");
  end

initial
  begin
     #0 clk = 1'b1;
     forever #10 clk = ~clk;
  end
```

[2] Ver apartado 6.2 del capítulo 6: *Verificación a nivel de módulo*

```
initial
  begin
     #0  reset = 1'b1;

     #15 reset = 1'b0;
  end
```

Con el primero se almacena el resultado de la simulación en una base de datos, donde se incluyen todas las señales de todos los módulos que van a formar parte de la simulación. El segundo genera una señal periódica de reloj y el tercero activa la señal de reset al principio de la simulación para que todos los módulos se reinicien.

Para llevar a cabo la simulación se ha utilizado el *Verilog-XL*, y el resultado de la misma, la base de datos, se ha visualizado en el *Simvision*, donde observando la forma de onda de cada señal, se puede comprobar el correcto funcionamiento de cada módulo.

5. Síntesis e implementación del sistema

La herramienta utilizada para la síntesis e implementación del sistema ha sido el *ISE* de *Xilinx*, que dispone de un navegador desde donde se puede organizar los archivos del diseño y ejecutar distintos procesos sobre los mismos.

Lo primero que hay que hacer es crear un proyecto con el comando *New Projyect…*que ejecutará un asistente que pedirá, entre otras cosas, el dispositivo donde se va a implementar el diseño, en este caso la *Spartan-3*, y sus características, así como los archivos que forman parte del diseño, que contienen la descripción en *Verilog* del sistema y los subsistemas. Para asignar un pin de la *FPGA* a una señal hay que añadir la siguiente directiva:

```
// synthesis attribute LOC of clk_in is T9
```

En este ejemplo la señal *clk_in* se asigna al pin *T9*.

Una vez creado el proyecto el navegador tiene la apariencia mostrada en la figura 4.7. La ventana principal consta a su vez de las siguientes ventanas:

1. Barra de herramientas: Permite el acceso a los comandos más usados.

2. Fuentes: Aquí se muestran todos los archivos del diseño que forman parte del proyecto.

3. Procesos: Desde aquí se pueden ejecutar los procesos deseados sobre el elemento seleccionado en la ventana de fuentes.

4. Espacio de trabajo: Todos los archivos que se abran aparecen aquí.

5. Transcripción: Aquí se muestran los mensajes de salida de los procesos que han sido ejecutados.

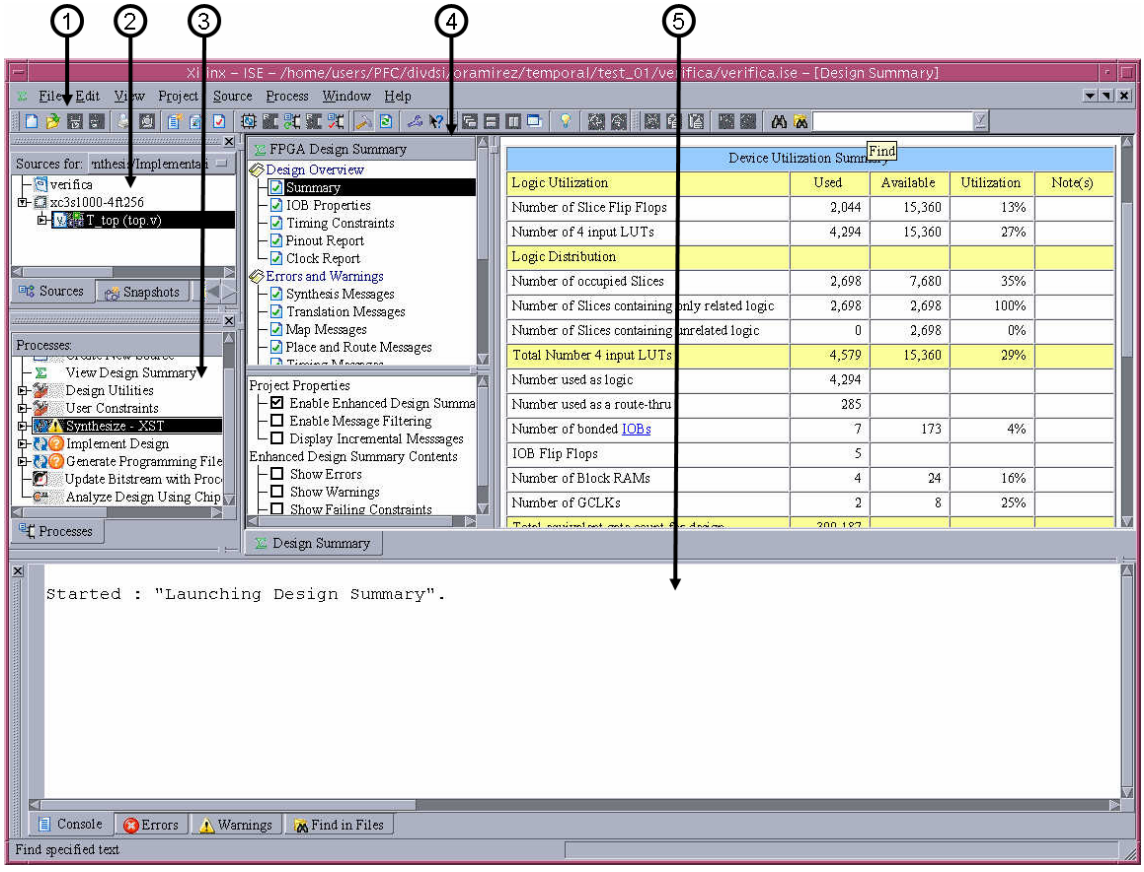


Figura 4.7: Ventana principal del navegador de proyectos del *ISE*

Para realizar la síntesis del sistema hay que hacer clic con el botón derecho del ratón sobre el proceso *Synthesize* en la ventana de procesos, teniendo seleccionado el módulo de nivel superior en la ventana de fuentes.

Terminado el proceso de síntesis se procede a la implementación del diseño, que incluye la traducción, el mapeo y el *Placement & Route*.

Una vez finalizada la implementación, se generará un fichero *bitstream*, con extensión *.bit*, necesario para la configuración de la *FPGA*.

6. Configuración de la *FPGA*

Para la configuración de la *FPGA* se ha utilizado el *iMPACT*. La placa de prototipado se conecta a través de su puerto *JTAG* al puerto paralelo de la estación de trabajo. Al iniciar la herramienta se ejecuta un asistente en el que se le especifica el modo de configuración de la *FPGA*, en este caso el *Boundary-Scan*, y la detección automática de los dispositivos conectados a la cadena *Boundary-Scan*. También se le indicará el fichero *bitstream* de configuración creado en el paso anterior.

Una vez finalizado el asistente, se puede observar como la herramienta ha detectado dos dispositivos (figura 4.8): la *FPGA* y la memoria *PROM* incluida en la placa de prototipado. Ya sólo queda programar la *FPGA* haciendo clic con el botón derecho del ratón sobre la misma en el *iMPACT* y seleccionando el comando *programar* del menú emergente.

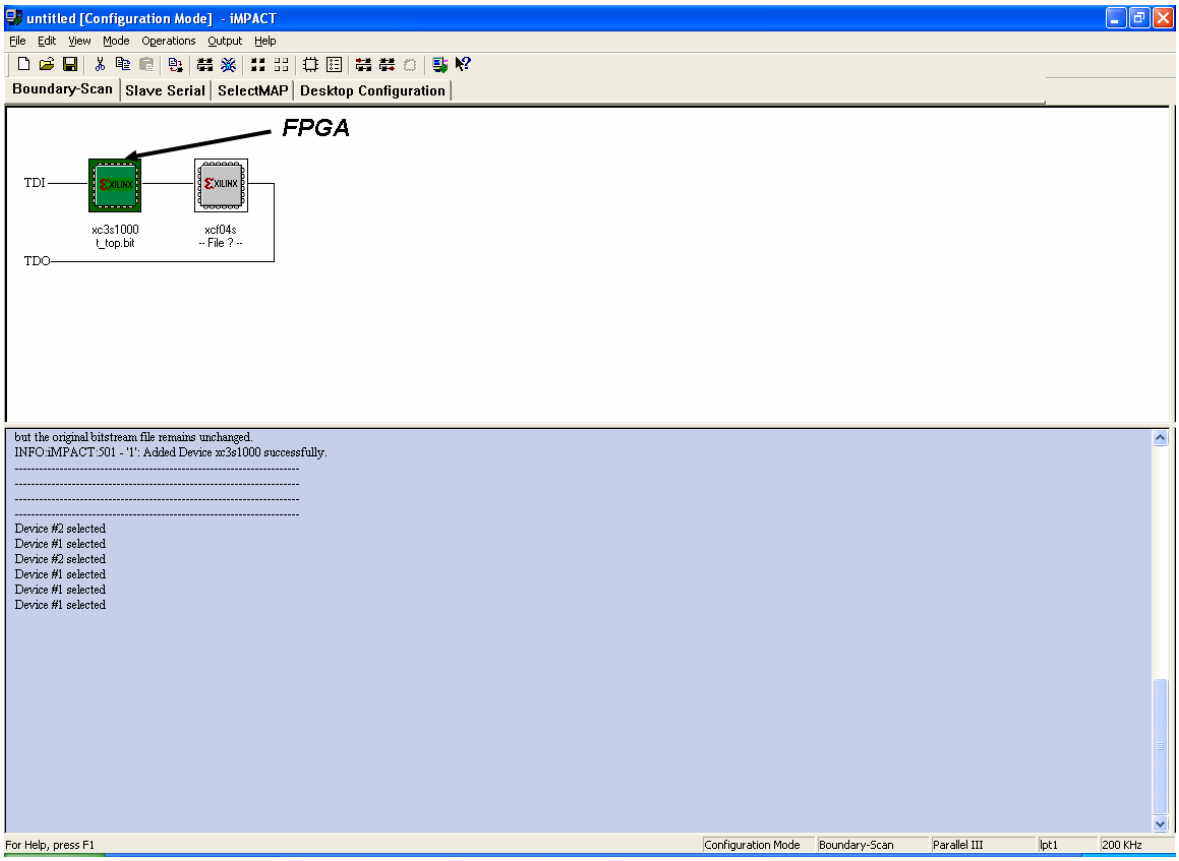


Figura 4.8: Herramienta *iMPACT*

7. Verificación *hardware*

Con la *FPGA* ya programada, hay que verificar el correcto funcionamiento del diseño, conectando el monitor *VGA* al puerto *VGA* de la placa de prototipado (figura 4.9).

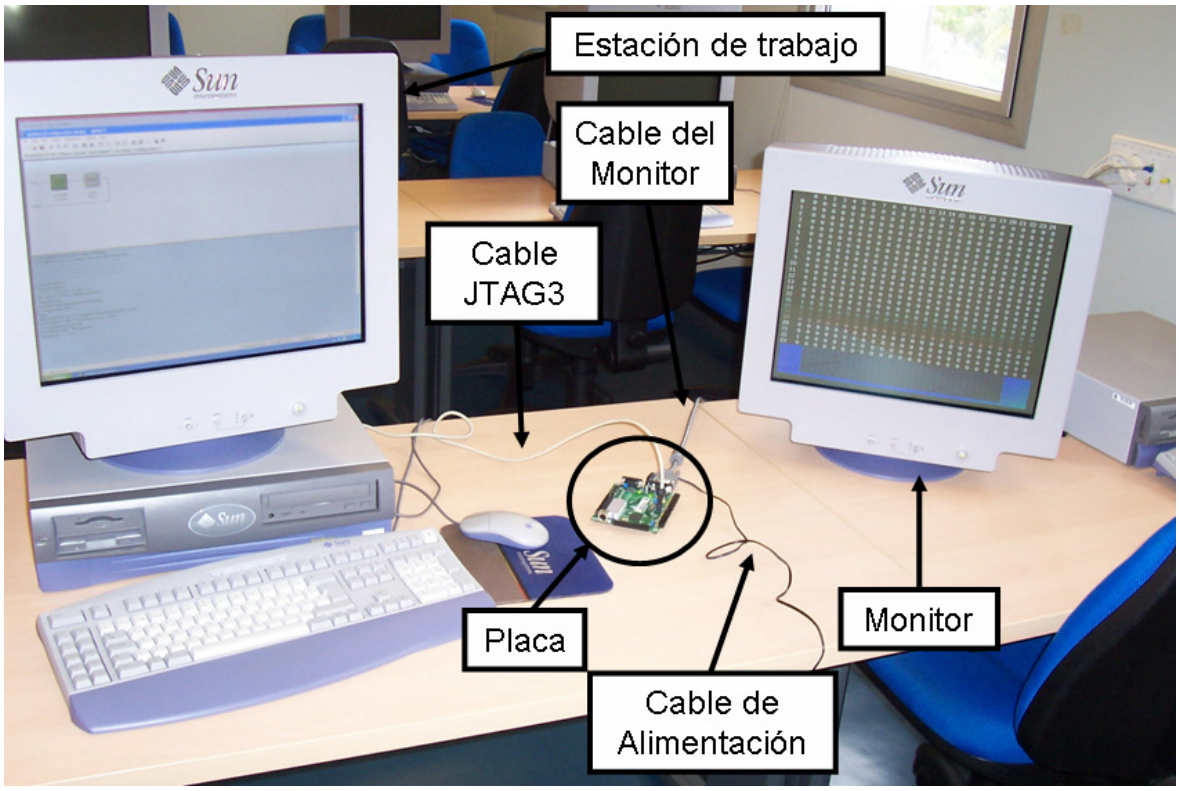


Figura 4.9: Elementos utilizados en la verificación hardware

PARTE III
DESARROLLO DEL PROYECTO

Capítulo 5

Especificaciones

En este capítulo se describen las especificaciones del receptor de tramas *SDH STM-16* que transportan *VC-4-16c* concatenados, objeto del presente proyecto. Para cada módulo se ve su interfaz, sus funciones y su funcionamiento.

En el último apartado se da una relación de las señales y alarmas descritas a lo largo del capítulo.

5.1 Descripción General

La función principal del receptor de tramas *SDH* es la de extraer el *C-4-16c* que transporta cada trama que va recibiendo a su entrada. Además, debe capturar los octetos de las distintas taras y activar un cierto número de señales y alarmas.

Para lograr esto tiene que seguir un procedimiento que podemos resumir en tres pasos:

1º *Sincronizar*: se trata de identificar la primera palabra de cada trama entrante. De esta forma, el receptor podrá determinar cuándo los datos presentes a su entrada son octetos pertenecientes a la *RSOH* o a la *MSOH*, cuándo son punteros o cuándo pertenecen a la carga útil.

2º *Interpretar el puntero*: este segundo paso consiste en interpretar los octetos *H1* y *H2*, correspondientes a los punteros, para así poder conocer el comienzo del *VC*.

3º *Identificar el comienzo del VC*: en este último paso y una vez interpretado los punteros, se determina cuando la palabra presente a la entrada del receptor es el comienzo del *VC*. De esta forma, al igual que en el paso 1, se podrá determinar cuando los datos presentes a la entrada forman parte, en este caso, de la *POH* o cuándo son datos.

Las tramas se procesarán en paralelo en grupos de 32 bits o, lo que es lo mismo, de 4 octetos, denominándose palabra a dicho grupo. Con esto se consigue dividir la frecuencia de reloj a 32 veces la de transmisión. Como la velocidad de transmisión es de 2.488,32 Gbps y la transmisión se realiza bit a bit sería necesaria una frecuencia de reloj de 2.488,32 MHz, pero al dividirla entre 32 se queda en 77,76 MHz.

En la figura 5.1 y en la tabla 5.1 se muestran la interfaz del receptor de tramas *SDH*. Las entradas y salidas se pueden agrupar en 5 grupos:

- Datos de entrada: a través de *din[31:0]* entran las tramas recibidas al receptor.

- Datos de salida: a través de *dout[31:0]* salen las tramas ya desaleatorizadas del receptor. La señal *act_datos* indica, con un "1", cuando los datos presentes en *dout[31:0]* forman parte del contenedor, y *SOF* se activa al comienzo de cada trama.

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
reset	E	Reset
din[31:0]	E	Datos de entrada
dout[31:0]	S	Datos de salida
act_datos	S	Señalización de datos
SOF	S	Comienzo de trama
LOS	E	Pérdida de señal
BER	E	Degradación de la señal
sinc	S	Señal de sincronismo
LOF	S	Pérdida de trama
OOF	S	Fuera de trama
MS_AIS	S	Alarma de sección de multiplexación
AU_AIS	S	Alarma de punteros
VC_AIS	S	Alarma de contenedor virtual
int	S	Interrupción
ADR_I[4:0]	E	Vector de dirección del protocolo *Wishbone*
DAT_I[7:0]	E	Vector de datos de entrada del protocolo *Wishbone*
DAT_O[7:0]	S	Vector de datos de salida del protocolo *Wishbone*
CYC_I	E	Señal de ciclo de bus válido del protocolo *Wishbone*
STB_I	E	Señal de selección del protocolo *Wishbone*
WE_I	E	Señal escritura del protocolo *Wishbone*
ACK_O	S	Señal de fin de un ciclo de bus del protocolo *Wishbone*
signal2Rx	E	Protocolo *Handshake*
signal2Tx	S	Protocolo *Handshake*
output_frame_B2[7:0]	S	Errores detectados por el octeto *B2*

Tabla 5.1: Interfaz del receptor

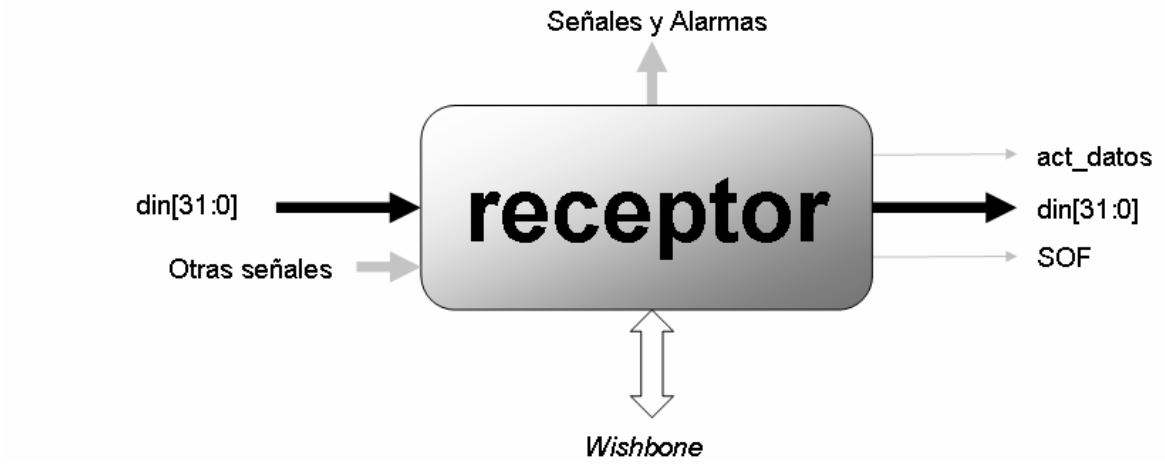


Figura 5.1: Interfaz del receptor

- Señales y alarmas: compreden la señal *sinc* y las alarmas *LOF*, *OOF*, *MS-AIS*, *AU-AIS* y *VC-AIS*.

- Protocolo *Wishbone*: aquí se incluyen todas las señales del protocolo *Wishbone*, a través del cual, se van a entregar los octetos de las taras capturados y el resto de señales de error y alarmas.

- Otras señales de entrada: como por ejemplo la señal de reloj *clk_Rx*, y las señales *reset*, *LOS* y *BER*.

La figura 5.2 muestra el diagram de bloques del receptor. El funcionamiento general es el siguiente: el módulo *entrada* sincroniza el sistema con las tramas entrantes y pone en marcha un contador de palabra que permite identificar cada palabra a la entrada. A partir de ese contador, el módulo *sec_Rx* activa todas las señales de control de los bloques del receptor en el instante correspondiente. Los módulos *RSOH_Rx*, *MSOH_Rx*, *POH_Rx* y *AU_P_Rx* procesan y/o capturan los octetos correspondientes a la *RSOH*, a la *MSOH*, a la *POH* y a los punteros, respectivamente. Por último el módulo *capturar_oct* almacena, al inicio de cada trama, los octetos de la trama anterior capturados por los módulos mencionados anteriormente.

En los siguientes apartados se describen detalladamente los módulos que forman el receptor.

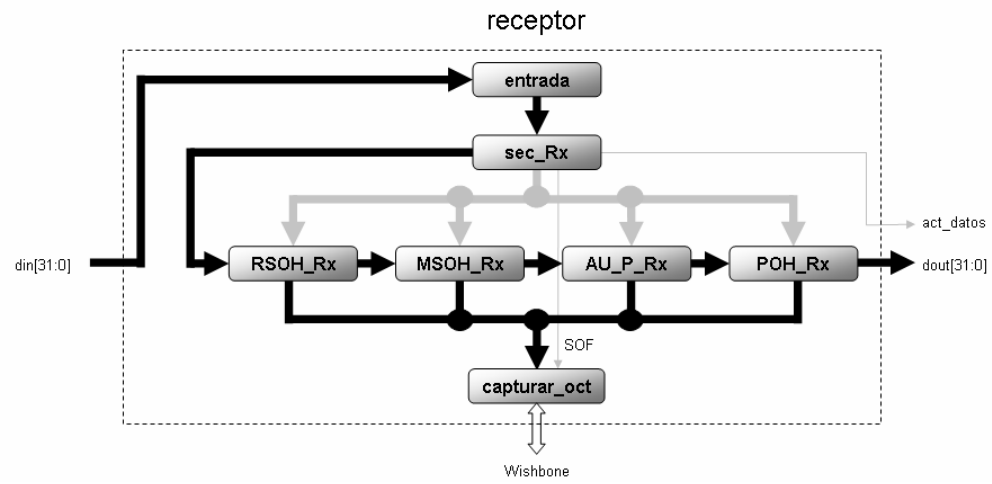


Figura 5.2: Diagrama de bloques del receptor

5.2 Módulo *entrada*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
din[31:0]	E	Datos de entrada
dout[31:0]	S	Datos de salida
num_pal[13:0]	S	Número de palabra
sinc	S	Señal de sincronismo
LOF	S	Pérdida de trama
OOF	S	Fuera de trama

Tabla 5.2: Interfaz del módulo *entrada*

- Estructura interna

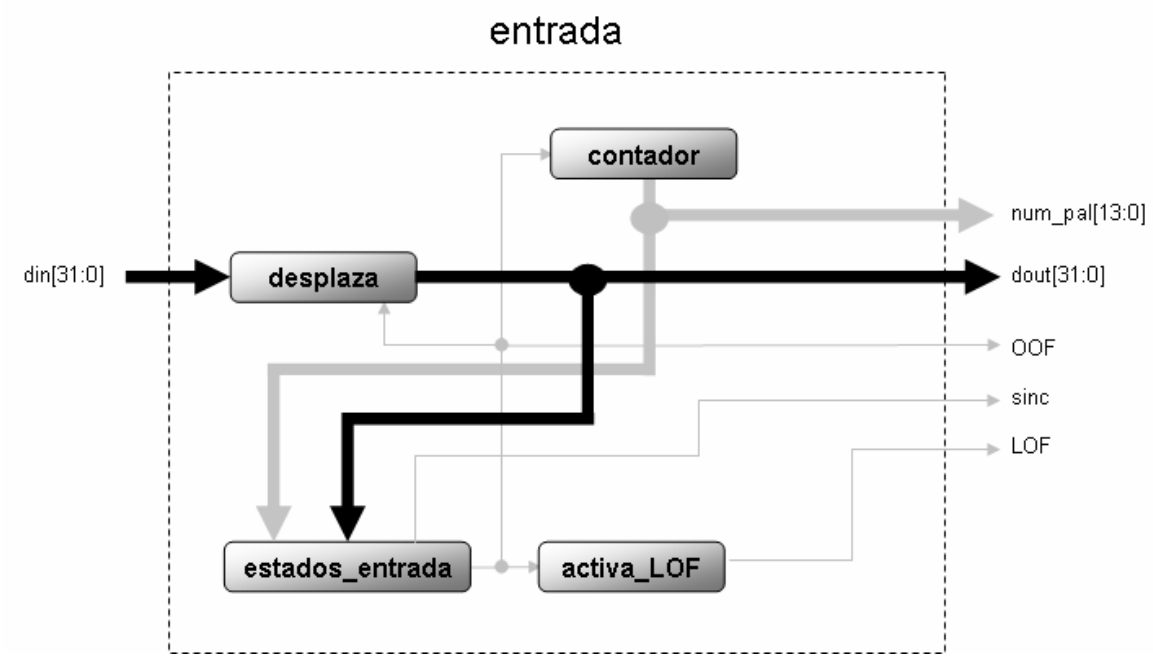


Figura 5.3: Estructura interna del módulo *entrada*

- Funciones

  Este módulo es el encargado de identificar el comienzo de las tramas, o lo que es lo mismo, determinar cuándo el dato presente a su entrada *din[31:0]* contiene el primer octeto *A1*.

  Además lleva la cuenta de las palabras de la trama actual recibidas.

- Funcionamiento

  El módulo *desplaz*a se encarga de desplazar los datos de entrada *din[31:0]*, de forma que el primer octeto *A1* de la trama actual coincida con los 8 bits más significativos de la palabra de datos.

  Una vez alienados los datos, el módulo *estados_entrada* determina cuándo el sistema se ha sincronizado, cómo se verá más adelante, dándole una señal al módulo *contador* para que lleve la cuenta de las palabras de la trama actual que se van recibiendo.

  Por último el módulo *activa_LOF* activa la señal *LOF*.

En los siguientes apartados se describen con más detalles los módulos que componen el módulo *entrada*.

5.2.1 Módulo *desplaza*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
din[31:0]	E	Datos de entrada
dout[31:0]	S	Datos de salida
OOF	E	Fuera de trama

Tabla 5.3: Interfaz del módulo *desplaza*

- Funciones

  Este módulo se encarga de alinear los datos entrantes de forma que el primer octeto *A1* de la trama actual coincida con los 8 bits más significativos de la palabra de datos.

- Funcionamiento

  La figura 5.4 muestra el paralelizador demultiplexando los datos recibidos en serie. En este ejemplo se puede observar cómo el primer bit del primer octeto *A1* se ha situado en el bit 25 de la palabra de datos de 32 bits que va a parar al receptor. En este caso, el módulo *desplaza* debe detectar que en el bit 25 de la entrada *din[31:0]* está el primer bit del octeto *A1*, para posteriormente, situarlo en el bit 31 de la palabra de datos *dout[31:0]* (figura 5.5).

  También se podría dar el caso en el que el primer bit del octeto *A1* se encuentre en cualquiera de los bits de *din[31:0]* entre 6 y 0, cómo en el ejemplo de la figura 5.6, que está situado en el bit 5, razón por la cuál, es necesario almacenar la palabra de datos actual en un registro denominado *din_ant[31:0]*.

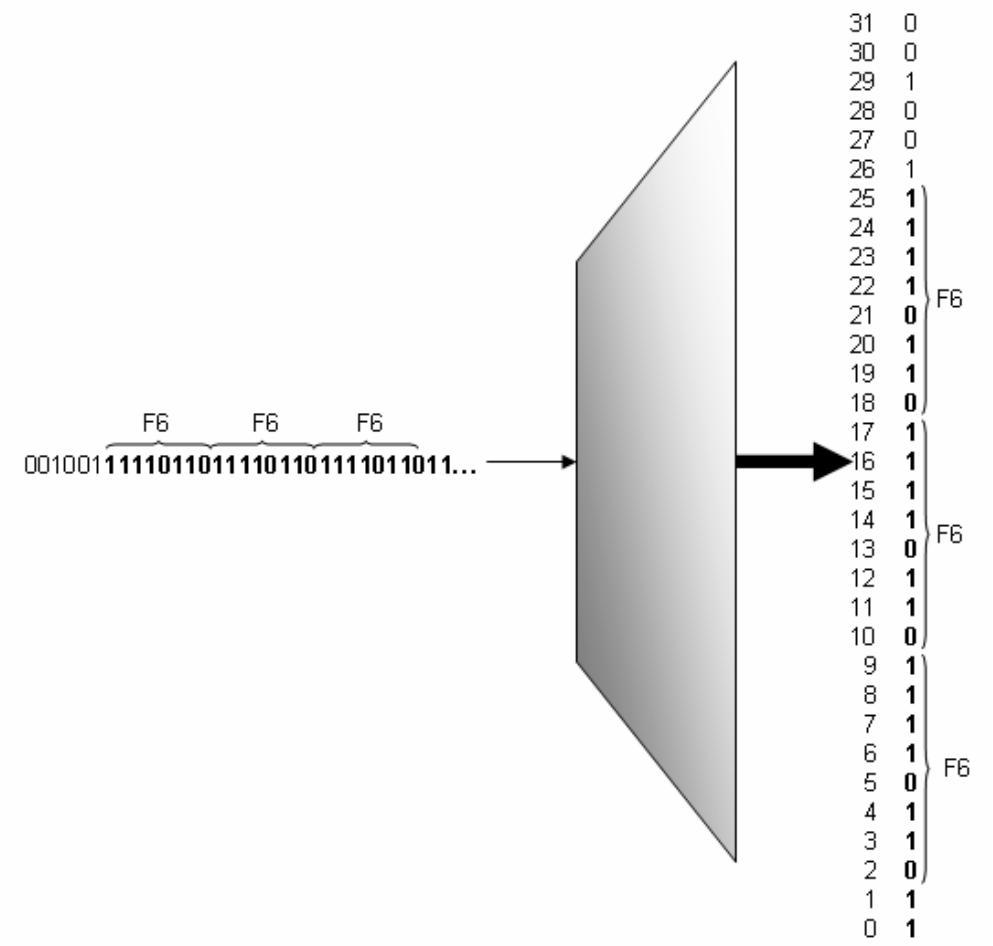
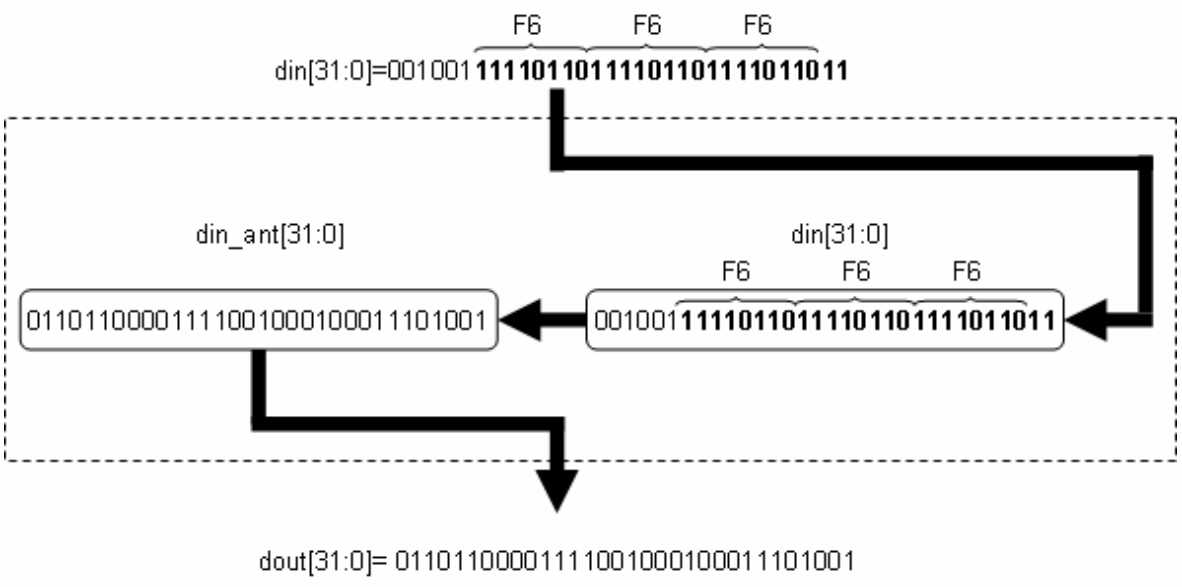
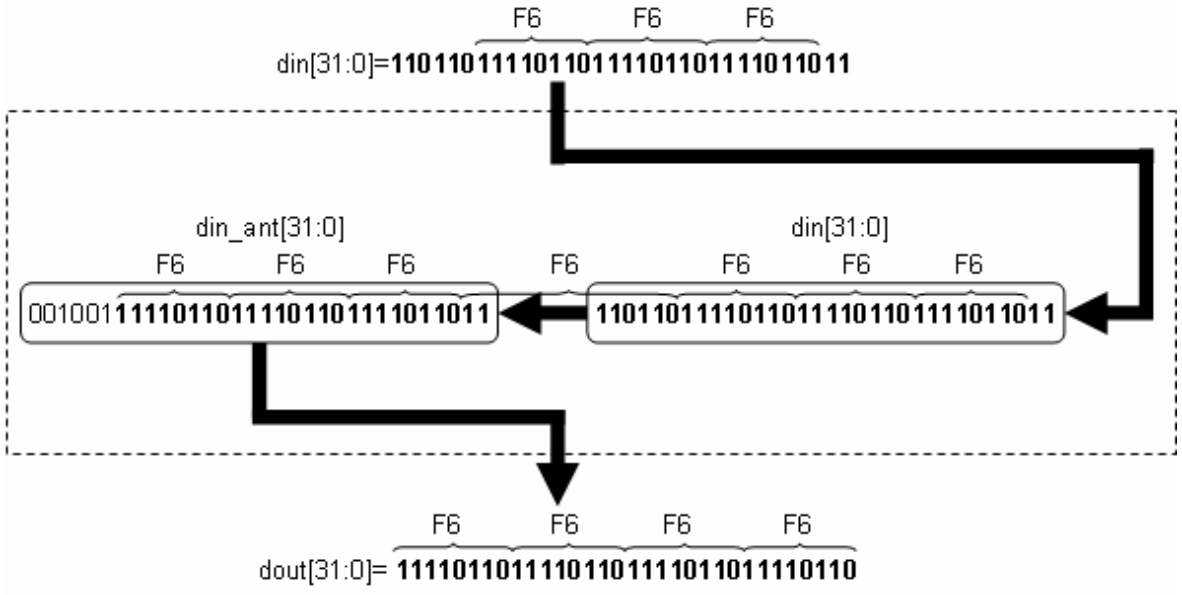


Figura 5.4: Funcionamiento del paralelizador

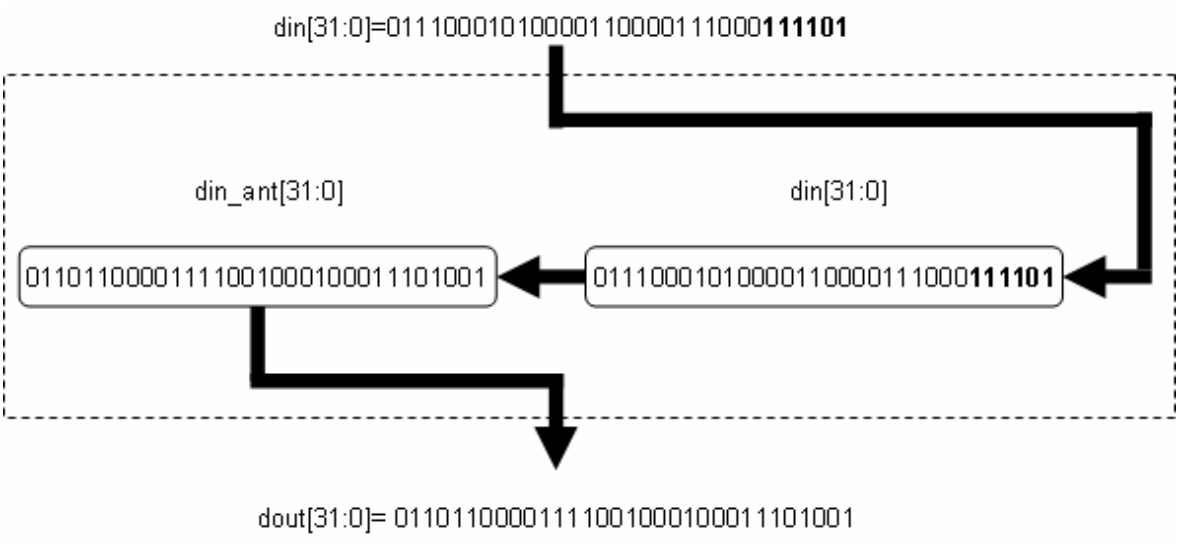


(a)

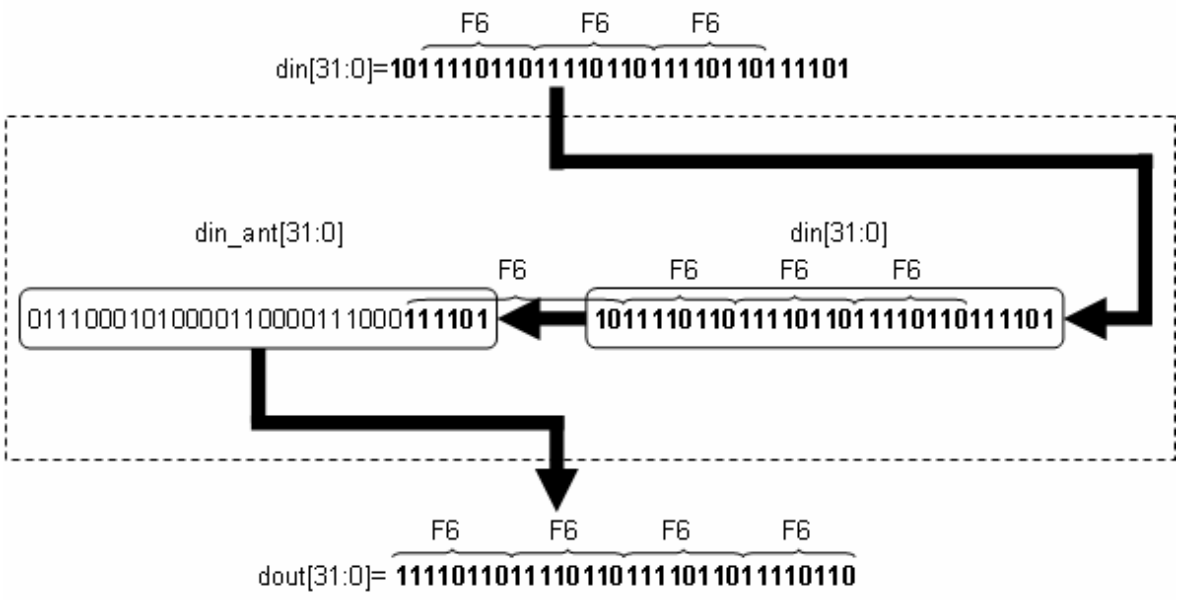


(b)

Figura 5.5: a) En la entrada del módulo se encuentra el primer bit del octeto *A1*, pero la palabra que se analiza es la que había a la entrada un ciclo de reloj antes almacenada en el registro *din_ant[31:0]*. b) El contenido de *din[31:0]* se ha pasado a *din_ant[31:0]*, que se analiza y se detecta el valor *F6* correspondiente al octeto *A1*, y en la salida *dout[31:0]* se entregan los datos alineados.



(a)



(b)

Figura 5.6: a) En la entrada del módulo se encuentra otra vez el primer bit del octeto *A1*, esta vez en el bit 5 de *din[31:0]*. b) En este ejemplo queda claro porque es necesario almacenar el valor de *din[31:0]* un ciclo de reloj anterior en *din_ant[31:0]*, ya que si no se hiciera, es imposible detectar un valor de *F6h*, de ocho bits, en los 6 bits menos significativos de la palabra.

La forma de detectar el octeto *A1* es haciendo una comparación de 8 bits en cada 8 bits consecutivos que forman las palabras *din_ant[31:0]* y *din[31:0]*, es decir:

din_ant[31:24]
din_ant[30:23]
din_ant[29:22]
...
din_ant[2:0] y *din[7:3]*
din_ant[1:0] y *din[7:2]*
din_ant[0] y *din[7:1]*
din[7:0]

Si encuentra un *F6h* (valor del octeto *A1*) se desplazan los datos hasta que dicho *F6h* se sitúe en los 8 bits más significativos de la palabra de datos *dout[31:0]*.

Si un instante después de encontrar un *F6h* la señal *OOF*, procedente del módulo *estados_entrada*, se desactiva, entonces se debe aplicar el mismo desplazamiento al resto de datos de entrada. Si no debe seguir buscando otro valor de *F6h* a su entrada.

En el siguiente apartado se explicará cuándo se activa y desactiva la señal *OOF*.

5.2.2 Módulo *estados_entrada*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
din[31:0]	E	Datos de entrada
num_pal[13:0]	E	Número de palabra
sinc	S	Señal de sincronismo
OOF	S	Fuera de trama

Tabla 5.4: Interfaz del módulo *estados_entrada*

- Funciones

  El módulo *estados_entrada* se encarga de sincronizar el sistema con las tramas entrantes.

- Funcionamiento

  Este módulo se trata de una máquina de estados como la mostrada en la figura 5.7.

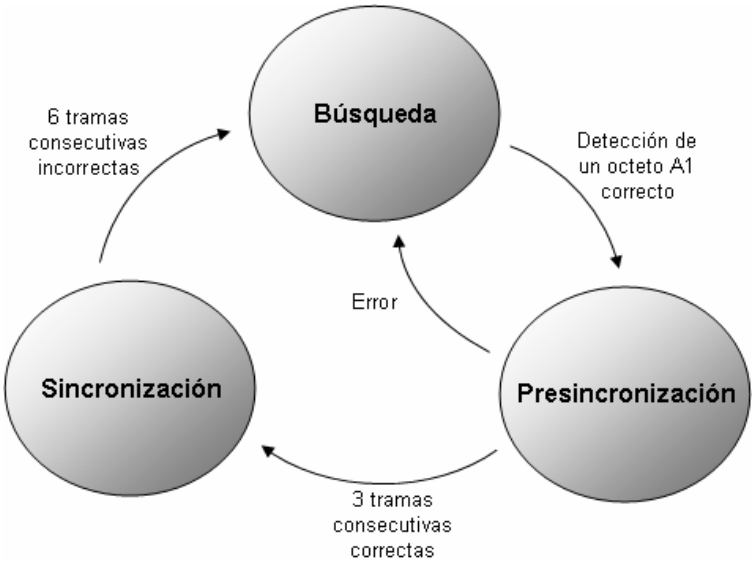


Figura 5.7: Máquina de delineación de trama

Después de un reset, el módulo arranca en el estado de *Búsqueda*. Cuando el módulo *desplaza* detecta un *F6h* lo sitúa en los 8 bits más significativos de la palabra de datos *din[31:0]* del módulo *estados_entrada*. Éste último debe diferenciar si se trata de un *F6h* aislado o del primer octeto *A1* de la trama actual. Si estuviéramos en este último caso, los tres octetos restantes de *din[31:0]* deben tener también un valor de *F6h*, correspondientes a los tres siguientes octetos *A1*, con lo que se pasaría al estado de *Presincronización*.

En el estado de *Presincronización*, el módulo debe comprobar que se reciben 3 tramas consecutivas correctas, es decir, con todos sus octetos *A1* y *A2* correctos, pasándose al estado de *Sincronización* si es así. Si no, se vuelve al estado de *Búsqueda*.

Una vez el sistema haya sincronizado permanecerá en este estado todo el tiempo a menos que se reciban 6 tramas consecutivas incorrectas, es decir, con alguno de los octetos *A1* y/o *A2* incorrectos.

En el estado de *Sincronización* se activa la señal *sinc*, permaneciendo inactiva en el resto de estados.

La señal *OOF* se activa en el estado de *Búsqueda* y se desactiva en el resto. De esta forma, cuando el módulo *desplaza* detecta un *F6h* y el módulo *estados_entrada* determina que se trata del primer *A1*, se pasa al estado de *presincronización* desactivándose la señal *OOF*, que puede servir para indicar al módulo *desplaza* que realice el mismo desplazamiento al resto de palabras.

5.2.3 Módulo *contador*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
OOF	E	Fuera de trama
num_pal[13:0]	S	Número de palabra

Tabla 5.5: Interfaz del módulo *contador*

- Funciones

El módulo *contador* lleva la cuenta de las palabras de la trama actual que se van recibiendo.

- Funcionamiento

Mientras el módulo *estados_entrada* no haya identificado el comienzo de la trama, el contador permanecerá inactivo. Si se desactiva la señal *OOF* se habrá encontrado la primera palabra de la trama y el módulo *contador* empezará a contar de forma cíclica de 0 a 9719.

Si se desincroniza el sistema, activándose la señal *OOF*, el contador volverá a estar inactivo.

5.2.4 Módulo *activa_LOF*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
OOF	E	Fuera de trama
LOF	S	Pérdida de trama

Tabla 5.6: Interfaz del módulo *activa_LOF*

- Funciones

  La única función de este módulo es la de activar la señal *LOF* cuando la señal *OOF* permanezca 2 ms activada, y desactivarla cuando ésta última permanezca el mismo tiempo desactivada.

- Funcionamiento

  Desde el momento en que se activa la señal *OOF*, un contador de 18 bits *count[17:0]* se pone en marcha hasta que transcurren 155520 ciclos de reloj (siempre que *OOF* no se desactive), momento en el que se activa la señal *LOF*. Lo mismo ocurre cuando la señal *OOF* se desactiva.

  Los 155520 ciclos de reloj se calculan de la siguiente forma: se sabe que la frecuencia de reloj es de 77,76 MHz, ya que la velocidad de recepción es de 2,48832 Gbps y que los datos se procesan en paralelo de 32 en 32 bits. Por lo tanto su periodo es de 1/77,76 μs. Entonces para saber cuántos periodos corresponden a 2 ms:

$$\frac{2 \cdot 10^{-3}}{1 / 77{,}76 \cdot 10^{-6}} = 155520 \text{ ciclos de reloj}$$

Y el número de bits del contador necesarios para poder contar hasta 155520:

$$2^{17} < 155520 < 2^{18}$$

5.3 Módulo *RSOH_Rx*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
din[31:0]	E	Datos de entrada
dout[31:0]	S	Datos de salida
SOF	E	Comienzo de la trama
start_scr	E	Señal de activación del aleatorizador
act_B1	E	Señal de captura del octeto *B1*
act_J0	E	Señal de captura del octeto *J0*
act_E1	E	Señal de captura del octeto *E1*
act_F1	E	Señal de captura del octeto *F1*
act_D1	E	Señal de captura del octeto *D1*
act_D2	E	Señal de captura del octeto *D2*
act_D3	E	Señal de captura del octeto *D3*
read_J0	S	Datos del octeto *J0*
read_E1	S	Datos del octeto *J0*
read_F1	S	Datos del octeto *J0*
read_D1	S	Datos del octeto *D1*
read_D2	S	Datos del octeto *D2*
read_D3	S	Datos del octeto *D3*
error_B1	S	Señal de error en *B1*

Tabla 5.7: Interfaz del módulo *RSOH_Rx*

- Estructura interna

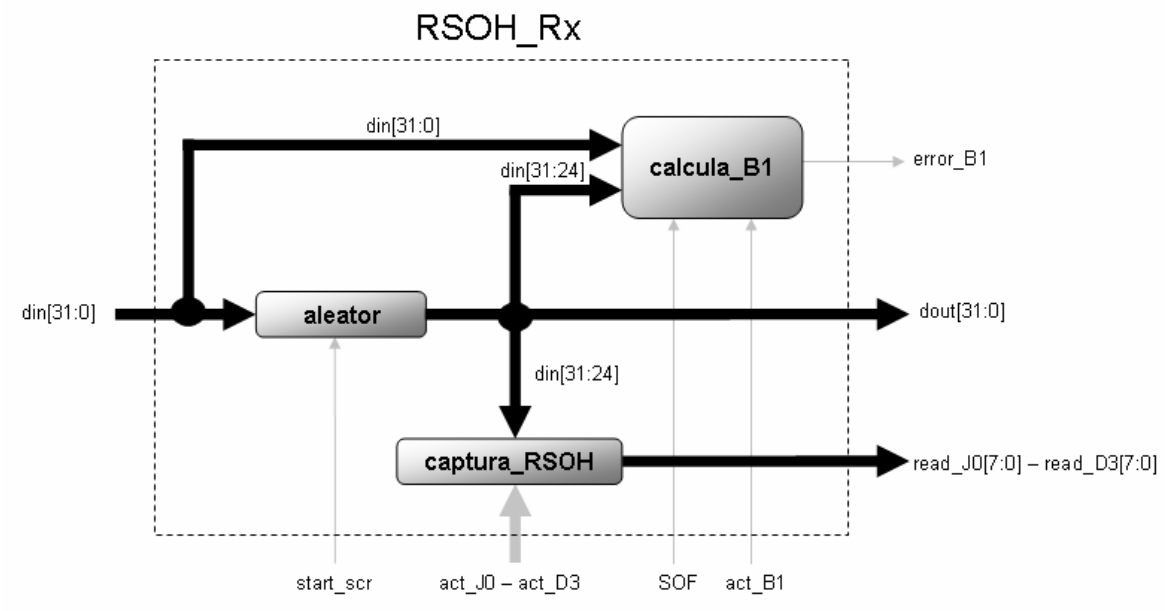


Figura 5.8: Estructura interna del módulo *RSOH_Rx*

- Funciones

  Este módulo es el encargado de desaleatorizar las tramas *SDH*, realizar el cálculo y comparación del octeto *B1* y extraer los octetos *J0*, *E1*, *F1*, *D1*, *D2* y *D3*.

- Funcionamiento

  El módulo *aleator* desaleatoriza la trama. El módulo *calcula_B1* realiza el cálculo del octeto *B1* en la trama actual sin desaleatorizar para compararlo con el recibido en la siguiente trama.

  El módulo *captura_RSOH* extrae los octetos *J0*, *E1*, *F1*, *D1*, *D2* y *D3* de la trama una vez desaleatorizada.

En los siguientes apartados se describen con más detalle los módulos que componen el módulo *RSOH_Rx*.

5.3.1 Módulo *aleator*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
din[31:0]	E	Datos de entrada
start_scr	E	Señal de activación del aleatorizador
dout[31:0]	S	Datos de salida

Tabla 5.8: Interfaz del módulo *aleator*

- Funciones

  Éste es el módulo encargado de desaleatorizar la trama.

- Funcionamiento

  La señal *start_scr* indica cuando se debe desaleatorizar la trama, que debe afectar a toda la trama *SDH* excepto a la primera fila de la *RSOH*.

  Por un lado se genera la secuencia de desaleatorización mediante el registro *aux_scr[6:0]*, cuyo valor inicial en cada trama es *1111111b*. Si los datos llegaran en serie, habría que realizar la *OR-exclusiva* entre el bit de dato entrante y el bit 6 del registro *aux_scr[6:0]* y desplazar éste último para poder realizar la misma operación en el siguiente ciclo de reloj con el siguiente dato. En este caso, los bits llegan en paralelo de 32 en 32. El bit 6 de *aux_scr[6:0]* le corresponde a *din[31]*, pero ¿cuál le correspondería al resto? Observando la figura 2.32, si desplazáramos una vez el registro *aux_scr[6:0]* entonces en el bit 6 de *aux_scr[6:0]* tendríamos el bit que le corresponde a *din[30]* que sería el valor de *aux_scr[5]* un ciclo de reloj anterior. Si volviéramos a desplazar *aux_scr[6:0]* otra vez tendríamos el bit que le corresponde a *din[29]* que sería el valor de *aux_scr[4]* dos ciclos de reloj antes. Y

así sucesivamente hasta obtener los 32 bits (mostrados en la tabla 5.9, junto con el siguiente valor de *aux_scr[6:0]*) que les corresponderían a *din[31:0]*.

Desp	aux_scr	din
0	aux_scr[6]	din[31]
1	aux_scr[5]	din[30]
2	aux_scr[4]	din[29]
3	aux_scr[3]	din[28]
4	aux_scr[2]	din[27]
5	aux_scr[1]	din[26]
6	aux_scr[0]	din[25]
7	aux_scr[5] $\oplus$ aux_scr[6]	din[24]
8	aux_scr[4] $\oplus$ aux_scr[5]	din[23]
9	aux_scr[3] $\oplus$ aux_scr[4]	din[22]
10	aux_scr[2] $\oplus$ aux_scr[3]	din[21]
11	aux_scr[1] $\oplus$ aux_scr[2]	din[20]
12	aux_scr[0] $\oplus$ aux_scr[1]	din[19]
13	aux_scr[0] $\oplus$ aux_scr[5] $\oplus$ aux_scr[6]	din[18]
14	aux_scr[4] $\oplus$ aux_scr[6]	din[17]
15	aux_scr[3] $\oplus$ aux_scr[5]	din[16]
16	aux_scr[2] $\oplus$ aux_scr[4]	din[15]
17	aux_scr[1] $\oplus$ aux_scr[3]	din[14]
18	aux_scr[0] $\oplus$ aux_scr[2]	din[13]
19	aux_scr[1] $\oplus$ aux_scr[5] $\oplus$ aux_scr[6]	din[12]
20	aux_scr[0] $\oplus$ aux_scr[4] $\oplus$ aux_scr[5]	din[11]
21	aux_scr[3] $\oplus$ aux_scr[4] $\oplus$ aux_scr[5] $\oplus$ aux_scr[6]	din[10]
22	aux_scr[2] $\oplus$ aux_scr[3] $\oplus$ aux_scr[4] $\oplus$ aux_scr[5]	din[9]
23	aux_scr[1] $\oplus$ aux_scr[2] $\oplus$ aux_scr[3] $\oplus$ aux_scr[4]	din[8]
24	aux_scr[0] $\oplus$ aux_scr[1] $\oplus$ aux_scr[2] $\oplus$ aux_scr[3]	din[7]
25	aux_scr[0] $\oplus$ aux_scr[1] $\oplus$ aux_scr[2] $\oplus$ aux_scr[5] $\oplus$ aux_scr[6]	din[6]
26	aux_scr[0] $\oplus$ aux_scr[1] $\oplus$ aux_scr[4] $\oplus$ aux_scr[6]	din[5]
27	aux_scr[0] $\oplus$ aux_scr[3] $\oplus$ aux_scr[6]	din[4]
28	aux_scr[2] $\oplus$ aux_scr[6]	din[3]
29	aux_scr[1] $\oplus$ aux_scr[5]	din[2]
30	aux_scr[0] $\oplus$ aux_scr[4]	din[1]
31	aux_scr[3] $\oplus$ aux_scr[5] $\oplus$ aux_scr[6]	din[0]
32	aux_scr[2] $\oplus$ aux_scr[4] $\oplus$ aux_scr[5]	aux_scr[6]
33	aux_scr[1] $\oplus$ aux_scr[3] $\oplus$ aux_scr[4]	aux_scr[5]
34	aux_scr[0] $\oplus$ aux_scr[2] $\oplus$ aux_scr[3]	aux_scr[4]
35	aux_scr[1] $\oplus$ aux_scr[2] $\oplus$ aux_scr[5] $\oplus$ aux_scr[6]	aux_scr[3]
36	aux_scr[0] $\oplus$ aux_scr[1] $\oplus$ aux_scr[4] $\oplus$ aux_scr[5]	aux_scr[2]
37	aux_scr[0] $\oplus$ aux_scr[3] $\oplus$ aux_scr[4] $\oplus$ aux_scr[5] $\oplus$ aux_scr[6]	aux_scr[1]
38	aux_scr[2] $\oplus$ aux_scr[3] $\oplus$ aux_scr[4] $\oplus$ aux_scr[6]	aux_scr[0]

Tabla 5.9: Operación a realizar con cada bit de *din[31:0]* para la desaleatorización en paralelo

El aleatorizador realiza la *OR-exclusiva* entre éstos bits y *din[31:0]* en cada ciclo de reloj para poder desaleatorizar la trama en paralelo.

5.3.2 Módulo *calcula_B1*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
din[31:0]	E	Datos de entrada
input_B1[7:0]	E	*B1* de entrada
SOF	E	Comienzo de la trama
act_B1	E	Señal de captura del octeto *B1*
error_B1	S	Señal de error en *B1*
lect_B1	E	Señal de lectura de los errores en *B1* acumulados
output_B1[15:0]	S	Errores acumulados por el octeto *B1*

Tabla 5.10: Interfaz del módulo *calcula_B1*

- Funciones

El módulo *calcula_B1* se encarga de realizar el cálculo y comparación del octeto *B1*.

- Funcionamiento

Se va realizando palabra por palabra el cálculo de *B1* de la trama sin aleatorizar (a través de *din[31:0]*) almacenando temporalmente el resultado en un registro llamado *B1_temp[7:0]*. Al inicio de una nueva trama (indicado con *SOF*), el valor calculado en *B1_temp[7:0]* se pasa al registro *B1_calc[7:0]* y se inicia otra vez el cálculo con *B1_temp[7:0]*.

Al activarse la señal *act_B1*, el registro *B1_calc[31:0]*, que contiene el valor de *B1* calculado de la trama anterior, se compara con el *B1* capturado en la trama actual, que está presente en la entrada *input_B1[7:0]*. Si no coinciden, se activa la señal *error_B1* que permanecerá así al menos hasta que se reciba el siguiente octeto *B1*.

Además hay que contar el número de bits erróneos en el octeto *B1*, que se irán acumulando trama por trama en el registro *acum_B1[15:0]*. Este registro puede ser leído mediante la salida *output_B1[15:0]*, cuando la señal *lect_B1* esté activada. Dicha lectura produce inicialización del registro *acum_B1[15:0]*.

5.3.3 Módulo *captura_RSOH*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
din[7:0]	E	Datos de entrada
act_J0	E	Señal de captura del octeto *J0*
act_E1	E	Señal de captura del octeto *E1*
act_F1	E	Señal de captura del octeto *F1*
act_D1	E	Señal de captura del octeto *D1*
act_D2	E	Señal de captura del octeto *D2*
act_D3	E	Señal de captura del octeto *D3*
read_J0	S	Datos del octeto *J0*
read_E1	S	Datos del octeto *J0*
read_F1	S	Datos del octeto *J0*
read_D1	S	Datos del octeto *D1*
read_D2	S	Datos del octeto *D2*
read_D3	S	Datos del octeto *D3*

Tabla 5.11: Interfaz del módulo *captura_RSOH*

- Funciones

  El módulo *captura_RSOH* se encarga de la extracción de los octetos *J0*, *E1*, *F1*, *D1*, *D2* y *D3*.

- Funcionamiento

  Cuándo recibe la señal de alguna de sus entradas *act_XX*, pasa el valor de *din[7:0]* al registro de salida *read_XX[7:0]* correspondiente.

5.4 Módulo *MSOH_Rx*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
din[31:0]	E	Datos de entrada
dout[31:0]	S	Datos de salida
SOF	E	Comienzo de la trama
BER	E	Señal de error *BER*
act_reg	E	Señalización de la tara de regeneración
act_B2	E	Señal de captura del octeto *B2*
act_K1	E	Señal de captura del octeto *K1*
act_K2	E	Señal de captura del octeto *K2*
act_D4	E	Señal de captura del octeto *D4*
act_D5	E	Señal de captura del octeto *D5*
act_D6	E	Señal de captura del octeto *D6*
act_D7	E	Señal de captura del octeto *D7*
act_D8	E	Señal de captura del octeto *D8*
act_D9	E	Señal de captura del octeto *D9*
act_D10	E	Señal de captura del octeto *D10*
act_D11	E	Señal de captura del octeto *D11*
act_D12	E	Señal de captura del octeto *D12*
act_S1	E	Señal de captura del octeto *S1*

Tabla 5.12: Interfaz del módulo *MSOH_Rx*

Nombre	E/S	Descripción
act_M1	E	Señal de captura del octeto *M1*
act_E2	E	Señal de captura del octeto *E2*
read_K1	S	Datos del octeto *K1*
read_K2	S	Datos del octeto *K2*
read_D4	S	Datos del octeto *D4*
read_D5	S	Datos del octeto *D5*
read_D6	S	Datos del octeto *D6*
read_D7	S	Datos del octeto *D7*
read_D8	S	Datos del octeto *D8*
read_D9	S	Datos del octeto *D9*
read_D10	S	Datos del octeto *D10*
read_D11	S	Datos del octeto *D11*
read_D12	S	Datos del octeto *D12*
read_S1	S	Datos del octeto *S1*
read_M1	S	Datos del octeto *M1*
read_E2	S	Datos del octeto *E2*
MS_AIS	S	Señal *MS-AIS*
MS_RDI	S	Señal *MS-RDI*
error_B2	S	Señal de error en *B2*

Tabla 5.12: Interfaz del módulo *MSOH_Rx*
(continuación)

- Estructura interna

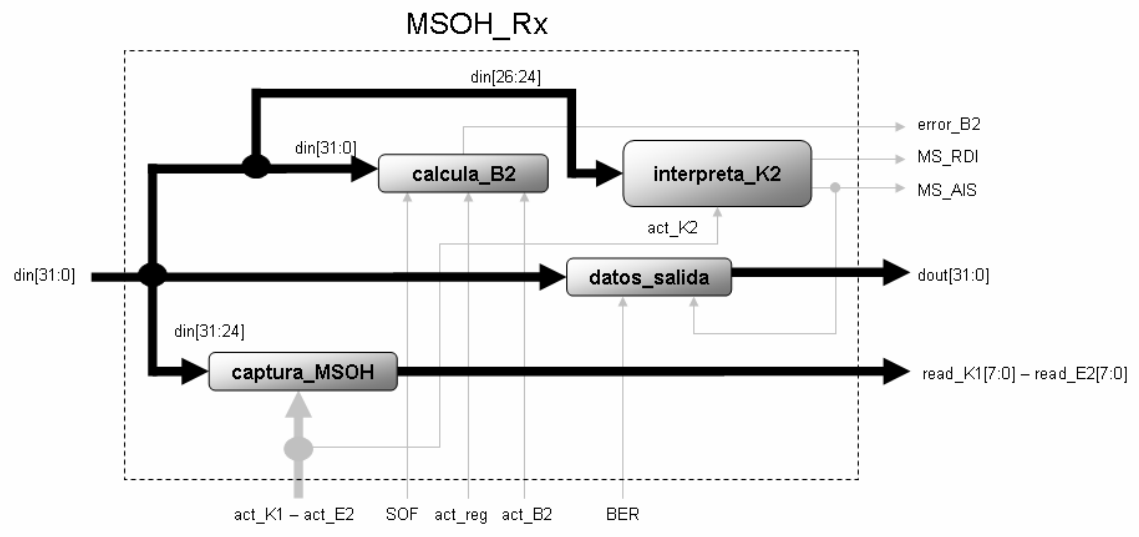


Figura 5.9: Estructura interna del módulo *MSOH_Rx*

- Funciones

  Este módulo es el encargado de realizar el cálculo y comparación de los octetos de *B2*, detectar y generar los errores *MS-AIS* y *MS-RDI* y extraer los octetos *K1*, *K2*, *D4*, *D5*, *D6*, *D7*, *D8*, *D9*, *D10*, *D11*, *D12*, *S1*, *M1* y *E2*.

- Funcionamiento

  El módulo *calcula_B2* realiza el cálculo de los octetos de *B2* en la trama actual para compararlo con los recibidos en la siguiente trama.

  El módulo *interpreta_K2* activa las señales *MS-AIS* y *MS-RDI* en función del valor del octeto *K2* que se va recibiendo en cada trama.

  El módulo *captura_MSOH* extrae los octetos *K1*, *K2*, *D4*, *D5*, *D6*, *D7*, *D8*, *D9*, *D10*, *D11*, *D12*, *S1*, *M1*, y *E2* de la trama.

  Por último el módulo *datos_salida* dejará pasar los datos si las señales *MS-AIS* y *BER* están desactivadas. En caso contrario, mostrará un valor de *FFFFFFFFh* a su salida.

En los siguientes apartados se describen con más detalle los módulos que componen el módulo *MSOH_Rx*.

5.4.1 Módulo *calcula_B2*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
din[31:0]	E	Datos de entrada
SOF	E	Comienzo de la trama
act_reg	E	Señalización de la tara de regeneración
act_B2	E	Señal de captura de los octetos de *B2*
error_B2	S	Señal de error en *B2*
signal2Rx	E	Protocolo *Handshake*
signal2Tx	S	Protocolo *Handshake*
output_frame_B2[7:0]	S	Errores detectados por el octeto *B2*

Tabla 5.13: Interfaz del módulo *calcula_B2*

- Funciones

  El módulo *calcula_B2* se encarga de realizar el cálculo y comparación de los octetos de *B2*.

- Funcionamiento

  El módulo dispone internamente de dos registros de 384 bits: *B2_temp[383:0]*, donde se almacena temporalmente los resultados intermedios del cálculo de *B2*, y *B2_calc[383:0]*, donde se almacena el resultado final del cálculo de *B2* de una trama para ser comparado con el *B2* recibido en la trama siguiente.

  El procedimiento para calcular *B2*, descrito en la figura 5.10, se puede resumir en los siguientes pasos:

  1º Al comienzo de una trama (se activa *SOF*) *B2_temp[383:0]* se inicializa a 0, almacenándose su valor en *B2_calc[383:0]* (figura 5.10 a)).

2º Las palabras deberían ser agrupadas hasta formar 384 bits para realizar la *OR-exclusiva* con *B2_temp[383:0]*. En lugar de esto se realiza siempre la *OR-exclusiva* del dato entrante *din[31:0]* con *B2_temp[383:352]* y se almacena el resultado en *B2_temp[31:0]*, al tiempo que se rotan los bits de *B2_temp[383:0]* a la izquierda (figura 5.10 b)).

3º Tras 11 ciclos de reloj el resultado que se había almacenado en *B2_temp[31:0]* en la figura 5.10 b) ya se ha desplazado hasta *B2_temp[383:352]* (figura 5.10 c)).

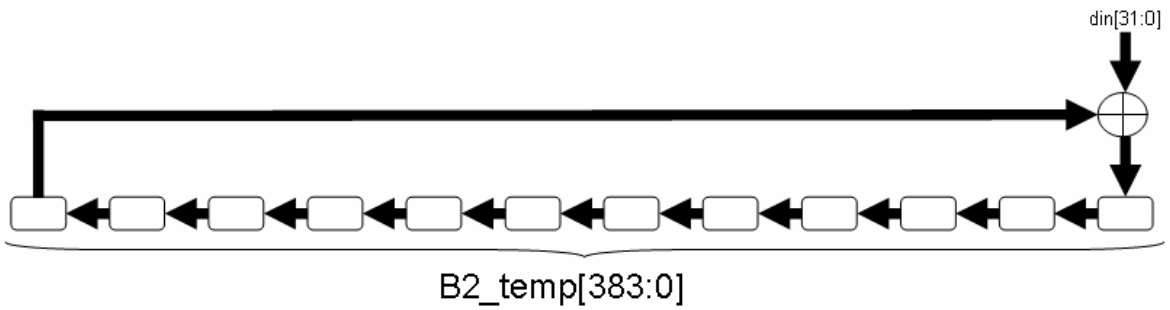
Este procedimiento se repite hasta que comience una nueva trama. De esta forma no es necesario agrupar las palabras para realizar el cálculo.

La comparación del *B2* calculado y del *B2* recibido se realiza a medida que se recibe éste último, mientras la señal *act_B2* esté activada. Cuando se recibe los primeros 32 bits de *B2* por *din[31:0]* se comparan con *B2_calc[383:352]* y se desplazan los bits hacia la izquierda. Éste proceso se repite hasta que *B2_calc[31:0]* llegue a *B2_calc[383:352]* y se compare con *din[31:0]*, que será el último ciclo durante la trama actual en el que la señal *act_B2* esté activada.

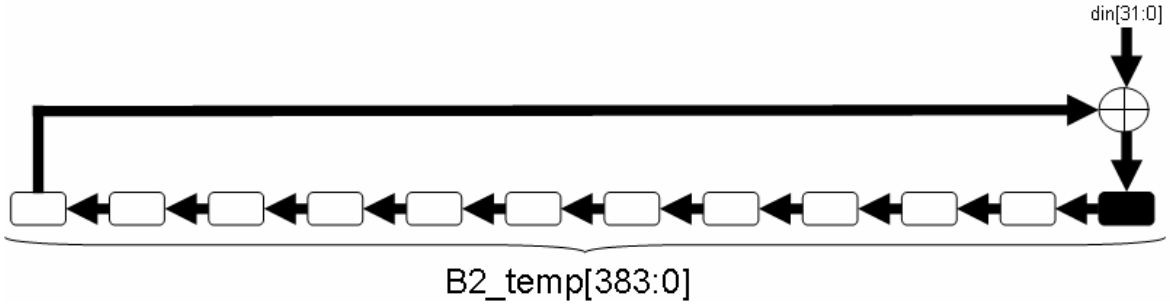
Al igual que en el módulo *calcula_B1*, si no coinciden *B2_calc[383:0]* con el *B2* recibido se activa la señal *error_B2* hasta al menos cuando se reciba el *B2* de la siguiente trama.

La cantidad de errores detectados en cada trama se transfiere a un sistema externo mediante un protocolo *handshake*, al final de cada trama. Para ello se activa la señal *signal2Tx* a la vez que se pone el número de errores en la salida *output_frame_B2[7:0]*. Cuando se active la señal *signal2Rx* se desactiva entonces *signal2Tx*.

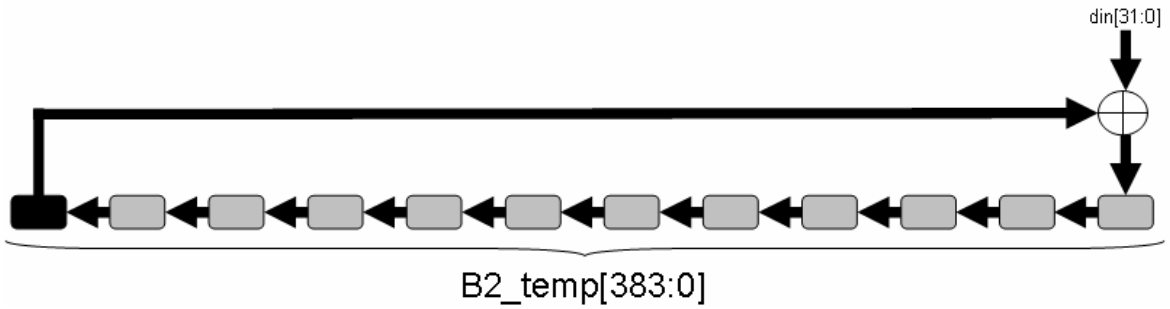
Debido a que la máxima cantidad de errores que se pueden detectar son 384 y sólo se utiliza un octeto, puede ser necesario truncar el número de errores a 256. No se contempla ninguna medida específica en este último caso.



(a)



(b)



(c)

Figura 5.10: a) Se inicializa el registro *B2_temp[383:0]* a cero, representado en la figura con todos los cuadros en blanco. b) Se realiza el cálculo entre *din[31:0]* y *B2_temp[383:352]* y se almacena en *B2_temp[31:0]*, representado por el cuadro en negro. Las palabras se desplazan hacia la derecha. c) Ya se ha realizado el cálculo con todos bits de *B2_temp[383:0]*, y el registro está preparado para repetir el mismo procedimiento hasta que se reciba toda la trama.

5.4.2 Módulo *captura_MSOH*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
din[7:0]	E	Datos de entrada
act_K1	E	Señal de captura del octeto *K1*
act_K2	E	Señal de captura del octeto *K2*
act_D4	E	Señal de captura del octeto *D4*
act_D5	E	Señal de captura del octeto *D5*
act_D6	E	Señal de captura del octeto *D6*
act_D7	E	Señal de captura del octeto *D7*
act_D8	E	Señal de captura del octeto *D8*
act_D9	E	Señal de captura del octeto *D9*
act_D10	E	Señal de captura del octeto *D10*
act_D11	E	Señal de captura del octeto *D11*
act_D12	E	Señal de captura del octeto *D12*
act_S1	E	Señal de captura del octeto *S1*
act_M1	E	Señal de captura del octeto *M1*
act_E2	E	Señal de captura del octeto *E2*
read_K1	S	Datos del octeto *K1*
read_K2	S	Datos del octeto *K2*
read_D4	S	Datos del octeto *D4*
read_D5	S	Datos del octeto *D5*
read_D6	S	Datos del octeto *D6*
read_D7	S	Datos del octeto *D7*
read_D8	S	Datos del octeto *D8*
read_D9	S	Datos del octeto *D9*
read_D10	S	Datos del octeto *D10*
read_D11	S	Datos del octeto *D11*
read_D12	S	Datos del octeto *D12*
read_S1	S	Datos del octeto *S1*
read_M1	S	Datos del octeto *M1*
read_E2	S	Datos del octeto *E2*

Tabla 5.14: Interfaz del módulo *captura_MSOH*

- Funciones

  El módulo *captura_MSOH* se encarga de la extracción de los octetos *K1*, *K2*, *D4*, *D5*, *D6*, *D7*, *D8*, *D9*, *D10*, *D11*, *D12*, *S1*, *M1*, y *E2*.

- Funcionamiento

  Su funcionamiento es similar al del módulo *captura_RSOH* visto en el apartado 5.3.3.

5.4.3 Módulo *interpreta_K2*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
din[2:0]	E	Datos de entrada
act_K2	E	Señal de captura del octeto *K2*
MS_AIS	S	Señal *MS-AIS*
MS_RDI	S	Señal *MS-RDI*

Tabla 5.15: Interfaz del módulo *interpreta_K2*

- Funciones

  Este módulo se encarga de detectar y generar los errores *MS-AIS* y *MS-RDI*.

- Funcionamiento

  La generación de las señales de error *MS-AIS* y *MS-RDI* depende de los bits [2:0] del octeto *K2*. Si se reciben tres tramas consecutivas con *111b* se activa la señal *MS-AIS*, desactivándose *MS-RDI* si estaba activa. Por el contrario, si se reciben tres

tramas consecutivas con *110b* se activa la señal *MS-RDI*, desactivándose *MS-AIS*. Si se reciben tres tramas consecutivas con valores distintos a los anteriores se desactivan ambas señales.

5.4.4 Módulo *datos_salida*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
din[31:0]	E	Datos de entrada
BER	E	Señal de error *BER*
MS_AIS	S	Señal *MS-AIS*
dout[31:0]	S	Datos de salida

Tabla 5.16: Interfaz del módulo *datos_salida*

- Funciones

Este módulo dejará pasar o no los datos en función de las señales *BER* y *MS-AIS*.

- Funcionamiento

Si alguna de las señales *BER* o *MS-AIS* está activa, se mostrará a la salida un valor de *FFFFFFFFh*. Si no, los datos de la entrada pasarán directamente a la salida.

5.5 Módulo *AU_P_Rx*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
din[31:0]	E	Datos de entrada
dout[31:0]	S	Datos de salida
act_H1	E	Señal de captura del octeto *H1*
act_H2	E	Señal de captura del octeto *H2*
act_H1_c	E	Señal de captura de los octetos *H1* concatenado
act_H2_c	E	Señal de captura de los octetos *H2* concatenado
act_pointer	E	Señal de procesamiento de punteros
dec	S	Señal de decremento de puntero
inc	S	Señal de incremento de puntero
desp[9:0]	S	Valor del desplazamiento del *VC* de la trama actual
desp_ant[9:0]	S	Valor del desplazamiento del *VC* de la trama anterior
AU_AIS	S	Señal *AU_AIS*
AU_LOP	S	Señal *AU_LOP*

Tabla 5.17: Interfaz del módulo *AU_P_Rx*

- Estructura interna

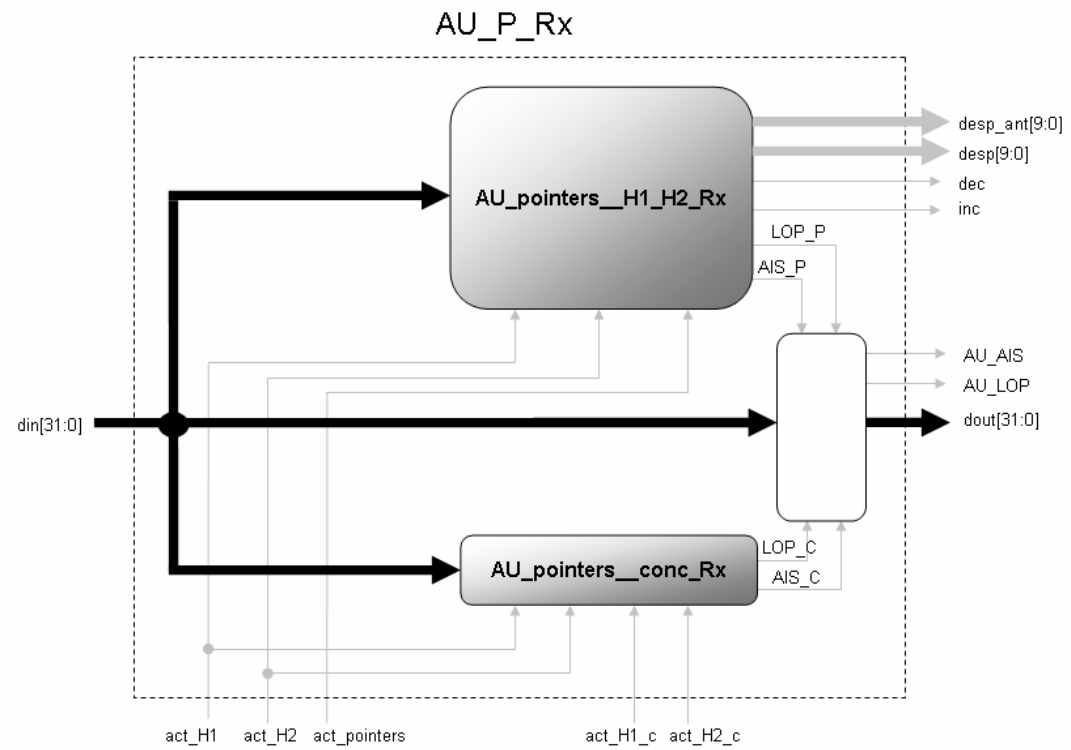


Figura 5.11: Estructura interna del módulo *AU_P_Rx*

- Funciones

  La misión de este módulo es interpretar los punteros para encontrar el *VC* que almacena los datos. Además genera las señales de error *AU_AIS* y *AU_LOP*.

- Funcionamiento

  El módulo *AU_pointers_H1_H2_Rx* procesa los octetos *H1* y *H2* que contienen el desplazamiento del *VC* y el módulo *AU_pointers_conc_Rx* los punteros concatenados.

  La señal *AU_AIS* se activa mientras lo estén las señales *AIS_P* y *AIS_C* al mismo tiempo. La señal *AU_LOP* se activa cuando lo esté cualquiera de las señales *AIS_C*, *AIS_P*, *LOP_C* y *LOP_P*, además la salida *dout[31:0]* valdrá *FFFFFFFFh*.

Estas últimas cuatro señales así como los módulos que componen el módulo *AU_P_Rx* se explican a continuación.

5.5.1 Módulo *AU_pointers_H1_H2_Rx*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
din[7:0]	E	Datos de entrada
act_H1	E	Señal de captura del octeto *H1*
act_H2	E	Señal de captura del octeto *H2*
act_process_pointer	E	Señal de procesamiento de punteros
desp[9:0]	S	Valor del desplazamiento del *VC* de la trama actual
H1H2_ant[9:0]	S	Valor del desplazamiento del *VC* de la trama anterior
LOP_P	S	Señal *LOP-P*
AIS_P	S	Señal *AIS-P*
dec	S	Señal de decremento de puntero
inc	S	Señal de incremento de puntero

Tabla 5.18: Interfaz del módulo *AU_pointers_H1_H2_Rx*

- Funciones

  El módulo *AU_pointers_H1_H2_Rx* procesa los octetos *H1* y *H2* que contienen el desplazamiento del *VC*.

- Funcionamiento

  La figura 5.12 muestra un diagrama de bloques de este módulo. Los punteros se procesan según la máquina de estados vista en el apartado 2.2.5.3, figura 2.27.

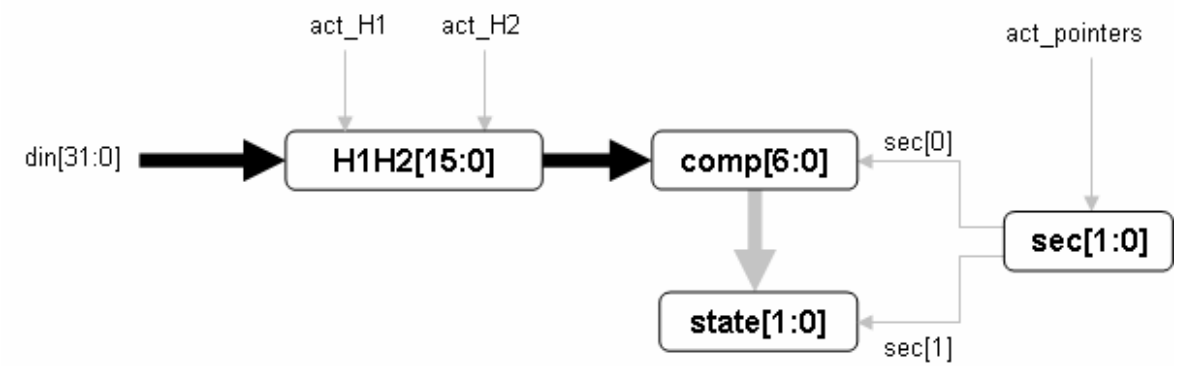


Figura 5.12: Diagrama de bloques del módulo *AU_pointers_H1_H2_Rx*

Cuando se activa la señal *act_H1* se almacena el octeto *H1* en los 8 bits más significativos de un registro llamado *H1H2[15:0]*, al tiempo que el valor del desplazamiento de la trama anterior se almacena en el registro de salida *H1H2_ant[9:0]*.

Al activarse *act_H2* se almacena el octeto *H2* en *H1H2[7:0]*. En este momento ya se han capturado los octetos *H1* y *H2* a la espera de que la señal *act_process_pointer* indique que se pueden empezar a procesar los punteros.

La señal *act_process_pointer* se mantendrá activada solo un ciclo de reloj en cada trama. Su valor se hace pasar por un registro de desplazamiento a la izquierda llamado *sec[1:0]* de forma que cuando en un determinado ciclo de reloj se active *act_process_pointer*, en el siguiente *sec[1:0]* valdrá *01b*, y en el siguiente *10b*. Así podemos aprovechar este último registro para secuenciar el procesamiento de los octetos *H1* y *H2*.

Cuándo *sec[0]* vale *1b*, se realizan siete comparaciones paralelamente con los octetos *H1* y *H2*:

- *Comparación 0*: comprueba si *NDF* es normal.
- *Comparación 1*: comprueba si *NDF* está habilitada.
- *Comparación 2*: comprueba si el rango del desplazamiento del puntero es correcto.

- *Comparación 3*: comprueba si la mayoría de bits *I* están invertidos.
- *Comparación 4*: comprueba si la mayoría de bits *D* están invertidos.
- *Comparación 5*: comprueba si el desplazamiento del *VC* de la trama actual es igual al de la trama anterior.
- *Comparación 6*: comprueba si se ha recibido una indicación *AIS*.

Por último, cuando sec[1] vale *1b*, se actualiza la máquina de estados en función del resultado de las comparaciones y del número de veces consecutivas que se produce una determinada indicación.

5.5.2 Módulo *AU_pointers_conc_Rx*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
din[31:0]	E	Datos de entrada
act_H1	E	Señal de captura del octeto *H1*
act_H2	E	Señal de captura del octeto *H2*
act_H1_c	E	Señal de captura de los octetos *H1* concatenados
act_H2_c	E	Señal de captura de los octetos *H2* concatenados
LOP_C	S	Señal *LOP_C*
AIS_C	S	Señal *AIS_C*

Tabla 5.19: Interfaz del módulo *AU_pointers_conc_Rx*

- Funciones

El módulo *AU_pointers_conc_Rx* procesa los punteros concatenados.

- Funcionamiento

La figura 5.13 muestra un diagrama de bloques de este módulo. Los punteros se procesan según la máquina de estados vista en el apartado 2.2.6.2, figura 2.31.

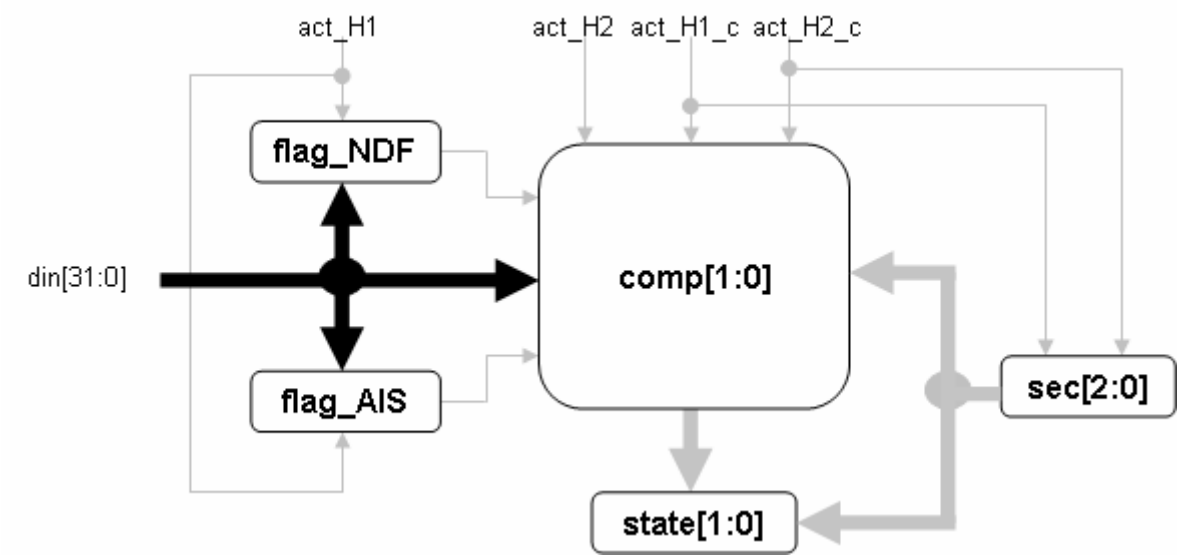


Figura 5.13: Diagrama de bloques del módulo *AU_pointers_conc_Rx*

El procedimiento es similar al del módulo *AU_pointers_H1_H2_Rx*. En este caso los punteros no se capturan. Las comparaciones se van realizando según van llegando los octetos.

Lo primero que se hace es comprobar que los tres primeros octetos concatenados *H1* que se reciben indican puntero concatenado correcto, *AIS* o puntero incorrecto. Esto se debe hacer al activarse la señal *act_H1*, ya que en ese momento habrá en *din[31:0]* el octeto *H1* perteneciente al puntero propiamente dicho y los tres primeros octetos *H1* pertenecientes a los punteros concatenados.

Seguidamente se realiza la comparación 0 que consiste en comprobar si el resto de octetos *H1* concatenados indican lo mismo que los tres primeros. Si no es así se trata de un puntero incorrecto.

Luego se realiza la comparación 1, que comprueba si los octetos *H2* concatenados indican puntero incorrecto.

Al desactivarse *act_H2_c* se actualiza la máquina de estados en función del resultado de las comparaciones y del número de veces consecutivas que se produce una determinada indicación.

5.6 Módulo *POH_Rx*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
din[31:0]	E	Datos de entrada
act_VC	E	Señalización del *VC*
act_J1	E	Señal de captura del octeto *J1*
act_B3	E	Señal de captura del octeto *B3*
act_C2	E	Señal de captura del octeto *C2*
act_G1	E	Señal de captura del octeto *G1*
act_F2	E	Señal de captura del octeto *F2*
act_H4	E	Señal de captura del octeto *H4*
act_F3	E	Señal de captura del octeto *F3*
act_K3	E	Señal de captura del octeto *K3*
act_N1	E	Señal de captura del octeto *N1*
HO_PATH_RDI	S	Señal de error *HO-PATH-RDI*
VC_AIS	S	Señal de error *VC-AIS*
dUNEQ	S	Señal de error *dUNEQ*
error_B3	S	Señal de error *error_B3*
read_J1	S	Datos del octeto *J1*
read_C2	S	Datos del octeto *C2*
read_G1	S	Datos del octeto *G1*
read_F2	S	Datos del octeto *F2*
read_H4	S	Datos del octeto *H4*
read_F3	S	Datos del octeto *F3*
read_K3	S	Datos del octeto *K3*
read_N1	S	Datos del octeto *N1*

Tabla 5.20: Interfaz del módulo *POH_Rx*

- Estructura interna

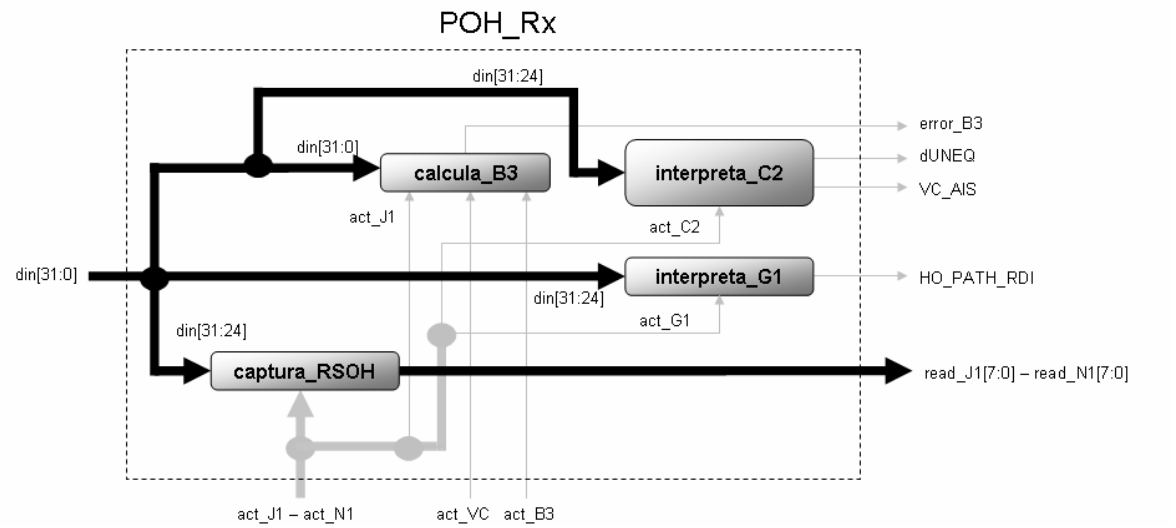


Figura 5.14: Estructura interna del módulo *POH_Rx*

- Funciones

  Este módulo es el encargado de realizar el cálculo y comparación del octeto *B3*, detectar y generar los errores *VC-AIS*, *dUNEQ* y *HO-PATH-RDI* y extraer los octetos *J1*, *C2*, *G1*, *F2*, *H4*, *F3*, *K3* y *N1*.

- Funcionamiento

  El módulo *calcula_B3* realiza el cálculo del octeto *B3* del *VC* actual para compararlo con el recibido en el siguiente *VC*.

  El módulo *interpreta_C2* activa las señales *VC-AIS* y *dUNEQ* en función del valor del octeto *C2* que se va recibiendo en cada *VC*, mientras que el módulo *interpreta_G1* hace lo mismo con la señal *HO-PATH-RDI* en función en este caso del octeto *G1*.

  El módulo *captura_POH* extrae los octetos *J1*, *C2*, *G1*, *F2*, *H4*, *F3*, *K3* y *N1* del *VC*.

En los siguientes apartados se describen con más detalle los módulos que componen el módulo *POH_Rx*.

5.6.1 Módulo *calcula_B3*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
din[31:0]	E	Datos de entrada
input_B3	E	*B3* de entrada
SOF	E	Comienzo del *VC*
act_VC	E	Señalización del *VC*
act_B3	E	Señal de captura del octeto *B3*
error_B3	S	Señal de error en *B3*

Tabla 5.21: Interfaz del módulo *calcula_B3*

- Funciones

  El módulo *calcula_B3* se encarga de realizar el cálculo y comparación del octeto *B3*.

- Funcionamiento

  Su funcionamiento es similar al del módulo *calcula_B1*, sólo que en este caso la trama ya está desaleatorizada y el cálculo sólo se realiza con las palabras correspondientes al *VC*, indicado por la señal *act_VC*.

5.6.2 Módulo *captura_POH*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
din[7:0]	E	Datos de entrada
SOF	E	Comienzo de la trama
act_J1	E	Señal de captura del octeto *J1*
act_C2	E	Señal de captura del octeto *C2*
act_G1	E	Señal de captura del octeto *G1*
act_F2	E	Señal de captura del octeto *F2*
act_H4	E	Señal de captura del octeto *H4*
act_F3	E	Señal de captura del octeto *F3*
act_K3	E	Señal de captura del octeto *K3*
act_N1	E	Señal de captura del octeto *N1*
read_J1	S	Datos del octeto *J1*
read_C2	S	Datos del octeto *C2*
read_G1	S	Datos del octeto *G1*
read_F2	S	Datos del octeto *F2*
read_H4	S	Datos del octeto *H4*
read_F3	S	Datos del octeto *F3*
read_K3	S	Datos del octeto *K3*
read_N1	S	Datos del octeto *N1*

Tabla 5.22: Interfaz del módulo *captura_POH*

- Funciones

  El módulo *captura_POH* se encarga de la extracción de los octetos *J1*, *C2*, *G1*, *F2*, *H4*, *F3*, *K3* y *N1*.

- Funcionamiento

  Su funcionamiento es similar al del módulo *captura_RSOH* visto en el apartado 5.3.3.

5.6.3 Módulo *interpreta_C2*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
din[7:0]	E	Datos de entrada
act_C2	E	Señal de captura del octeto *C2*
VC_AIS	S	Señal *VC-AIS*
dUNEQ	S	Señal *dUNEQ*

Tabla 5.23: Interfaz del módulo *interpreta_C2*

- Funciones

Este módulo activa las señales *VC-AIS* y *dUNEQ* en función del valor del octeto *C2* que se va recibiendo en cada *VC*.

- Funcionamiento

Si se reciben cinco *VC* consecutivos con el octeto *C2* a *FFh* se activa la señal *VC-AIS*, desactivándose *dUNEQ* si estaba activa. Por el contrario, si se reciben cinco *VC* consecutivos con *0h* se activa la señal *dUNEQ*, desactivándose *VC-AIS*. Si se reciben cinco *VC* consecutivos con valores distintos a los anteriores se desactivan ambas señales.

5.6.4 Módulo *interpreta_G1*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
din[7:0]	E	Datos de entrada
act_G1	E	Señal de captura del octeto *G1*
HO_PATH_RDI	S	Señal *HO-PATH-RDI*

Tabla 5.24 Interfaz del módulo *interpreta_G1*

- Funciones

  Este módulo activa la señal *HO-PATH-RDI* en función del valor del octeto *G1* que se va recibiendo en cada *VC*.

- Funcionamiento

  El bit 3 del octeto *G1* controla la señal *HO-PATH-RDI*. Si se reciben tres *VC* consecutivos con *1b* se activa la señal *HO-PATH-RDI*, desactivándose si se reciben tres *VC* consecutivas con *0b*.

5.7 Módulo *capturar_oct*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
SOF	E	Comienzo de la trama
read_J0	E	Datos del octeto *J0*
read_E1	E	Datos del octeto *E1*
read_F1	E	Datos del octeto *F1*
read_D1	E	Datos del octeto *D1*
read_D2	E	Datos del octeto *D2*
read_D3	E	Datos del octeto *D3*
read_K1	E	Datos del octeto *K1*
read_K2	E	Datos del octeto *K2*
read_D4	E	Datos del octeto *D4*
read_D5	E	Datos del octeto *D5*
read_D6	E	Datos del octeto *D6*
read_D7	E	Datos del octeto *D7*
read_D8	E	Datos del octeto *D8*
read_D9	E	Datos del octeto *D9*
read_D10	E	Datos del octeto *D10*
read_D11	E	Datos del octeto *D11*
read_D12	E	Datos del octeto *D12*
read_S1	E	Datos del octeto *S1*
read_M1	E	Datos del octeto *M1*
read_E2	E	Datos del octeto *E2*
read_J1	E	Datos del octeto *J1*
read_C2	E	Datos del octeto *C2*
read_G1	E	Datos del octeto *G1*
read_F2	E	Datos del octeto *F2*
read_H4	E	Datos del octeto *H4*
read_F3	E	Datos del octeto *F3*
read_K3	E	Datos del octeto *K3*
read_N1	E	Datos del octeto *N1*
B1	E	Señal de error en *B1*
B2	E	Señal de error en *B2*
MS_RDI	E	Señal *MS-RDI*
AU_LOP	E	Señal *AU-LOP*

Tabla 5.25: Interfaz del módulo *capturar_oct*

Nombre	E/S	Descripción
HO_PATH_RDI	E	Señal de error *HO-PATH-RDI*
dUNEQ	E	Señal de error *dUNEQ*
B3	E	Señal de error en *B3*
int	S	Señal de interrupción
DAT_I[7:0]	E	Vector de datos de entrada del protocolo *Wishbone*
ADR_I[4:0]	E	Vector de dirección del protocolo *Wishbone*
STB_I	E	Señal de selección del protocolo *Wishbone*
CYC_I	E	Señal de ciclo de bus válido del protocolo *Wishbone*
WE_I	E	Señal escritura del protocolo *Wishbone*
ACK_O	S	Señal de fin de un ciclo de bus del protocolo *Wishbone*
DAT_O[7:0]	S	Vector de datos de salida del protocolo *Wishbone*

Tabla 5.25: Interfaz del módulo *capturar_oct*
(continuación)

- Estructura interna

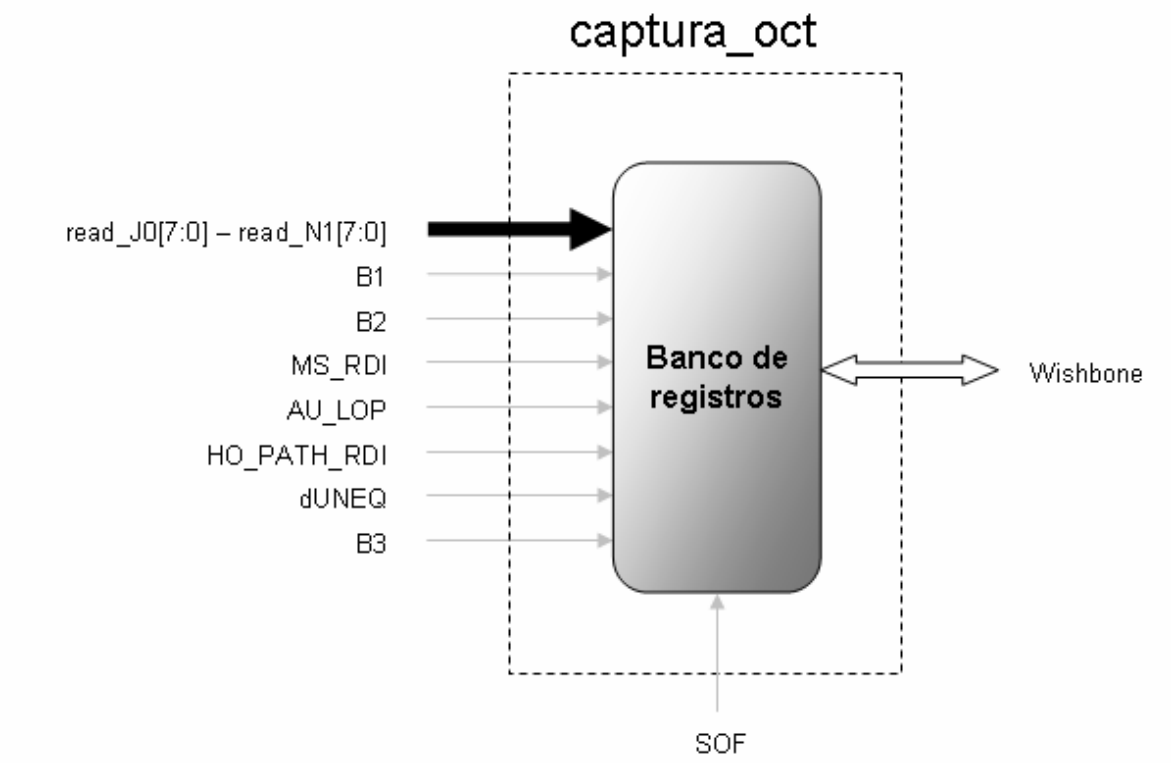


Figura 5.15: Estructura interna del módulo *capturar_oct*

- Funciones

  Este módulo almacena al inicio de cada trama todos los octetos capturados por los módulos anteriores en un banco de registros. Además, a través del protocolo *Wishbone B.3*[3] y actuando como esclavo, entrega a su salida el valor de cualquier octeto que solicite el maestro en cualquier momento.

  Por último genera una señal de interrupción *int*, que no se desactiva hasta que, mediante el protocolo *Wishbone B.3*, el maestro lea el registro interno *flag[6:0]*.

- Funcionamiento

  Cuando se activa la señal *SOF* indicando el comienzo de una nueva trama, el valor los octetos capturados se almacenan en el banco de registros, cada uno de los cuales tiene asociado una dirección como se muestra en la tabla 5.26.

  Para que un sistema externo pueda acceder a cualquiera de estos registros, debe hacerlo a través del protocolo *Wishbone B.3* actuando como maestro.

  Existe un registro interno llamado *flag[6:0]* cuyo formato se muestra en la figura 5.16. Cuando se activa cualquiera de las señales *B1*, *B2*, *MS-RDI*, *AU-LOP*, *HO-PATH-RDI*, *dUNEQ* o *B3*, se activa su bit correspondiente *flag[6:0]* y la señal *int*.

  La señal *int* se desactiva cuando lo haga la señal que la activó o cuando un sistema externo lea el contenido del registro *flag[6:0]* mediante el protocolo *Wishbone B.3*. Esto se hace así porque cuando una señal de error se activa, normalmente permanece así varias tramas consecutivas, y generaría muchas interrupciones seguidas.

  Los bits del registro *flag[6:0]* sólo se desactivarán si lo hacen sus señales correspondientes.

[3] Ver apéndice B para más información sobre el protocolo *Wishbone*.

Dirección	Registro	Lectura/Escritura
0	J0[7:0]	L
1	E1[7:0]	L
2	F1[7:0]	L
3	D1[7:0]	L
4	D2[7:0]	L
5	D3[7:0]	L
6	K1[7:0]	L
7	K2[7:0]	L
8	D4[7:0]	L
9	D5[7:0]	L
10	D6[7:0]	L
11	D7[7:0]	L
12	D8[7:0]	L
13	D9[7:0]	L
14	D10[7:0]	L
15	D11[7:0]	L
16	D12[7:0]	L
17	S1[7:0]	L
18	M1[7:0]	L
19	E2[7:0]	L
20	J1[7:0]	L
21	C2[7:0]	L
22	G1[7:0]	L
23	F2[7:0]	L
24	H4[7:0]	L
25	F3[7:0]	L
26	K3[7:0]	L
27	N1[7:0]	L
28	flag[6:0]	L
29	mascara[6:0]	L/E

Tabla 5.26: Banco de registros

Hay otro registro interno llamado *mascara[6:0]* con el que se pueden ignorar cualquiera de las señales *B1*, *B2*, *MS-RDI*, *AU-LOP*, *HO-PATH-RDI*, *dUNEQ* y/o *B3*, logrando así que la señal *int* sólo se active con determinadas señales. La figura 5.17 muestra el formato de dicho registro.

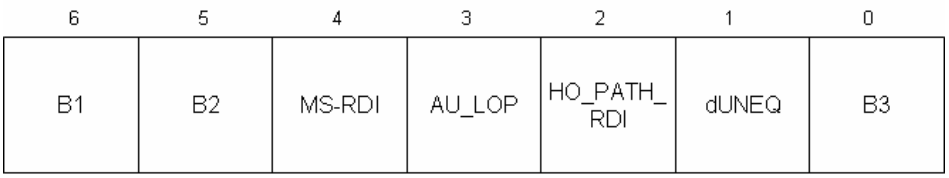


Figura 5.16: Formato del registro *flag[6:0]*

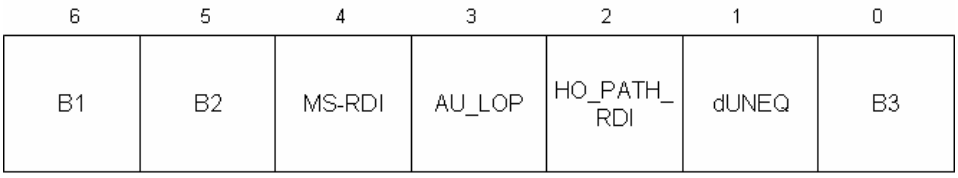


Figura 5.17: Formato del registro *mascara[6:0]*

5.8 Módulo *sec_Rx*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
reset	E	Reset de entrada
LOS	E	Pérdida de señal
mr	S	Reset maestro
din[31:0]	E	Datos de entrada
dout[31:0]	S	Datos de salida
num_pal[13:0]	E	Número de palabra
sinc	E	Señal de sincronismo
AU_LOP	E	Señal *AU_LOP*
inc	E	Señal de incremento de puntero
dec	E	Señal de decremento de puntero
desp[9:0]	E	Valor del desplazamiento del *VC* de la trama actual
desp_ant[9:0]	E	Valor del desplazamiento del *VC* de la trama anterior
start_scr	S	Señal de activación del aleatorizador
SOF_RSOH	S	Comienzo de trama en el módulo *RSOH*
act_B1	S	Señal de captura del octeto *B1*

Tabla 5.27: Interfaz del módulo *sec_Rx*

Nombre	E/S	Descripción
act_J0	S	Señal de captura del octeto *J0*
act_E1	S	Señal de captura del octeto *E1*
act_F1	S	Señal de captura del octeto *F1*
act_D1	S	Señal de captura del octeto *D1*
act_D2	S	Señal de captura del octeto *D2*
act_D3	S	Señal de captura del octeto *D3*
SOF_MSOH	S	Comienzo de trama en el módulo *MSOH*
act_reg	S	Señalización de la tara de regeneración
act_B2	S	Señal de captura del octeto *B2*
act_K1	S	Señal de captura del octeto *K1*
act_K2	S	Señal de captura del octeto *K2*
act_D4	S	Señal de captura del octeto *D4*
act_D5	S	Señal de captura del octeto *D5*
act_D6	S	Señal de captura del octeto *D6*
act_D7	S	Señal de captura del octeto *D7*
act_D8	S	Señal de captura del octeto *D8*
act_D9	S	Señal de captura del octeto *D9*
act_D10	S	Señal de captura del octeto *D10*
act_D11	S	Señal de captura del octeto *D11*
act_D12	S	Señal de captura del octeto *D12*
act_S1	S	Señal de captura del octeto *S1*
act_M1	S	Señal de captura del octeto *M1*
act_E2	S	Señal de captura del octeto *E2*
act_H1	S	Señal de captura del octeto *H1*
act_H2	S	Señal de captura del octeto *H2*
act_H1_c	S	Señal de captura de los octetos *H1* concatenado
act_H2_c	S	Señal de captura de los octetos *H2* concatenado
act_pointer	S	Señal de procesamiento de punteros
act_J1	S	Señal de captura del octeto *J1*
act_B3	S	Señal de captura del octeto *B3*
act_C2	S	Señal de captura del octeto *C2*
act_G1	S	Señal de captura del octeto *G1*
act_F2	S	Señal de captura del octeto *F2*
act_H4	S	Señal de captura del octeto *H4*
act_F3	S	Señal de captura del octeto *F3*
act_K3	S	Señal de captura del octeto *K3*
act_N1	S	Señal de captura del octeto *N1*
act_VC	S	Señalización del *VC*
act_datos	S	Señalización de datos
SOF_cap	S	Comienzo de trama en el módulo *captura*

Tabla 5.27: Interfaz del módulo *sec_Rx*
(continuación)

- Estructura interna

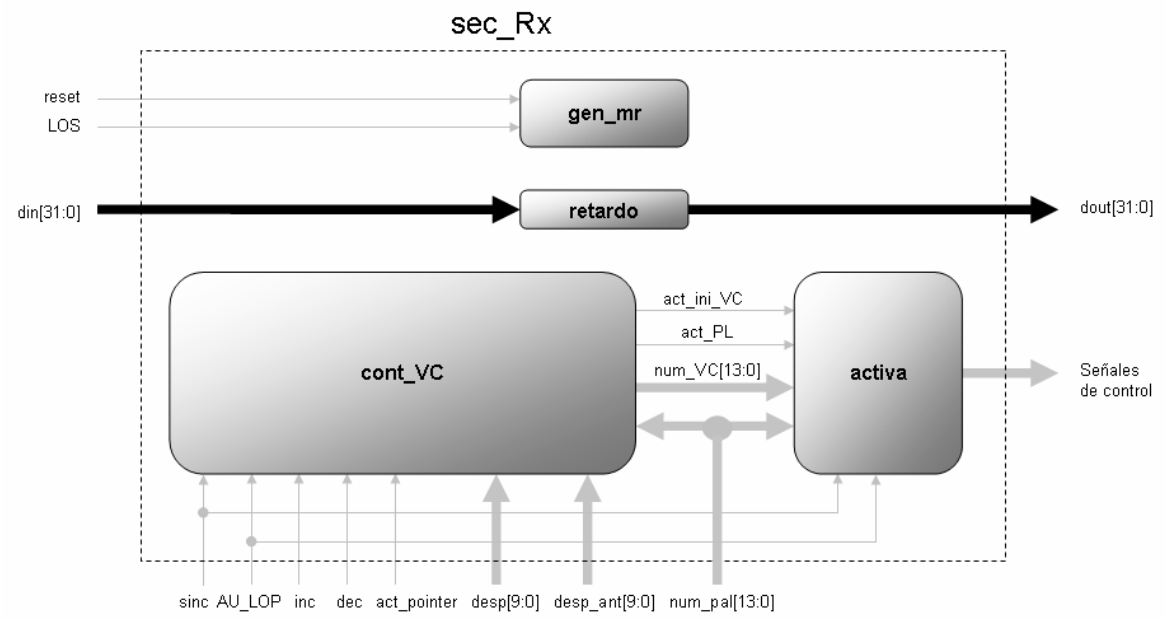


Figura 5.18: Estructura interna del módulo *sec_Rx*

- Funciones

  Este módulo se encarga de generar todas las señales de sincronización de los módulos que procesan la tara en el receptor. Además genera la señal de *mr* para todo el sistema.

- Funcionamiento

  El módulo *activa* es el que genera todas las señales de sincronización, a partir del contador del módulo *entrada* y del contador del módulo *cont_VC*. Éste último módulo lleva la cuenta de las palabras correspondientes al *VC* que se van recibiendo.

  Por último, el módulo *gen_mr* genera la señal *mr* y el módulo *retardo* retarda un ciclo de reloj los datos de entrada.

En los siguientes apartados se describen con más detalle los módulos que componen el módulo *sec_Rx*.

5.8.1 Módulo *activa*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
num_pal[13:0]	E	Número de palabra dentro de la trama
num_VC[13:0]	E	Número de palabra dentro del *VC*
sinc	E	Señal de sincronismo
AU_LOP	E	Señal *AU-LOP*
act_ini_VC	E	Comienzo del *VC*
act_PL	E	Señalización de carga útil
start_scr	S	Señal de activación del aleatorizador
SOF_RSOH	S	Comienzo de trama en el módulo *RSOH*
act_B1	S	Señal de captura del octeto *B1*
act_J0	S	Señal de captura del octeto *J0*
act_E1	S	Señal de captura del octeto *E1*
act_F1	S	Señal de captura del octeto *F1*
act_D1	S	Señal de captura del octeto *D1*
act_D2	S	Señal de captura del octeto *D2*
act_D3	S	Señal de captura del octeto *D3*
SOF_MSOH	S	Comienzo de trama en el módulo *MSOH*
act_reg	S	Señalización de la tara de regeneración
act_B2	S	Señal de captura del octeto *B2*
act_K1	S	Señal de captura del octeto *K1*
act_K2	S	Señal de captura del octeto *K2*
act_D4	S	Señal de captura del octeto *D4*
act_D5	S	Señal de captura del octeto *D5*
act_D6	S	Señal de captura del octeto *D6*
act_D7	S	Señal de captura del octeto *D7*
act_D8	S	Señal de captura del octeto *D8*
act_D9	S	Señal de captura del octeto *D9*
act_D10	S	Señal de captura del octeto *D10*

Tabla 5.28: Interfaz del módulo *activa*

Nombre	E/S	Descripción
act_D11	S	Señal de captura del octeto *D11*
act_D12	S	Señal de captura del octeto *D12*
act_S1	S	Señal de captura del octeto *S1*
act_M1	S	Señal de captura del octeto *M1*
act_E2	S	Señal de captura del octeto *E2*
act_H1	S	Señal de captura del octeto *H1*
act_H2	S	Señal de captura del octeto *H2*
act_H1_c	S	Señal de captura de los octetos *H1* concatenado
act_H2_c	S	Señal de captura de los octetos *H2* concatenado
act_pointer	S	Señal de procesamiento de punteros
act_J1	S	Señal de captura del octeto *J1*
act_B3	S	Señal de captura del octeto *B3*
act_C2	S	Señal de captura del octeto *C2*
act_G1	S	Señal de captura del octeto *G1*
act_F2	S	Señal de captura del octeto *F2*
act_H4	S	Señal de captura del octeto *H4*
act_F3	S	Señal de captura del octeto *F3*
act_K3	S	Señal de captura del octeto *K3*
act_N1	S	Señal de captura del octeto *N1*
act_VC	S	Señalización del *VC*
act_datos	S	Señalización de datos
SOF_cap	S	Comienzo de trama en el módulo *captura*

Tabla 5.28: Interfaz del módulo *activa*
(continuación)

- Funciones

  El módulo *activa* es el que genera todas las señales de sincronización.

- Funcionamiento

  A partir del contador del módulo *entrada* genera todas las señales de sincronización, excepto las correspondientes al módulo *POH_Rx*. Para éste último necesita otro contador, el del módulo *cont_VC*, ya que el *VC* puede desplazarse dentro de la carga útil de la trama y por lo tanto los octetos de la *POH* y los datos varían su posición con respecto a la misma.

Para generar las señales el sistema debe estar sincronizado, indicado mediante la señal de entrada *sinc*. Además las señales correspondientes al módulo *POH_Rx* necesitan que el módulo *AU_P_Rx* ya haya encontrado el desplazamiento del *VC*, indicándolo mediante la señal *AU_LOP*.

Las entradas *act_ini_VC* y *act_PL*, que se verán con el módulo *cont_VC*, son necesarias para generar las señales:

- *act_J1*: que vale tanto para indicar el inicio del *VC* y que se está recibiendo el octeto *J1*.

- *act_datos*: que indica que las palabras que están la salida del receptor son los datos.

- *act_VC*: que indica que las palabras que se están recibiendo pertenecen al contenedor virtual.

En la generación de las palabras, hay que tener en cuenta el retardo de los datos al pasar por los distintos módulos del receptor.

5.8.2 Módulo *cont_VC*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
num_pal[13:0]	E	Número de palabra dentro de la trama
sinc	E	Señal de sincronismo
AU_LOP	E	Señal *AU-LOP*
inc	E	Señal de incremento de puntero
dec	E	Señal de decremento de puntero
act_pointer	E	Señal de procesamiento de punteros
desp[9:0]	E	Valor del desplazamiento del *VC* de la trama actual
desp_ant[9:0]	E	Valor del desplazamiento del *VC* de la trama anterior
act_ini_VC	S	Comienzo del *VC*
act_PL	S	Señalización de carga útil
num_VC[13:0]	S	Número de palabra dentro del *VC*

Tabla 5.29: Interfaz del módulo *cont_VC*

- Funciones

  Este módulo lleva la cuenta de las palabras que se van recibiendo del *VC* actual. Además genera las señales *act_ini_VC*, que indica el comienzo del *VC*, y la señal *act_PL*, que indica que las palabras que se están recibiendo pertenecen a la carga útil.

- Funcionamiento

  La señal *act_PL* se activa a partir del contador del módulo *entrada* y de las señales de entrada *inc* y *dec*.

  Cuando el módulo *AU_P_Rx* determina el desplazamiento del *VC*, un contador descendente se inicializa en función del valor del desplazamiento y se pone en marcha. Cuando llega a cero se activa la señal *act_ini_VC* indicando el comienzo del *VC*.

En este instante se pone en marcha otro contador, *num_VC[13:0]*, que es el que lleva la cuenta de las palabras del *VC* actual que se van recibiendo.

5.8.3 Módulo *retardo*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
din[31:0]	E	Datos de entrada
dout[31:0]	S	Datos de salida

Tabla 5.30: Interfaz del módulo *retardo*

- Funciones

Este módulo realiza un retardo de un ciclo de reloj en los datos entrantes. Esto se realiza porque cuando el sistema se ha sincronizado, a la entrada *din[31:0]* de este módulo se encuentra ya el octeto *J0*, y la señal *act_J0* no ha sido activada un ciclo de reloj antes porque el sistema no se había sincronizado.

- Funcionamiento

Las palabras entrantes se hacen pasar por una salida registrada de 32 bits *dout[31:0]*.

5.8.4 Módulo *gen_mr*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
reset	E	Reset de entrada
LOS	E	Pérdida de señal
mr	S	Reset maestro

Tabla 5.31: Interfaz del módulo *gen_mr*

- Funciones

  El módulo *gen_mr* genera la señal de *mr* para todo el sistema.

- Funcionamiento

  La señal *mr* se genera si alguna de las señales *reset* o *LOS* está activa.

5.9 Relación de señales y alarmas

BER: Degradación de la señal. Está causada por exceder un umbral *BER* preseleccionado (de 10^{-5} a 10^{-9}). Señal producida por el sistema de control.

LOS: Pérdida de señal. Es declarada cuando no se detectan transiciones en la entrada previa a la desaleatorización. Esta señal la proporciona la subcapa del medio físico.

OOF: Fuera de trama. Se produce cuando se pierde la delineación de trama. Vuelve a cero tan pronto se recupere la delineación.

LOF: Pérdida de trama. Se pone a uno cuando se produce *OOF* durante 2 ms. Se desactivará cuando *OOF* permanezca desactivada durante el mismo periodo.

MS-AIS: Señal de indicación de alarma de sección de multiplexación. Se envía al destino para indicar que se ha detectado un fallo hacia el origen. Se produce cuando se detectan en tres tramas consecutivas los bits de 2-0 de *K2* con el valor de *111b*. Se desactiva si se reciben 3 tramas consecutivas con un valor diferente de estos bits.

MS-RDI: Indicación de defecto remoto de la sección de multiplexación. Se utiliza para devolver una indicación al extremo transmisor de que el receptor ha detectado un fallo de sección entrante, o se está recibiendo una *MS-AIS*. Se produce cuando se detectan en tres tramas consecutivas los bits de 2-0 de *K2* con el valor *110b*. Se desactiva si se reciben 3 tramas consecutivas con un valor diferente en estos bits.

AU-LOP: Pérdida de puntero. Se produce siempre que la máquina de estados de los punteros concatenados transite al estado *AISC* o al estado *LOPC*, o bien, si la máquina de punteros *AU-4* transita al estado *AIS* o *LOP*.
AU-AIS: Señal de indicación de alarma de puntero. Se produce cuando la máquina de punteros *AU-4* transita al estado *AIS* y cuando la máquina de punteros concatenados transita al estado *AISC*.

dUNEQ: No equipado, es decir, la sección está completa pero no hay configurada una conexión. Se activa cuando se reciben 5 tramas consecutivas con *C2* todo a cero. Se desactiva si se recibe *C2* con otro valor en 5 tramas consecutivas.

VC-AIS: Señal de indicación de alarma de contenedor virtual. Se activa cuando se reciben 5 tramas consecutivas con *C2* todo a unos. Se desactiva si se recibe *C2* con otro valor en 5 tramas consecutivas.

HO-PATH-RDI: Indicación de defecto remoto del *VC-4*. Se activa cuando se reciben 3 tramas consecutivas con el bit 3 de *G1* a 1. Se desactiva si se reciben 3 tramas consecutivas con dicho bit a 0.

Capítulo 6

Verificación

Todo diseño se compone de tres fases fundamentales: especificación, implementación y verificación. La especificación se lleva a cabo definiendo las distintas tareas que realiza el sistema desde un punto de vista global, para posteriormente, pasar a desglosar cada una de sus partes. A continuación, se implementa cada parte y se procede a su verificación. En este último paso, el orden seguido es inverso a la especificación, ya que se comienza desde los bloques o módulos que componen las unidades fundamentales, hasta llegar a la verificación del sistema completo. En este capítulo se elegirán un conjunto de vectores con el objetivo de comprobar que las especificaciones iniciales y la implementación del diseño coinciden. Para ello se verifican en primer lugar los módulos de nivel inferior, continuando por los módulos de niveles intermedios hasta llegar al sistema completo.

6.1 Descripción General

La verificación individual de cada uno de los módulos supone un trabajo que supera considerablemente los objetivos de este proyecto. Normalmente, la verificación de un módulo debe cubrir todos los modos de funcionamiento y los detalles más insignificantes, es decir, se debe invertir un gran esfuerzo en el test de cada módulo. Usualmente, cuando este módulo se ha introducido en el sistema permanece la mayor parte del tiempo funcionando en alguna configuración concreta, por lo que el tiempo invertido en los casos más extraños de funcionamiento no se ve recompensado.

Por todo lo dicho anteriormente y debido a las limitaciones propias de un Proyecto Fin de Carrera, se ha estimado realizar la verificación simulada de los módulos de nivel inferior, y pasar después directamente a la verificación hardware del sistema completo.

Para cada uno de los tests definidos se presenta su objetivo, el procedimiento seguido para su verificación, así como el resultado esperado del test y, en función de estos, el análisis de los resultados obtenidos.

6.2 Verificación a nivel de módulo

Cómo ya se describió anteriormente, la verificación a nivel de módulo se ha realizado solamente mediante simulación.

En los siguientes apartados se incluyen la estructura general de los tests, un resumen de todos los tests realizados y dos ejemplos de los mismos.

6.2.1 Estructura General

La figura 6.1 muestra la estructura general de un test a nivel de módulo. Consta de tres módulos:

- *top*: se encarga de conectar entre sí el módulo bajo test con el módulo g*enerador*. Además genera la señal de reloj *clk* y la de *reset*. Por último contiene un contador de ciclos de reloj, o un contador de tramas y otro de palabras, con el fin de tener una referencia del número de ciclos de reloj transcurridos o del número de tramas y palabras dentro de la trama generadas, respectivamente.

- *generador*: genera todas las señales de estímulo (salvo *clk* y *reset*) del módulo bajo test en función del número de ciclos transcurridos o del número de tramas y palabras generadas. Estas incluyen señales de control y tramas.

- Módulo bajo test: es el módulo que se pretende verificar.

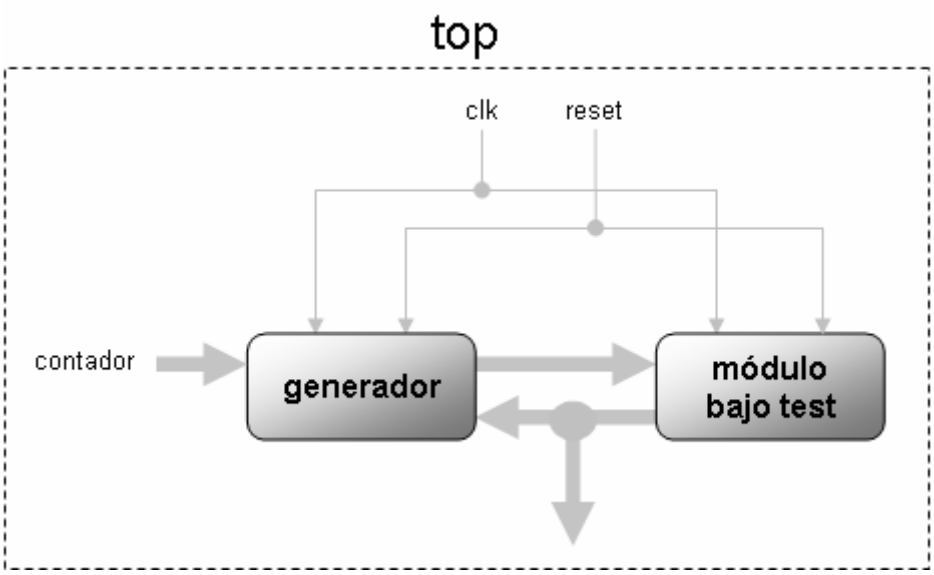


Figura 6.1: Estructura general de un test a nivel de módulo

6.2.2 Tests Realizados

En la tabla 6.1 vienen resumidos los tests realizados a nivel de módulo. Consta de tres columnas:

- Tests: identificador del test.
- Módulo: indica el nombre del módulo a testear.
- Objetivo: describe qué es lo que se pretende verificar.

Tests	Módulo	Objetivo
001-032	*desplaza*	Comprobar que los datos se alinean
033	*estados_entrada*	Comprobar la correcta transición entre los estados de la máquina de delineación de trama, así como la activación/desactivación de las señales *OOF* y *sinc*
034	*contador*	Comprobar el correcto funcionamiento del módulo *contador*
035	*activa_LOF*	Comprobar la activación/desactivación de la señal *LOF*
036	*aleator*	Comprobar la desaleatorización de la trama entrante
037	*calcula_B1*	Comprobar el cálculo de *B1* y la activación de la señal *error_B1*
038	*captura_RSOH*	Comprobar la captura de los octetos de la *RSOH*
039	*interpreta_K2*	Comprobar la activación/desactivación de las señales *MS-AIS* y *MS-RDI*
040	*calcula_B2*	Comprobar el cálculo de *B2* y la activación de la señal *error_B2*
041	*datos_salida*	Comprobar el correcto funcionamiento del módulo *datos_salida*
042	*captura_MSOH*	Comprobar la captura de los octetos de la *MSOH*
043	*AU_pointers_H1_H2_Rx*	Comprobar la correcta transición entre los estados de la máquina de interpretación de los punteros, así como la activación de las señales *LOP_P*, *AIS_P*, *inc* y *dec* y el valor del desplazamiento obtenido

Tabla 6.1: Tests a nivel de módulo

Tests	Módulo	Objetivo
044	*AU_pointers_conc_Rx*	Comprobar la correcta transición entre los estados de la máquina de interpretación de los punteros concatenados, así como la activación de las señales *LOP_C* y *AIS_C*
045	*calcula_B3*	Comprobar el cálculo de *B3* y la activación de la señal *error_B3*
046	*interpreta_G1*	Comprobar la activación/desactivación de las señal *HO-PATH-RDI*
047	*interpreta_C2*	Comprobar la activación/desactivación de las señales *VC-AIS* y *dUNEQ*
048	*captura_POH*	Comprobar la captura de los octetos de la *POH*
049	*capturar_oct*	Comprobar la captura de todos octetos
050	*capturar_oct*	Comprobar la lectura de los octetos capturados mediante el protocolo *Wishbone*
051	*capturar_oct*	Comprobar la activación/desactivación de la señal *int* y la escritura en el registro *mascara[7:0]*
052	*gen_mr*	Comprobar la activación/desactivación de la señal *mr*
053	*retardo*	Comprobar el correcto funcionamiento del módulo *retardo*
054	*cont_VC*	Comprobar la activación/desactivación de la señal *act_PL*
055-056	*cont_VC*	Comprobar la activación/desactivación de la señal *act_ini_VC* y el funcionamiento del contador *num_VC[13:0]*
057	*activa*	Comprobar la activación/desactivación de todas las señales de sincronización

Tabla 6.1: Tests a nivel de módulo
(continuación)

6.2.3 Ejemplos de tests

Debido a la gran cantidad de tests realizados a nivel de módulo (57 en total), sólo se va a incluir la descripción de dos ejemplos significativos en esta memoria, donde queda claro el procedimiento seguido para la verificación de cada módulo, ya que, en caso de incluirlos todos, el tamaño de esta memoria aumentaría considerablemente.

La totalidad de los tests se incluyen en el *CD-ROM* que se adjunta[4].

[4] Ver apéndice A para más información sobre el contenido del *CD-ROM*.

En el primero de los tests, el *Test 033*, se comprueba el correcto funcionamiento del módulo *estados_entrada*, muy importante porque de él dependen el resto de módulos, ya que se encarga de sincronizar el sistema con las tramas entrantes.

El otro test, el *Test 043*, verifica el funcionamiento del módulo *AU_pointers_H1_H2_Rx*, muy importante también ya que se encarga de obtener el comienzo del *VC* dentro de cada trama entrante, a partir del cual se puede identificar que parte de la trama forma parte del contenedor y cual no, pudiendo extraer este último.

6.2.3.1 Ejemplo 1: Test 033

Módulo bajo test: *estados_entrada*.

Objetivos:

- Comprobar la correcta transición entre los estados de la máquina de delineación de trama.

- Comprobar la activación/desactivación de las señales *OOF* y *sinc*.

Procedimiento:

- El módulo *generador* generará tramas de 64 palabras con el formato de la figura 6.2. Las primeras doce palabras son octetos *A1* y las 12 siguientes son los *A2*. el resto es un valor constante.

- Unas tramas se van a generar correctas, es decir, con todos los octetos *A1* y *A2* correctos, y otras erróneas.

- En la tabla 6.2 se indica qué tramas se van a generar correctas y cuáles no.

Resultados esperados:

- Hay que comprobar los distintos estados por los que pasa la máquina de estados, así como la activación y desactivación de las señales *OOF* y *sinc*.

- La tabla 6.2 muestra los resultados esperados.

A1 A1 A1 A1	A1 A1 A1 A1	A1 A1 A1 A1	A1 A1 A1 A1	A1 A1 A1 A1	A1 A1 A1 A1	A1 A1 A1 A1	A1 A1 A1 A1
A1 A1 A1 A1	A1 A1 A1 A1	A1 A1 A1 A1	A1 A1 A1 A1	A2 A2 A2 A2	A2 A2 A2 A2	A2 A2 A2 A2	A2 A2 A2 A2
A2 A2 A2 A2	A2 A2 A2 A2	A2 A2 A2 A2	A2 A2 A2 A2	A2 A2 A2 A2	A2 A2 A2 A2	A2 A2 A2 A2	A2 A2 A2 A2
cte	cte	cte	cte	cte	cte	cte	cte
cte	cte	cte	cte	cte	cte	cte	cte
cte	cte	cte	cte	cte	cte	cte	cte
cte	cte	cte	cte	cte	cte	cte	cte
cte	cte	cte	cte	cte	cte	cte	cte

Figura 6.2: Formato de la trama

ESTÍMULOS		RESULTADOS ESPERADOS		
Tramas	**Correctas/Erróneas**	**estado**	**OOF**	**sinc**
Inicialmente		0	1	0
0-1	C	1	0	0
2-14	C	2	0	1
15-16	E	2	0	1
17-20	C	2	0	1
20-24	E	2	0	1
25-29	E	0	1	0
30-31	C	1	0	0
32-33	E	0	1	0
34-35	C	1	0	0
36-49	C	2	0	1
50-51	E	2	0	1
Resto	C	2	0	1

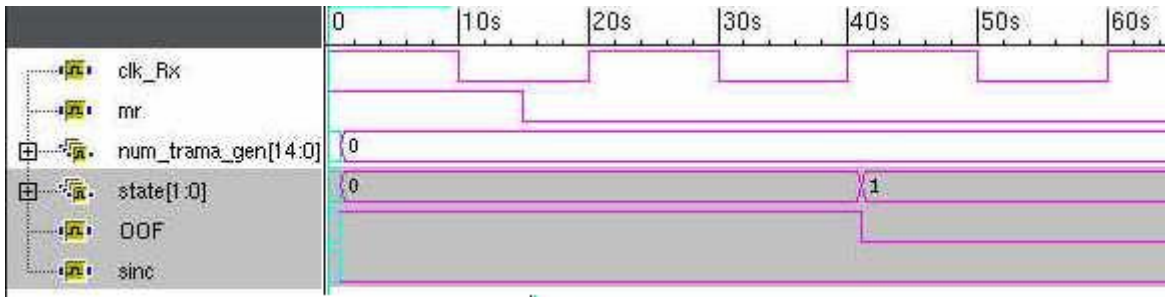
Tabla 6.2: Estímulos y resultados esperados

Resultados obtenidos:

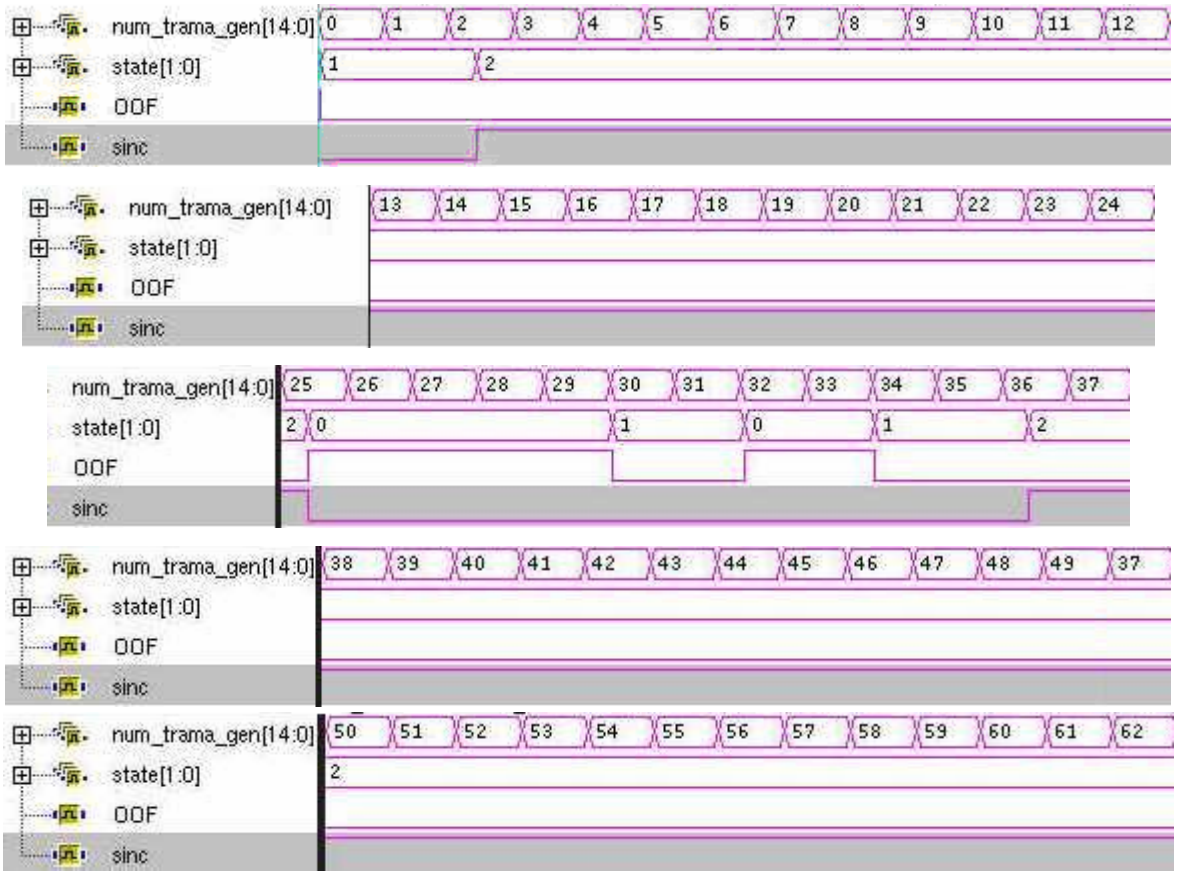
- Se muestran en la figura 6.3.

- La variable *num_trama_gen[14:0]* nos indica el número de tramas que se van generando.

- La variable *state[1:0]* nos indica el estado en que se encuentra el módulo.



(a)



(b)

Figura 6.3: a) Valores después de un reset. b) Resultados obtenidos entre las tramas 0 y 62.

Análisis de los resultados:

- Observando los resultados obtenidos, podemos ver como el módulo, estando en el estado 0, al recibir una trama correcta pasa al estado 1 (tramas 0, 30 y 34).

- Al recibir 3 tramas consecutivas correctas, estando primero en el estado 0 y después en el 1, pasa al estado 2 (tramas 2 y 36).

- Al recibir 6 tramas consecutivas erróneas, estando en el estado 2, pasa al estado 0 (trama 25).

- Al recibir 1 trama errónea, estando en el estado 1, pasa al estado 0 (trama 32).

- La señal OOF sólo se activa cuando el módulo se encuentra en el estado 0.

- La señal sinc sólo se activa cuando el módulo se encuentra en el estado 1.

Conclusión: Comparando los resultados obtenidos con los esperados, se puede decir que el módulo cumple con las especificaciones.

6.2.3.2 Ejemplo 2: Test 043

Módulo bajo test: *AU_pointers_H1_H2_Rx*.

Objetivos:

- Comprobar la correcta transición entre los estados de la máquina de interpretación de los punteros.

- Comprobar la activación de las señales *LOP_P*, *AIS_P*, *inc* y *dec*.

- Comprobar el valor del desplazamiento obtenido.

Procedimiento:

- El módulo *generador* generará tramas de 16 palabras con el formato de la figura 6.4. El octeto *H1* se inserta en los 8 bits más significativos de la palabra de trama número 1 y el *H2* en los de la número 2. El resto de la trama es un valor constante.

- El módulo *generador* también activará y desactivará las señales *act_H1* y *act_H2* y *act_process_pointer*.

- Se van a transmitir distintas indicaciones a través de los punteros.

- La tabla 6.3 muestra la indicación y el desplazamiento transmitidos en cada trama.

Resultados esperados:

- Hay que comprobar los distintos estados por los que pasa la máquina de estados, así como la activación y desactivación de las señales *LOP_P*, *AIS_P*, *inc* y *dec*, y el valor del desplazamiento obtenido cuando el sistema se encuentra en el estado 0.

- La tabla 6.3 muestra los resultados esperados.

cte	**H1,** 000000h	**H2,** 000000h	cte
cte	cte	cte	cte
cte	cte	cte	cte
cte	cte	cte	cte

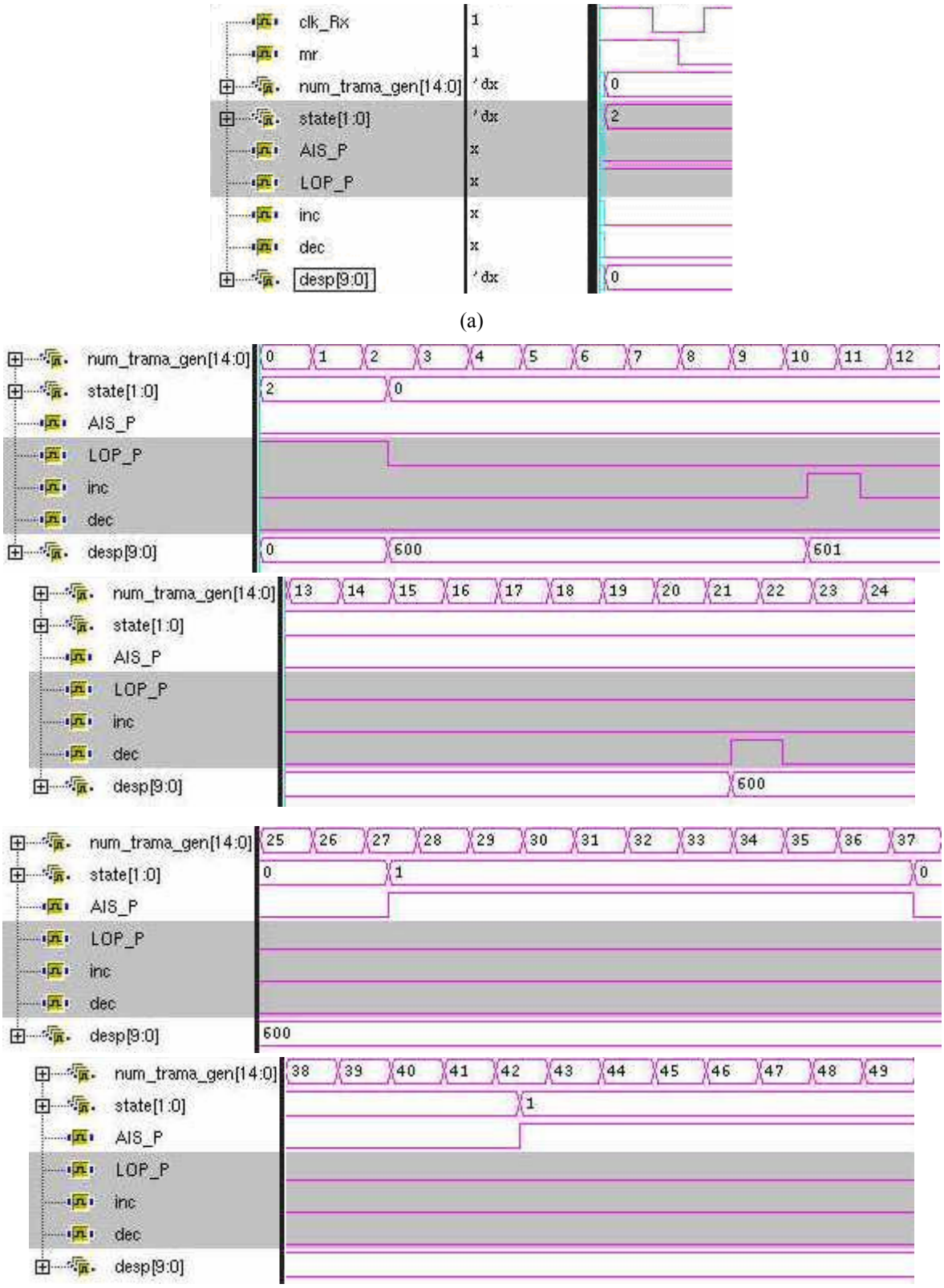
Figura 6.4: Formato de la trama

ESTÍMULOS			RESULTADOS ESPERADOS					
Tramas	**Indicación**	**Desplaza-miento**	**estado**	**LOP_P**	**AIS_P**	**inc**	**dec**	**Desplaza-miento**
Inicialmente			2	1	0	0	0	
0-1	*norm_point*	600	2	1	0	0	0	
2-9	*norm_point*	600	0	0	0	0	0	600
10	*Inc_ind*		0	0	0	1	0	601
11-20	*norm_point*	601	0	0	0	0	0	601
21	*Dec_ind*		0	0	0	0	1	600
22-24	*norm_point*	600	0	0	0	0	0	600
25-26	*AIS_ind*		0	0	0	0	0	600
27-34	*AIS_ind*		1	0	1	0	0	
35-36	*norm_point*	600	1	0	1	0	0	
37-39	*norm_point*	600	0	0	0	0	0	600
40-41	*AIS_ind*		0	0	0	0	0	600
42-49	*AIS_ind*		1	0	1	0	0	
50	*NDF_enable*	500	0	0	0	0	0	500
51-60	*norm_point*	500	0	0	0	0	0	500
61	*NDF_enable*	600	0	0	0	0	0	600
62-64	*norm_point*	600	0	0	0	0	0	600
65-71	*NDF_enable*	500	0	0	0	0	0	500
72-74	*NDF_enable*	500	2	1	0	0	0	500
75-76	*AIS_ind*		2	1	0	0	0	
77-79	*AIS_ind*		1	0	1	0	0	
80-86	*inv_point*		1	0	1	0	0	
87-89	*inv_point*		2	1	0	0	0	
90-91	*norm_point*	600	2	1	0	0	0	
92-99	*norm_point*	600	0	0	0	0	0	600
100-106	*inv_point*		0	0	0	0	0	600
107-109	*inv_point*		2	1	0	0	0	
110-111	*norm_point*	632	2	1	0	0	0	
112-119	*norm_point*	632	0	0	0	0	0	632
120-121	*norm_point*	600	0	0	0	0	0	632
Resto	*norm_point*	600	0	0	0	0	0	600

Tabla 6.3: Estímulos y resultados esperados

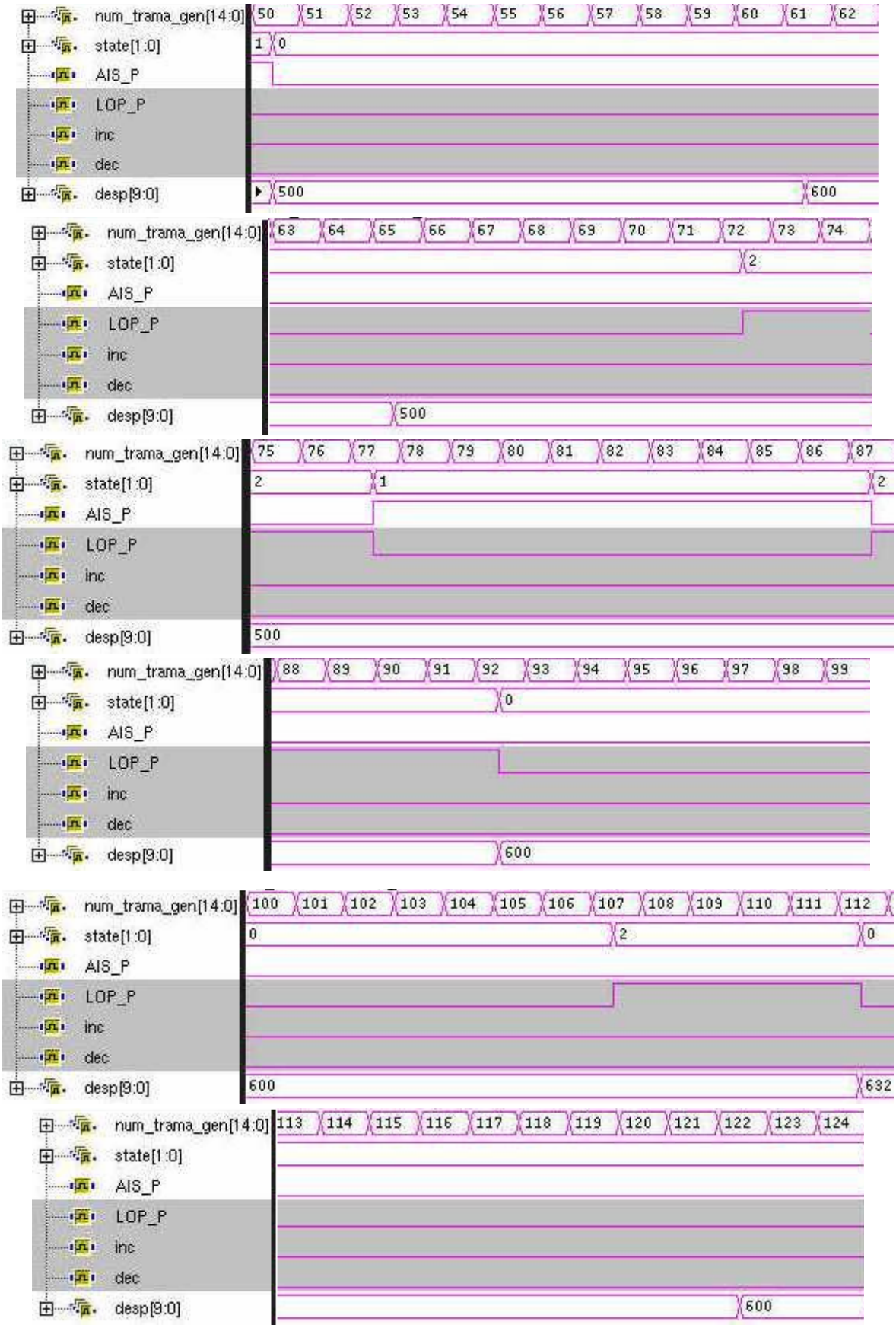
Resultados obtenidos:

- Se muestran en la figura 6.5.



(a)

Figura 6.5: a) Valores después de un reset. b) Resultados obtenidos entre las tramas 0 y 49.



(b)

Figura 6.5: b) Resultados obtenidos entre las tramas 50 y 124.

(continuación)

- La variable *num_trama_gen[14:0]* nos indica el número de tramas que se van generando.

- La variable *state[1:0]* nos indica el estado en que se encuentra el módulo.

- La variable *desp[9:0]* nos indica el valor del desplazamiento obtenido cuando el módulo se encuentra en el estado 0.

Análisis de los resultados:

- Observando los resultados obtenidos, podemos ver como el módulo pasa al estado 0 en los siguientes casos:

  - Al recibir tres indicaciones consecutivas *norm_point*, estando en el estado 2 (tramas 2, 92 y 112).

  - Al recibir tres indicaciones consecutivas *norm_point*, estando en el estado 1 (trama 37).

  - Al recibir una indicación *NDF_enable*, estando en el estado 1 (trama 50).

- El módulo pasa al estado 1 al recibir tres indicaciones consecutivas *AIS_ind*, estando en el estado 0 (tramas 27 y 42) o en el estado 2 (trama 77).

- El módulo pasa al estado 2 en los siguientes casos:

  - Al recibir ocho indicaciones consecutivas *inv_point*, estando en el estado 0 (trama 107) o en el estado 1 (trama 87).

  - Al recibir ocho indicaciones consecutivas *NDF_enable*, estando en el estado 0 (trama 72).

- Los valores del desplazamiento obtenidos en la simulación coinciden con los esperados.

- El valor del desplazamiento cambia, estando el módulo en el estado 0, en los siguientes casos:

  - Al recibir una indicación *NDF_enable* (tramas 50, 61 y 65).

  - Al recibir una indicación *Inc_ind*, incrementando su valor (trama 10).

  - Al recibir una indicación *Dec_ind*, decrementando su valor (trama 21).

  - Al recibir tres indicaciones *norm_point* consecutivas con un valor de desplazamiento distinto al actual (trama 120).

- La señal *AIS_P* sólo se activa cuando el módulo se encuentra en el estado 1.

- La señal *LOP_P* sólo se activa cuando el módulo se encuentra en el estado 2.

- La señal *inc* sólo se activa cuando se recibe una indicación *Inc_ind* (trama 10).

- La señal *dec* sólo se activa cuando se recibe una indicación *Dec_ind* (trama 21).

Conclusión: Comparando los resultados obtenidos con los esperados, se puede decir que el módulo cumple con las especificaciones.

6.3 Verificación a nivel de sistema

Cómo ya se describió anteriormente, a nivel de sistema se ha realizado solamente la verificación hardware.

En los siguientes apartados se incluyen la estructura general de los tests, un resumen de todos los tests realizados y la descripción de los mismos.

6.3.1 Estructura General

La figura 6.6 es una fotografía en la que se puede observar los equipos utilizados para testear el sistema, y la figura 6.7 muestra de forma esquemática la fotografía anterior. Podemos distinguir los siguientes elementos:

- *Placa*: contiene, entre otros elementos, la *FPGA*, los pulsadores, los conmutadores y los puertos *VGA* y *JTAG*.

- *Monitor*: donde se van a visualizar los resultados.

- *Estación de trabajo*: desde donde se va a programar la placa.

- *Cableado*: para conectar el monitor (a través del puerto *VGA*) y el *PC* (desde el puerto paralelo del mismo hasta el *JTAG* de la placa) con la placa, así como alimentar a la misma.

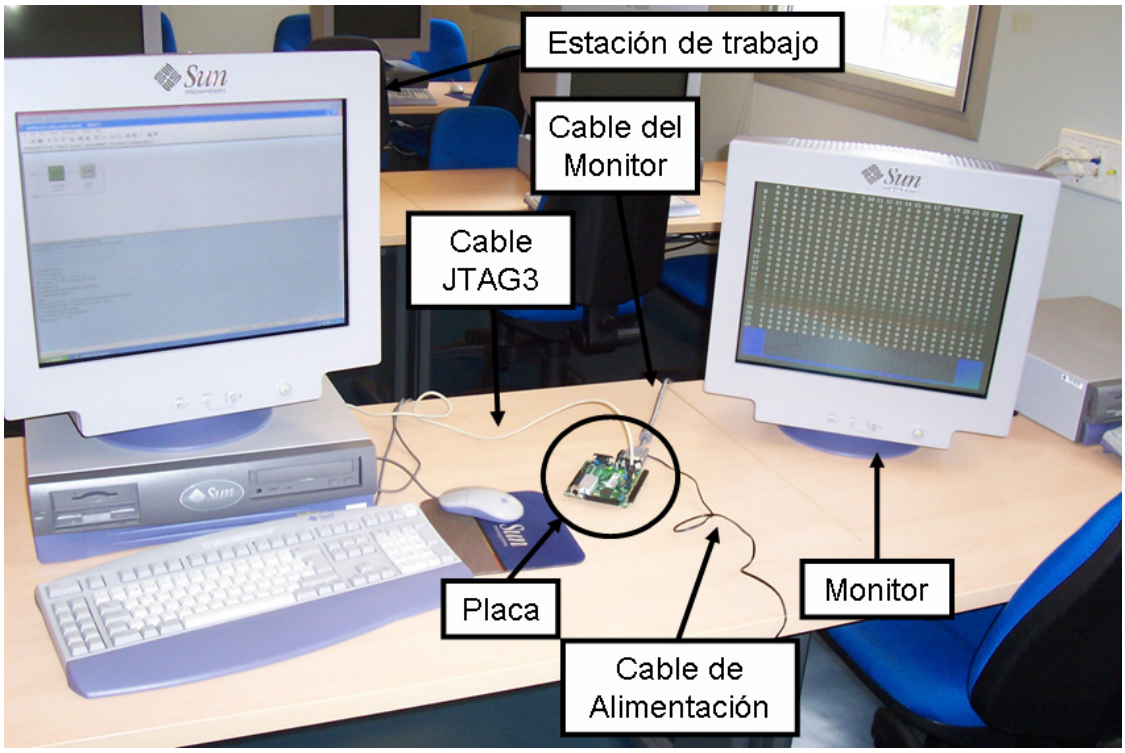


Figura 6.6: Elementos utilizados en la verificación hardware

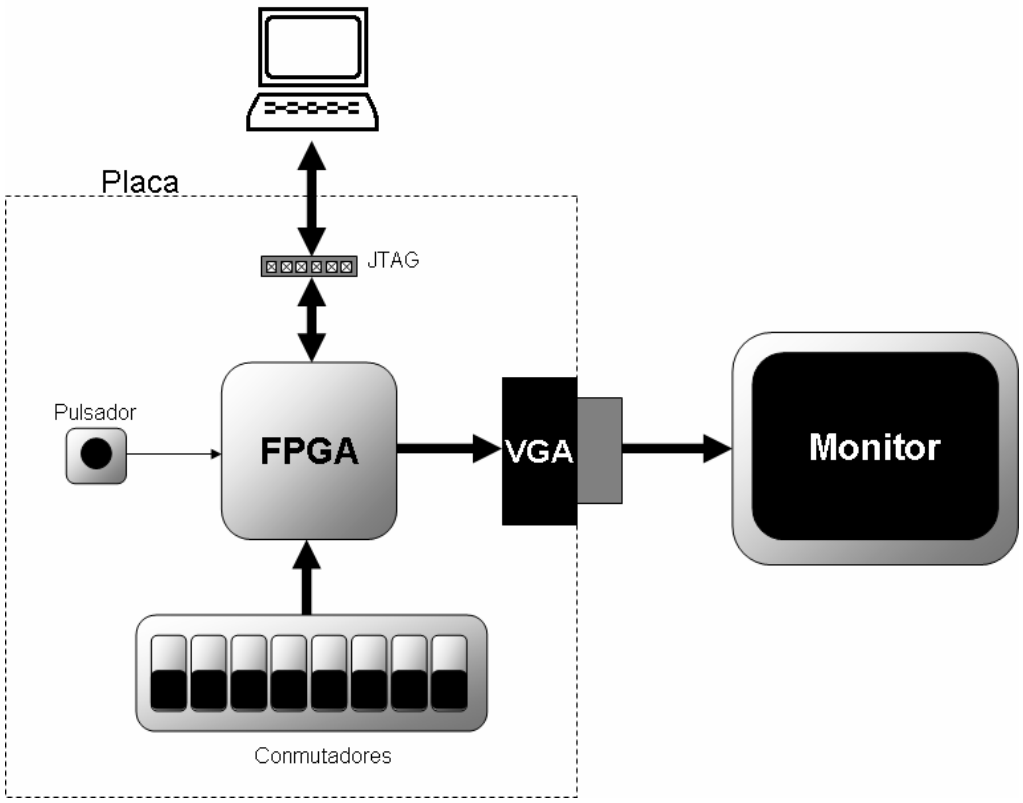


Figura 6.7: Esquema de los elementos utilizados en la verificación hardware

La estructura general del sistema que se va a implementar en la *FPGA* se puede ver en la figura 6.8. Está formada por un módulo de nivel superior *top* que contiene a su vez los módulos *test*, *interfaz_VGA* y *VGA_test*. Su funcionamiento es el siguiente:

- Los datos se van a presentar en el monitor con el formato que se indica en la figura 6.9. En él podemos distinguir una cabecera para las filas y otra para las columnas, que nos permiten identificar los datos; y los datos propiamente dicho, que serán una señal que queramos analizar. Cada posición de los datos será el valor de la señal en una trama determinada, contando de izquierda derecha y de arriba abajo a partir de 0. En este ejemplo, si queremos saber en qué trama se activa la señal, rodeada con un círculo, sólo tenemos que aplicar la siguiente fórmula:

  $$Nº\ Trama = Nº\ Fila \times 25 + Nº\ Columna$$

  En el ejemplo el número de fila es 10 y el de columna es 11, por lo tanto, aplicando la anterior fórmula, la señal se activará en la trama 261.

- Al accionar el pulsador se resetea el sistema.

- Al soltar el pulsador, se limpia la pantalla y se muestran las cabeceras.

- Cuando el receptor termina de recibir una trama, el valor de la señal a analizar se muestra en la pantalla.

- Cuando la pantalla esté completa se para el sistema, permaneciendo los datos en la pantalla hasta que se vuelve a resetear el mismo.

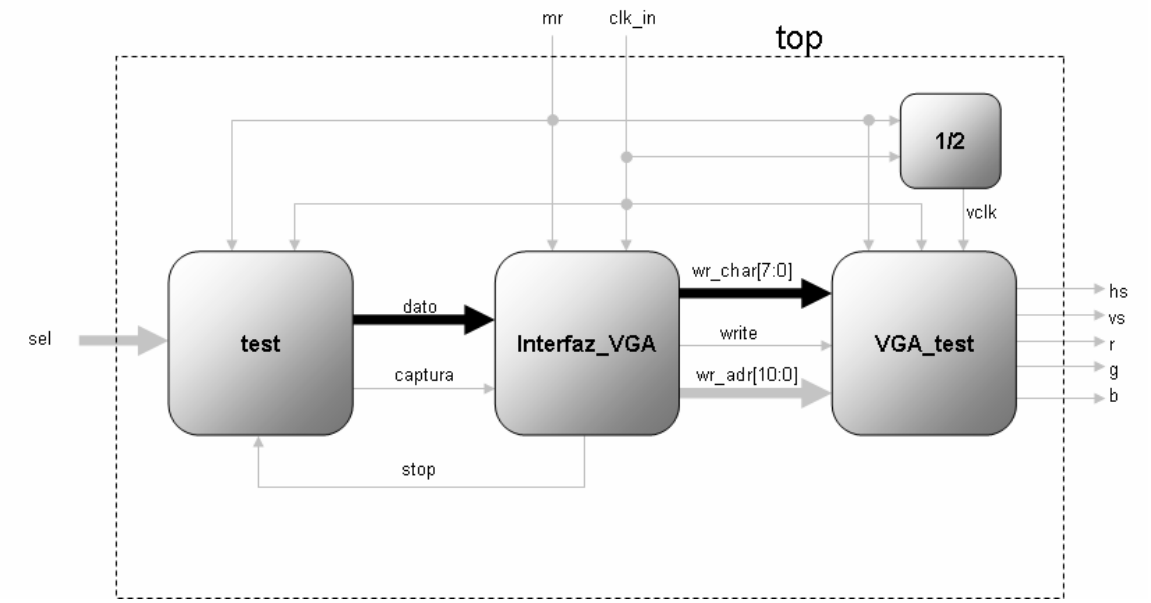


Figura 6.8: Estructura interna del módulo *top*

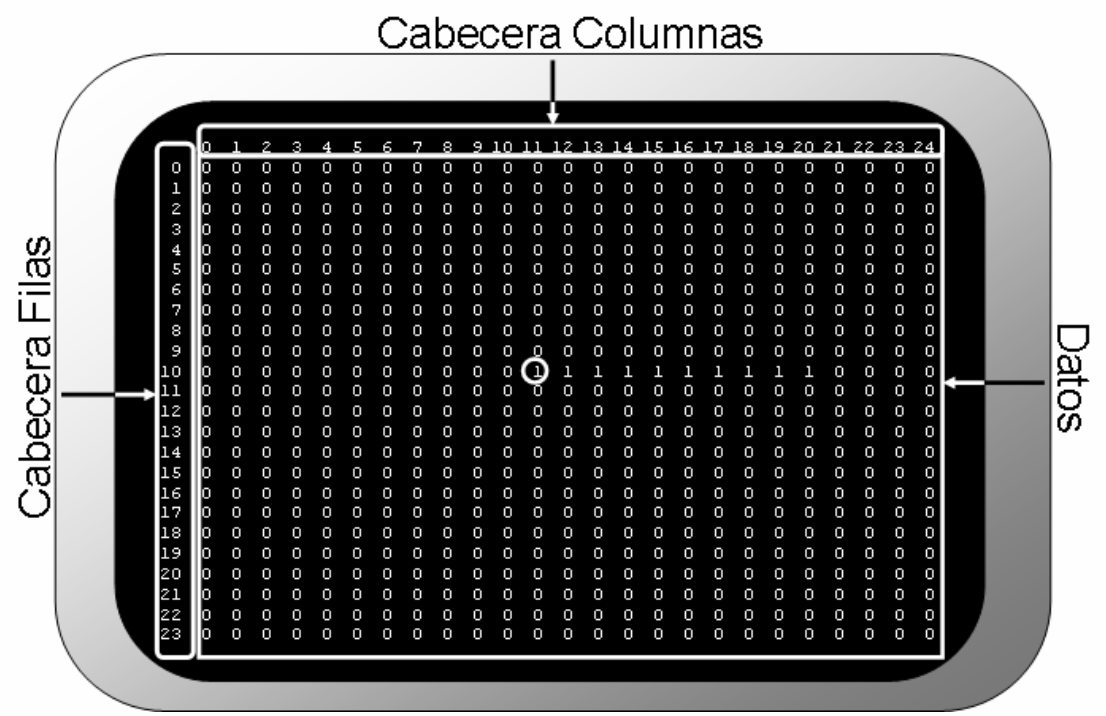


Figura 6.9: Formato de los datos representados en el monitor.

A continuación se explica de forma más detallada los módulos que componen la estructura general de cada test.

6.3.1.1 Módulo *top*

- Interfaz

Nombre	E/S	Descripción
clk_in	E	Reloj de entrada
mr	E	Reset, procedente de uno de los pulsadores
hs	S	Señal de sincronismo horizontal del puerto *VGA*
vs	S	Señal de sincronismo vertical del puerto *VGA*
r	S	Señal de rojo del puerto *VGA*
g	S	Señal de verde del puerto *VGA*
b	S	Señal de azul del puerto *VGA*
sel (opcional)	E	Selección, procedente de uno de los conmutadores

Tabla 6.4: Interfaz del módulo *top*

- Estructura interna

  Ya mostrada en la figura 6.8.

- Funciones

  Este módulo es el encargado de testear el módulo *receptor* y de presentar el resultado de dicho test en el monitor a través del puerto *VGA*.

- Funcionamiento

  A través de la entrada *clk_in* se proporciona la señal de reloj de 50 MHz del oscilador incorporado en la placa. Mediante un divisor de frecuencia se obtiene una señal de 25 MHz necesaria para el correcto funcionamiento del módulo *VGA_test*.

  La señal *mr* procede de uno de los pulsadores, el *BTN3*, y se utiliza para resetear el sistema.

La señal *sel* procede de uno o varios conmutadores. Se utiliza cuando se quiere conocer el valor de distintas señales para un mismo test.

El módulo *test* consta de un generador de trama y del receptor. Cuando éste último termina de recibir una trama, el módulo *test* presenta un valor en su salida *dato*, que será la señal a analizar, y una señal para capturar dicho dato, la señal *captura*.

El módulo *VGA_test* representa en el monitor el carácter suministrado en código *ASCII* a su entrada *wr_char[7:0]*, en la posición indicada en su entrada *wr_adr[10:0]*.

El módulo *interfaz_VGA*, genera las señales y los valores adecuados para que el módulo *VGA_test*, después de un reset, limpie la pantalla y muestre en la misma las cabeceras y los datos proporcionados por el módulo *test*.

En los siguientes apartados se describen con más detalle los módulos que componen el módulo *top*.

6.3.1.1.1 Módulo *test*

- Interfaz

Nombre	E/S	Descripción
clk_Rx	E	Reloj
mr	E	Reset
sel (opcional)	E	Selección
captura	S	Señal de captura
dato	S	Dato a mostrar

Tabla 6.5: Interfaz del módulo *test*

- Estructura interna

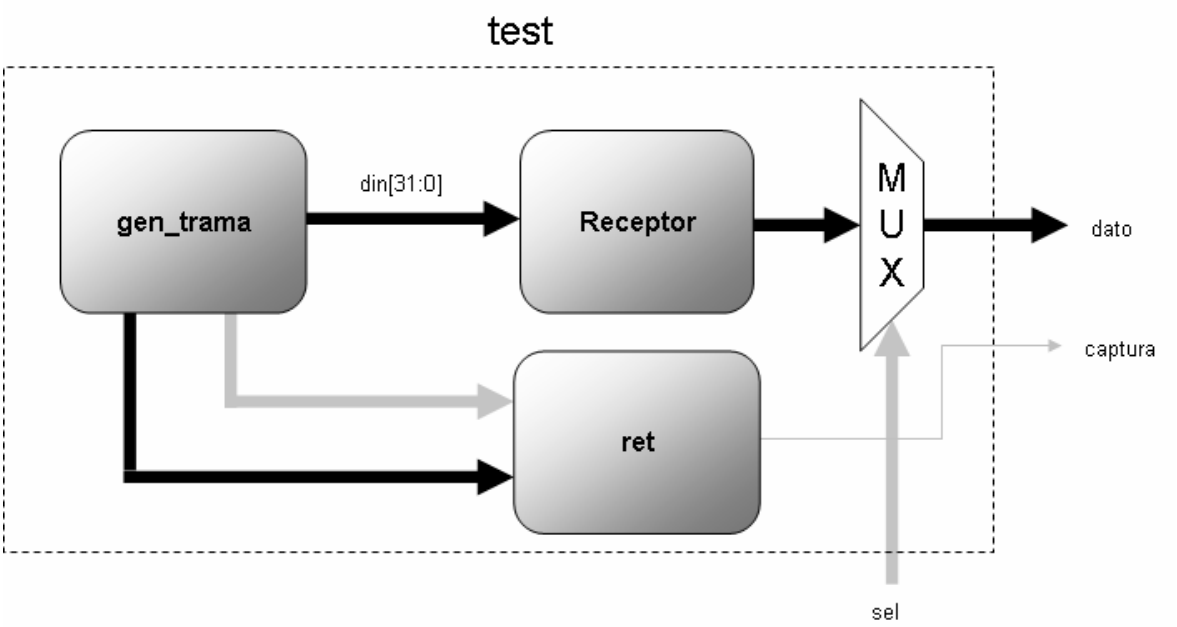


Figura 6.10: Estructura interna del módulo *test*

- Funciones

  Este módulo se encarga de generar tramas para el receptor y de proporcionar la señal a analizar, así como su señal de captura.

- Funcionamiento

  El módulo *gen_trama* genera las tramas para el receptor.

  El módulo *ret* retarda algunas señales, dependiendo de los tests a realizar, unos ciclos de reloj, para que coincidan con los datos a la salida del receptor, ya que éste retardará también los datos. Una señal común para todos los tests es la señal *SOF* activada por el generador, que indica el comienzo de trama en el generador. Debido a que el dato ha de ser presentado después de cada trama recibida, se puede aprovechar esta señal, con su retardo correspondiente, para generar la señal *captura*.

En el siguiente apartado se describe el módulo *gen_trama*

6.3.1.1.1.1 Módulo *gen_trama*

- Interfaz

Varía en función del test realizado, pero incluye al menos las siguientes señales:

Nombre	E/S	Descripción
clk_Rx	E	Reloj del receptor
mr	E	Reset
dout[31:0]	E	Datos de salida
SOF	E	Comienzo de la trama

Tabla 6.6: Interfaz del módulo *gen_trama*

- Estructura interna

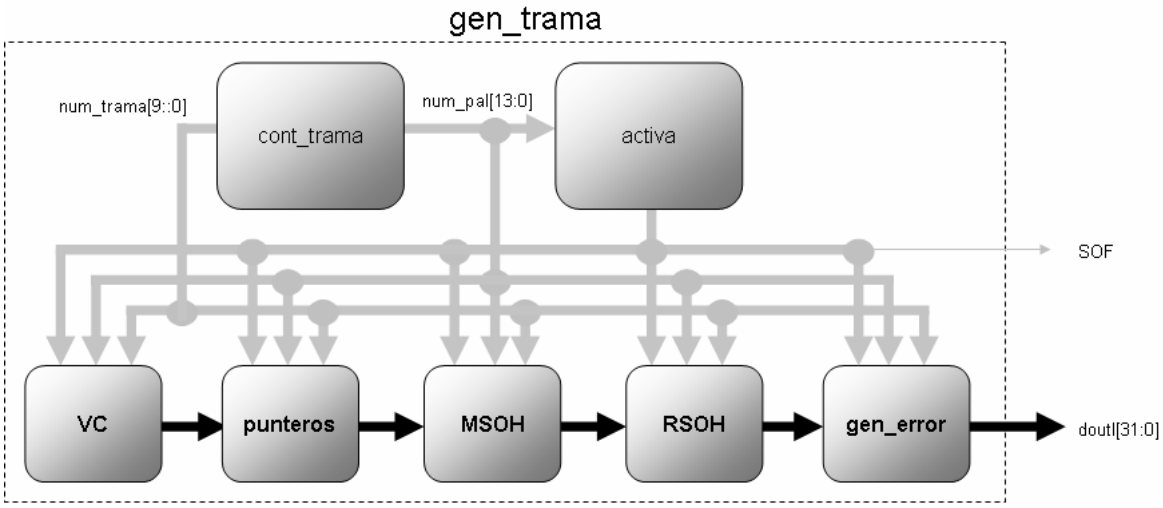


Figura 6.11: Estructura interna del módulo *gen_trama*

- Funciones

  El módulo *gen_trama* genera las tramas para el receptor, con un valor de puntero constante 3.

- Funcionamiento

  El módulo *cont_trama* lleva la cuenta de las tramas y de las palabras de cada trama que se van generando.

  El módulo *activa* genera todas las señales de sincronización de los demás módulos, en función del número de palabras que se han generado de cada trama.

  El módulo *VC*, genera una trama con el *VC* ya insertado y con las demás taras vacías. Para los datos del *VC* emplea un generador de datos pseudoaleatorios de 32 bits. Esta trama irá pasando por los módulos *punteros*, *MSOH* y *RSOH*, que irán añadiendo las restantes taras. Los octetos podrán ser variados en cada trama gracias al contador de trama.

  Por último con el módulo *gen_error* se podrá insertar errores en determinadas palabras para distintas tramas.

6.3.1.1.2 Módulo *VGA_test*

- Interfaz

Nombre	E/S	Descripción
vclk	E	Reloj de 25 MHz
mr	E	Reset
hs	S	Señal de sincronismo horizontal del puerto *VGA*
vs	S	Señal de sincronismo vertical del puerto *VGA*
r	S	Señal de rojo del puerto *VGA*
g	S	Señal de verde del puerto *VGA*
b	S	Señal de azul del puerto *VGA*
bg[2:0]	E	Señal de captura del octeto *D2*
fg[2:0]	E	Señal de captura del octeto *D3*
wr_clk	E	Reloj de escritura
wr_adr[10:0]	E	Dirección de escritura
wr_char[7:0]	E	Carácter a escribir
write	E	Señal de escritura

Tabla 6.7: Interfaz del módulo *VGA_test*

- Funciones

El módulo *VGA_test* ha sido proporcionado por uno de los tutores del proyecto. Se encarga de escribir caracteres en código *ASCII* en pantalla.

- Funcionamiento

Este módulo posee una memoria interna de 2000 posiciones (0-1999), correspondiéndole una posición en pantalla a cada una.

A partir del valor de cada posición estará continuamente generando las señales *hs*, *vs*, *r*, *g* y *b* para representar los valores almacenados en la memoria en pantalla. Para ello necesita una señal de reloj de 25 MHz en su entrada *vclk*.

Para que represente un dato en una determinada posición en pantalla, hay que hacer lo siguiente: en la entrada *wr_char[7:0]* se presenta el dato codificado en *ASCII* y

en *wr_adr[10:0]* la posición de memoria donde se desea escribir. El módulo *VGA_test* almacenará dicho dato en el flanco de subida de la señal de reloj *wr_clk*, cuando la señal *write* este activa (a nivel alto). Para la señal *wr_clk* se ha utilizado la señal de reloj procedente del oscilador incorporado en la placa.

Por último, también se puede controlar el color de la fuente y del fondo para cada posición de pantalla, aunque en todos los tests se han utilizado el blanco para la fuente y el negro para el fondo.

6.3.1.1.3 Módulo *interfaz_VGA*

- Interfaz

Nombre	E/S	Descripción
clk	E	Reloj
mr	E	Reset
captura	E	Señal de captura
dato	E	Dato a mostrar
wr_adr[10:0]	S	Dirección de escritura
wr_char[7:0]	S	Carácter a escribir
write	S	Señal de escritura
stop	S	Parar el módulo test

Tabla 6.8: Interfaz del módulo *interfaz_VGA*

- Estructura interna

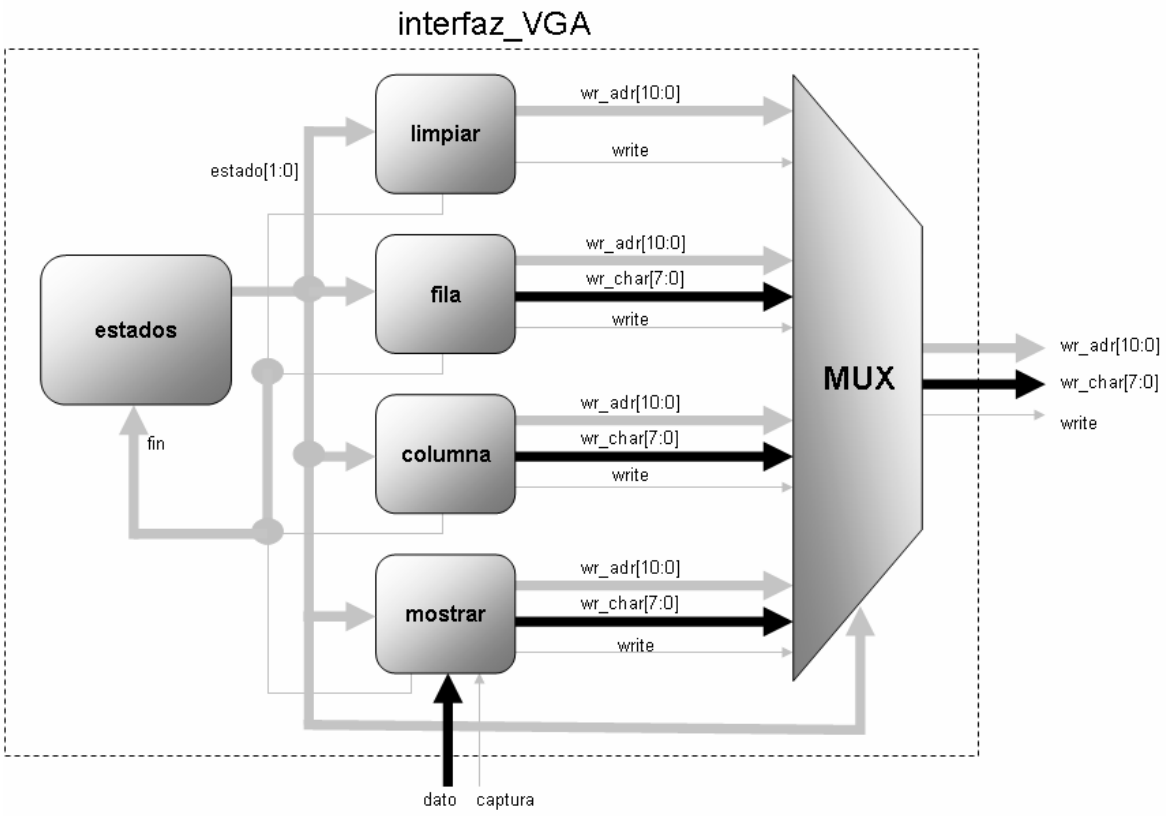


Figura 6.12: Estructura interna del módulo *interfaz_VGA*

- Funciones

  Este módulo genera las señales y los valores adecuados para que el módulo *VGA_test*, después de un reset, limpie la pantalla y muestre en la misma las cabeceras y los datos proporcionados por el módulo *test*.

- Funcionamiento

  El módulo trabaja en cuatro estados diferentes:

1. *Limpiar pantalla*: limpia todas las posiciones de la pantalla.

2. *Representar fila*: muestra la fila correspondiente a la cabecera para las columnas.

3. *Representar columna*: muestra la columna correspondiente a la cabecera para las filas.

4. *Mostrar los datos*: muestra los datos correspondientes a la señal a analizar en cada trama.

El módulo *estados* es el que se encarga de transitar entre los distintos estados. Lo hace de forma secuencial:

1. Se parte del estado limpiar pantalla después de un reset.

2. En este estado, trabaja el módulo *limpiar*, recorriendo todas las posiciones de la pantalla con el valor en código *ASCII* correspondiente al espacio en blanco.

3. Al terminar de limpiar la pantalla, envía la señal *fin_limpiar* al módulo *estados* para que transite al estado *Representar fila*.

4. Ahora es el módulo *fila* el que trabaja, representando la cabecera para las columnas en pantalla. Al finalizar, envía la señal *fin_fila* y se transita al estado *Representar columna*.

5. En este estado, el módulo *columna* representa la cabecera para las filas en pantalla, enviando la señal *fin_columna* al finalizar, pasándose al estado *Mostrar datos*.

6. En el último estado, el módulo *mostrar* muestra cada dato que le va llegando cada vez que se activa la señal *captura* en una determinada posición de memoria.

7. Una vez se llena la pantalla con los datos, se activa la señal *fin_mostrar*, permaneciéndose en este estado hasta que se resetee otra vez el sistema. Además se activa la señal *stop* para que el módulo *test* se detenga.

El módulo *MUX* es un multiplexor que selecciona entre las señales *wr_char[7:0]*, *wr_adr[10:0]* y *write* procedentes de los módulos *limpiar*, *fila*, *columna* y *mostrar*, según el estado en que se encuentre el sistema.

6.3.2 Resumen de los Tests Realizados

En la tabla 6.9 vienen resumidos los tests realizados a nivel de sistema. Consta de dos columnas:

- Tests: identificador del test.
- Objetivo: describe qué es lo que se pretende verificar.

Tests	Objetivo
01	Comprobar la activación/desactivación de las señal *act_datos*
02	Comprobar la activación/desactivación de las señal *sinc*
03	Comprobar la activación/desactivación de las señales *MS-AIS* y *MS-RDI*
04	Comprobar la activación/desactivación de la señal *AU-AIS*
05	Comprobar la activación/desactivación de la señal *AU-LOP*
06	Comprobar la activación/desactivación de las señales *VC-AIS* y *dUNEQ*
07	Comprobar la activación/desactivación de la señal *HO-PATH-RDI*
08	Comprobar la activación/desactivación de las señales *error_B1*, *error_B2* y *error_B3*

Tabla 6.9: Tests a nivel de sistema

6.3.3 Tests Realizados

6.3.3.1 Test 01

Objetivos:

- Comprobar la activación/desactivación de la señal *act_datos*.

Procedimiento:

- El módulo *test* presenta la estructura mostrada en la figura 6.13, donde se puede observar que incluye un módulo *comparador*, además de los ya mencionados anteriormente.

- El módulo *gen_trama* generará tramas sin aleatorizar que van a pasar por el módulo *ret_datos*. Estas tramas serán también aleatorizadas, añadiéndoles errores a algunas, y pasadas al *receptor*.

- El módulo *ret_datos* retardará las tramas sin aleatorizar y sin errores y las señales *act_datos* y *SOF* procedentes del *gen_trama*.

- El *comparador* realizará dos comprobaciones simultáneas:

  - Verificará si la señal *act_datos* del *receptor* y del *gen_trama* permanecen activadas y desactivadas el mismo tiempo.

  - Verificará que los datos suministrados por el *receptor* coinciden con los generados por el *gen_trama*.

- Estas comprobaciones serán realizadas cuando el *receptor* se haya sincronizado y haya encontrado el desplazamiento del *VC*. Si en una trama hay algún error, es decir, los datos no coinciden o las señales *act_datos* del *receptor* y del *gen_trama*

no permanecen activadas y desactivadas el mismo tiempo, se activa la señal *d_error*, que es la que va a ser utilizada para representarse en pantalla como *dato*.

- En la tabla 6.10 se indica qué tramas se van a generar sin errores y cuales no.

Resultados esperados:

- Hay que comprobar el valor de la señal *d_error*.

- La tabla 6.10 muestra los resultados esperados.

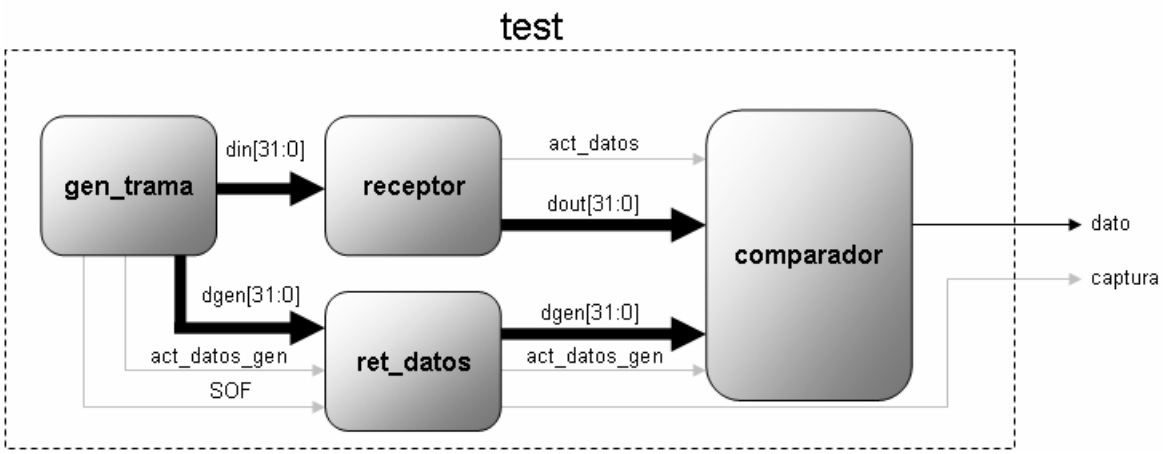


Figura 6.13: Estructura interna del módulo *test*

ESTÍMULOS		RESULTADOS ESPERADOS
Tramas	**Correctas/Erróneas**	**d_error**
0-149	C	0
150	E	1
151-399	C	0
400	E	1
401-549	C	0
550	E	1
Resto	C	0

Tabla 6.10: Estímulos y resultados esperados

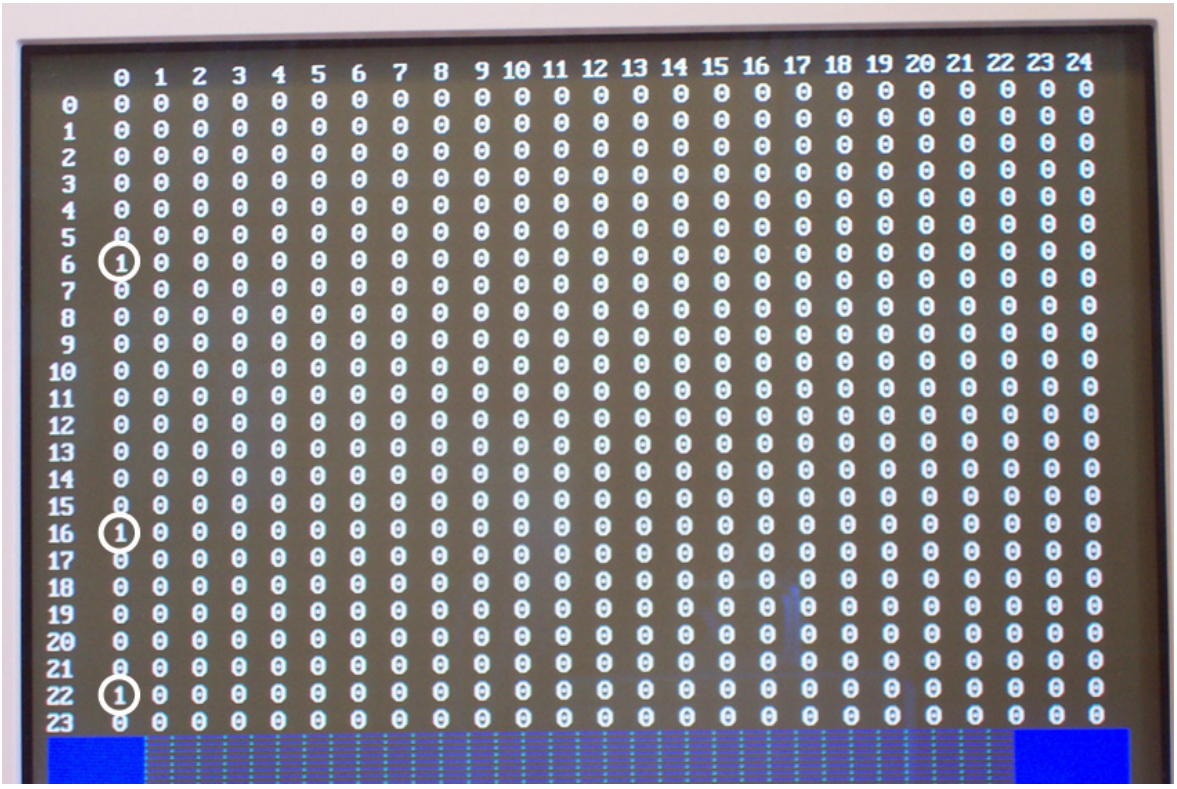


Figura 6.14: Resultados obtenidos para la señal *d_error* en el test 01

Resultados obtenidos:

- Se muestran en la figura 6.14.

Análisis de los resultados:

- Observando los resultados obtenidos, podemos ver como en las tramas 150, 400 y 550 se ha activado la señal *d_error*, permaneciendo inactiva en el resto.

Conclusión: Comparando los resultados obtenidos con los esperados, se puede decir que el sistema cumple con las especificaciones.

6.3.3.2 Test 02

Objetivos:

- Comprobar la activación/desactivación de la señal *sinc*.

Procedimiento:

- Se van a generar tramas con los octetos *A1* y/o *A2* incorrectos en alguna de ellas.

- En la tabla 6.11 se indica qué tramas se van a generar sin errores y cuales no.

Resultados esperados:

- Hay que comprobar el valor de la señal *sinc*.

- La tabla 6.11 muestra los resultados esperados.

ESTÍMULOS		RESULTADOS ESPERADOS
Tramas	**Correctas/Erróneas**	**sinc**
0-1	C	0
2-29	C	1
30-34	E	1
35-50	E	0
51-52	C	0
53-79	C	1
80-84	E	1
85-100	E	0
101-102	C	0
Resto	C	1

Tabla 6.11: Estímulos y resultados esperados

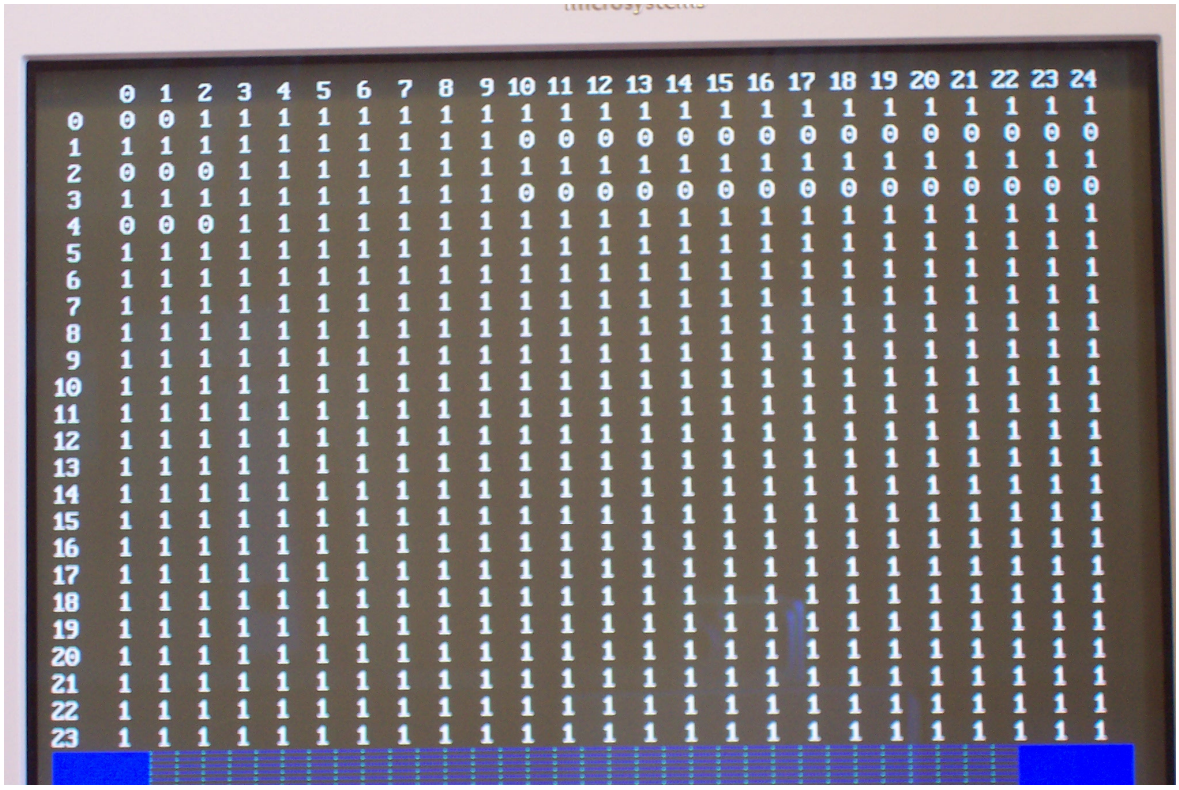


Figura 6.15: Resultados obtenidos para la señal *sinc* en el test 02

Resultados obtenidos:

- Se muestran en la figura 6.15.

Análisis de los resultados:

- Observando los resultados obtenidos se puede verificar que cuando se reciben tres tramas consecutivas con todos los octetos *A1* y *A2* correctos, estando las señal *sinc* desactivada, ésta se activa (tramas 2, 53 y 103).

- Al recibir seis tramas consecutivas con alguno de los octetos *A1* y *A2* incorrectos, estando las señal *sinc* activada, ésta se desactiva (tramas 35 y 85).

Conclusión: Comparando los resultados obtenidos con los esperados, se puede decir que el sistema cumple con las especificaciones.

6.3.3.3 Test 03

Objetivos:

- Comprobar la activación/desactivación de las señales *MS-AIS* y *MS-RDI*.

Procedimiento:

- Se van a generar tramas en las que se variará el valor del octeto *K2*.
- Con el conmutador *SW0* de la placa (pin *F12* de la *FPGA*) se seleccionará la señal a visualizar en el monitor: *0b* para *MS-AIS* y *1b* para *MS-RDI*.
- La tabla 6.12 muestra el valor del octeto *K2* para cada trama.

Resultados esperados:

- Hay que comprobar el valor de las señales *MS-AIS* primero y de *MS-RDI* después.
- La tabla 6.12 muestra los resultados esperados.

ESTÍMULOS		RESULTADOS ESPERADOS	
Tramas	**K2[2:0]**	**MS-AIS**	**MS-RDI**
0-29	000b	0	0
30-31	111b	0	0
32-40	111b	1	0
41-42	000b	1	0
43-69	000b	0	0
70-71	110b	0	0
72-80	110b	0	1
81-82	000b	0	1
Resto	000b	0	0

Tabla 6.12: Estímulos y resultados esperados

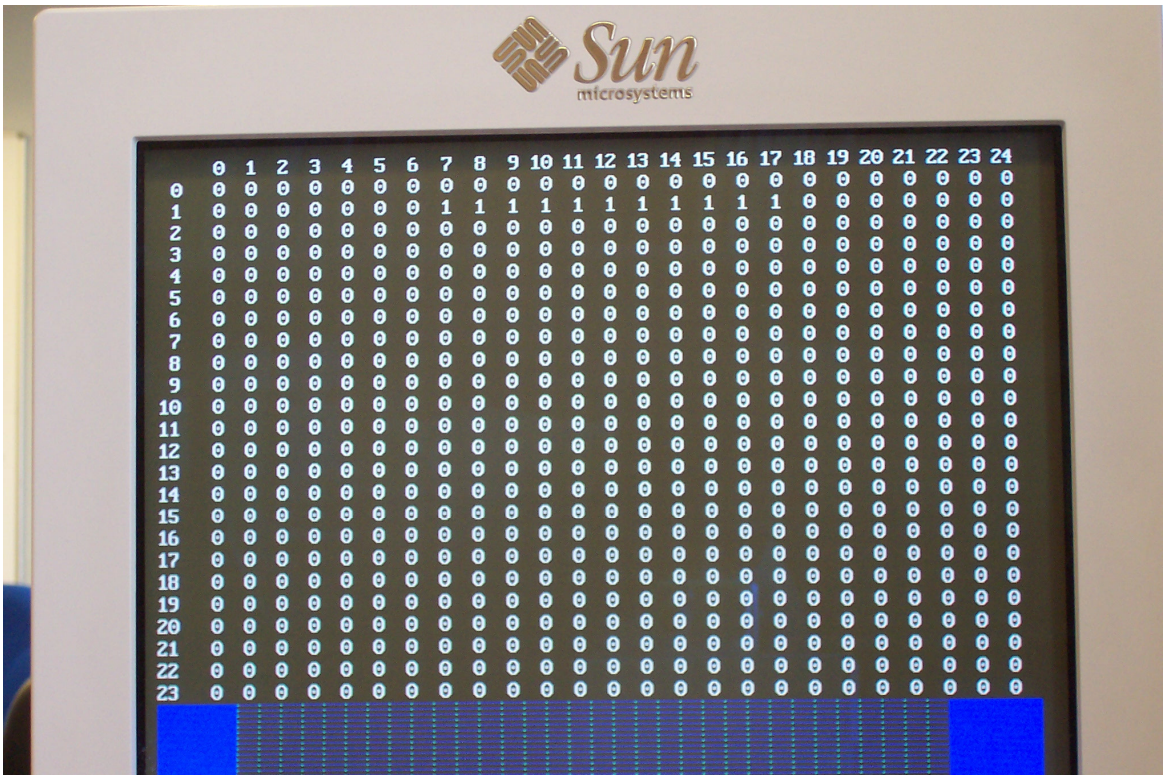


Figura 6.16: Resultados obtenidos para la señal *MS-AIS* en el test 03

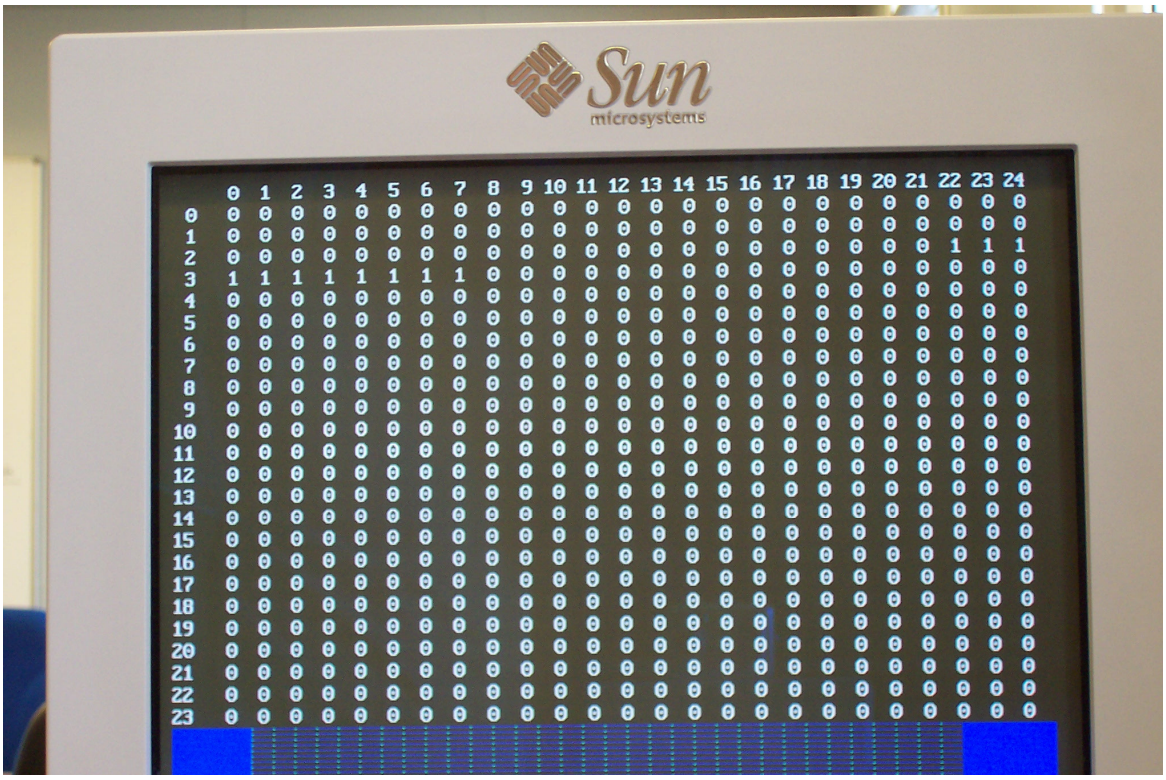


Figura 6.17: Resultados obtenidos para la señal *MS-RDI* en el test 03

Resultados obtenidos:

- Se muestran en las figura 6.16 y 6.17.

Análisis de los resultados:

- Observando los resultados obtenidos se puede verificar que cuando se reciben tres tramas consecutivas con los tres bits menos significativos del octeto *K2* con un valor de *111b*, estando la señal *MS-AIS* desactivada, ésta se activa (trama 32).

- Al recibir tres tramas consecutivas con los tres bits menos significativos del octeto *K2* con un valor de *110b*, estando la señal *MS-RDI* desactivada, ésta se activa (trama 72).

- Al recibir tres tramas consecutivas con los tres bits menos significativos del octeto *K2* con un valor distinto de *110b* y *111b*, se desactivan tanto la señal *MS-AIS* como la *MS-RDI* (tramas 43 y 83).

Conclusión: Comparando los resultados obtenidos con los esperados, se puede decir que el sistema cumple con las especificaciones.

6.3.3.4 Test 04

Objetivos:

- Comprobar la activación/desactivación de la señal *AU-AIS*.

Procedimiento:

- Se van a generar tramas en las que se variará el valor de los punteros, incluidos los concatenados, para enviar indicaciones *AIS_ind*.

- La tabla 6.13 muestra la indicación enviada en cada trama.

Resultados esperados:

- Hay que comprobar el valor de la señal *AU-AIS*.

- La tabla 6.13 muestra los resultados esperados.

ESTÍMULOS			RESULTADOS ESPERADOS
Tramas	**Puntero del VC**	**Punteros concatenados**	**AU-AIS**
0-19	*norm_point*	*conc_point*	0
20-30	*norm_point*	*AIS_ind*	0
30-31	*AIS_ind*	*AIS_ind*	0
32-40	*AIS_ind*	*AIS_ind*	1
41-42	*AIS_ind*	*conc_point*	1
43-50	*AIS_ind*	*conc_point*	0
51-52	*norm_point*	*conc_point*	0
Resto	*norm_point*	*conc_point*	0

Tabla 6.13: Estímulos y resultados esperados

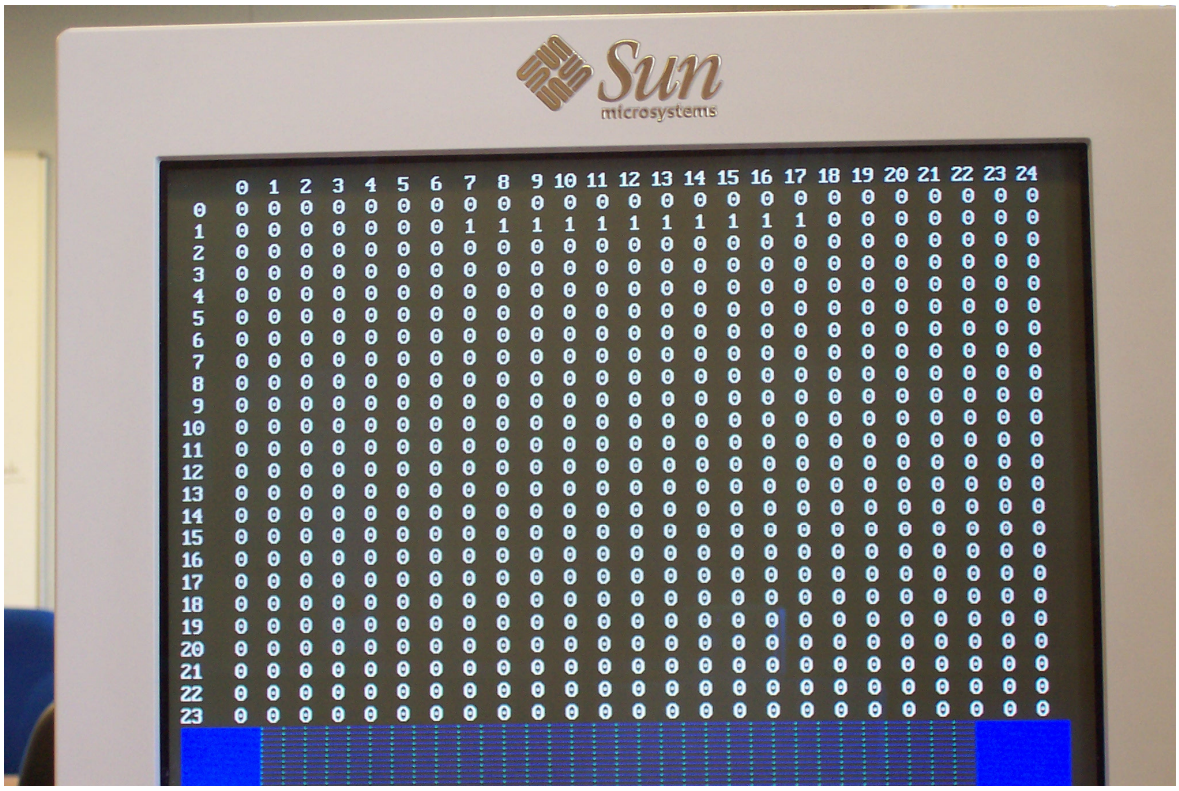


Figura 6.18: Resultados obtenidos para la señal *AU-AIS* en el test 04

Resultados obtenidos:

- Se muestran en la figura 6.18.

Análisis de los resultados:

- Observando los resultados obtenidos se puede verificar que cuando se reciben durante tres tramas consecutivas indicaciones *AIS_ind*, tanto para los punteros del *VC* como para los concatenados, se activa la señal *AU-AIS* (trama 32).

- Al recibir tres tramas consecutivas sin ninguna indicación *AIS_ind* en los punteros del *VC* y/o en los concatenados, se desactiva la señal *AU-AIS* (trama 43).

Conclusión: Comparando los resultados obtenidos con los esperados, se puede decir que el sistema cumple con las especificaciones.

6.3.3.5 Test 05

Objetivos:

- Comprobar la activación/desactivación de la señal *AU-LOP*.

Procedimiento:

- Se van a generar tramas en las que se variará el valor de los punteros, incluidos los concatenados, para enviar distintas indicaciones.

- La tabla 6.14 muestra la indicación enviada en cada trama.

Resultados esperados:

- Hay que comprobar el valor de la señal *AU-LOP*.

- La tabla 6.14 muestra los resultados esperados.

Resultados obtenidos:

- Se muestran en la figura 6.19.

Análisis de los resultados:

- Observando los resultados obtenidos se puede verificar que cuando se reciben durante tres tramas consecutivas indicaciones *AIS_ind* para los punteros del *VC* o tres para los punteros concatenados, se activa la señal *AU-LOP* (tramas 50 y 130).

- Al recibir ocho tramas consecutivas con indicaciones *inv_point* en los punteros del *VC* o tres en los concatenados, también se activa la señal *AU-AIS* (trama 43).

- Al recibir tres tramas consecutivas indicaciones *norm_point* y *conc_point*, se desactiva la señal *AU-AIS* (tramas 4, 43, 63, 123 y 143).

Conclusión: Comparando los resultados obtenidos con los esperados, se puede decir que el sistema cumple con las especificaciones.

ESTÍMULOS			RESULTADOS ESPERADOS
Tramas	**Puntero del VC**	**Punteros concatenados**	**AU-LOP**
0-3	*norm_point*	*conc_point*	1
4-19	*norm_point*	*conc_point*	0
20-26	*inv_point*	*conc_point*	0
27-40	*inv_point*	*conc_point*	1
41-42	*norm_point*	*conc_point*	1
43-49	*norm_point*	*conc_point*	0
50-56	*AIS_ind*	*conc_point*	0
57-60	*AIS_ind*	*conc_point*	1
61-62	*norm_point*	*conc_point*	1
63-99	*norm_point*	*conc_point*	0
100-106	*norm_point*	*inv_point*	0
107-120	*norm_point*	*inv_point*	1
121-122	*norm_point*	*conc_point*	1
123-129	*norm_point*	*conc_point*	0
130-131	*norm_point*	*AIS_ind*	0
132-140	*norm_point*	*AIS_ind*	1
141-142	*norm_point*	*conc_point*	1
Resto	*norm_point*	*conc_point*	0

Tabla 6.14: Estímulos y resultados esperados

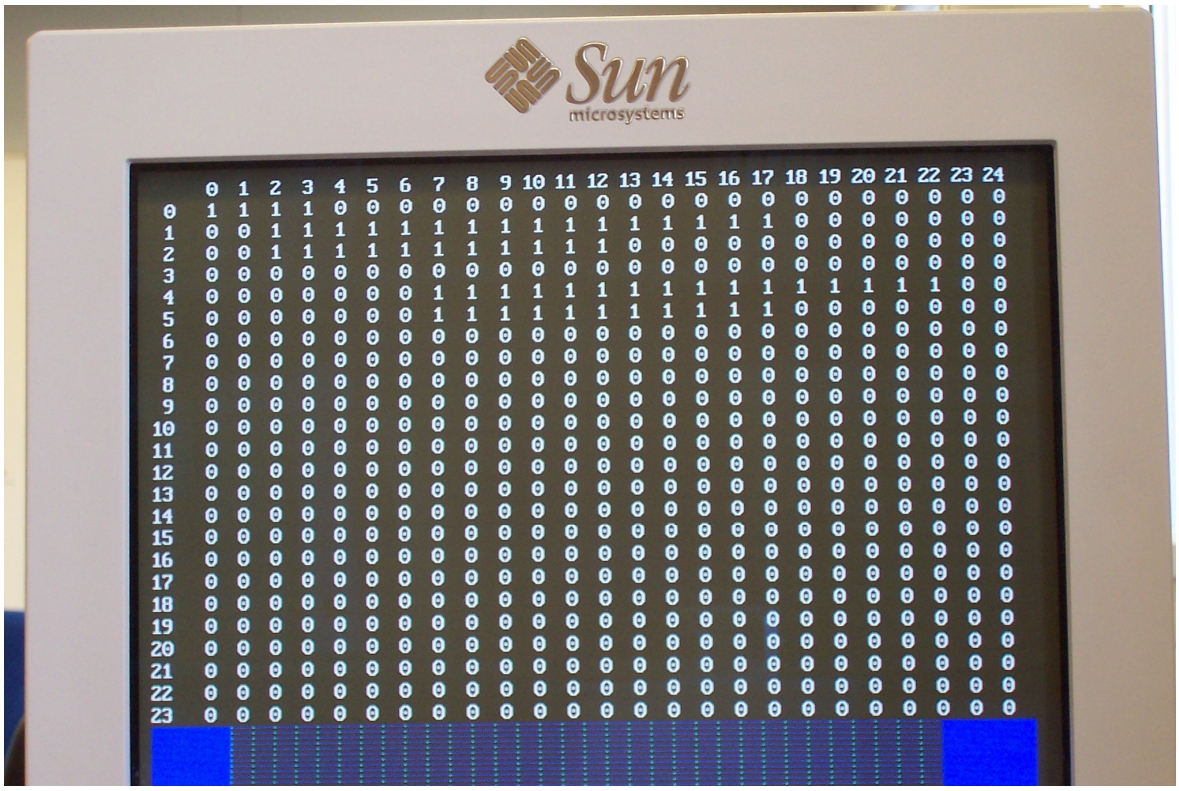


Figura 6.19: Resultados obtenidos para la señal *AU-LOP* en el test 05

6.3.3.6 Test 06

Objetivos:

- Comprobar la activación/desactivación de las señales *VC-AIS* y *dUNEQ*.

Procedimiento:

- Se van a generar tramas en las que se variará el valor del octeto *C2*.
- Con el conmutador *SW0* de la placa (pin *F12* de la *FPGA*) se seleccionará la señal a visualizar en el monitor: *0b* para *VC-AIS* y *1b* para *dUNEQ*.
- La tabla 6.15 muestra el valor del octeto *C2* para cada trama.

Resultados esperados:

- Hay que comprobar el valor de la señal *VC-AIS* primero y *dUNEQ* después.

- La tabla 6.15 muestra los resultados esperados.

ESTÍMULOS		RESULTADOS ESPERADOS	
Tramas	**C2[7:0]**	**VC-AIS**	**dUNEQ**
0-29	17h	0	0
30-33	FFh	0	0
34-50	FFh	1	0
51-54	17h	1	0
55-79	17h	0	0
80-83	0h	0	0
84-100	0h	0	1
101-104	17h	0	1
Resto	17h	0	0

Tabla 6.15: Estímulos y resultados esperados

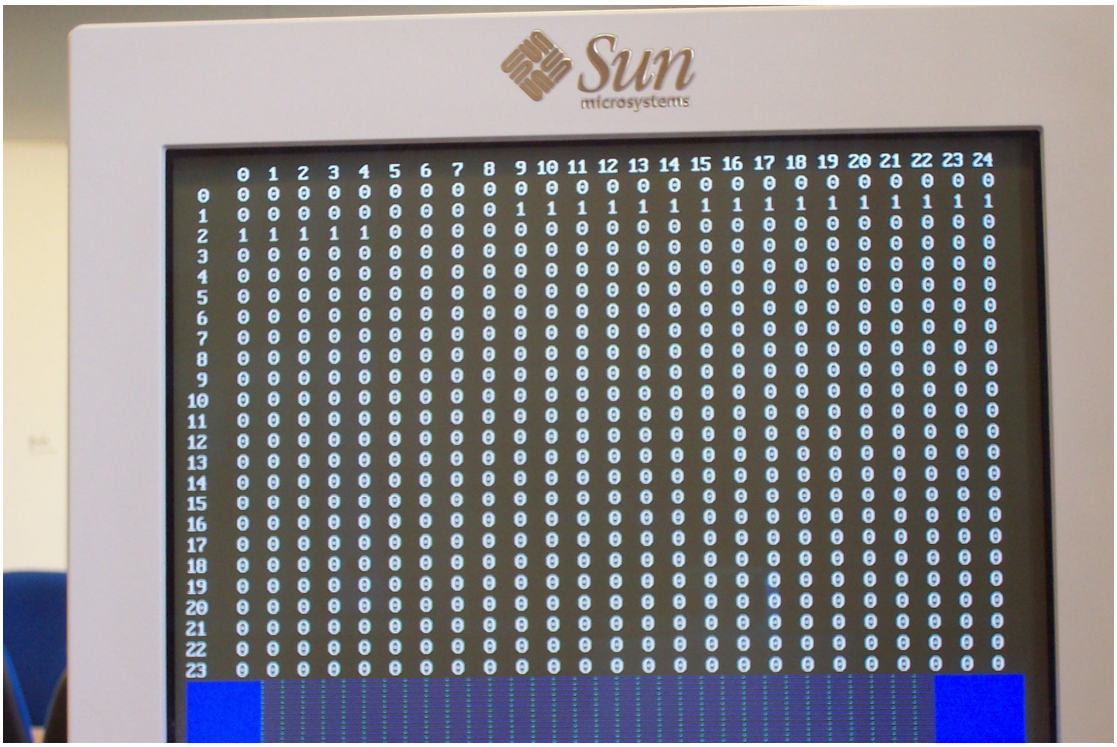


Figura 6.20: Resultados obtenidos para la señal *VC-AIS* en el test 06

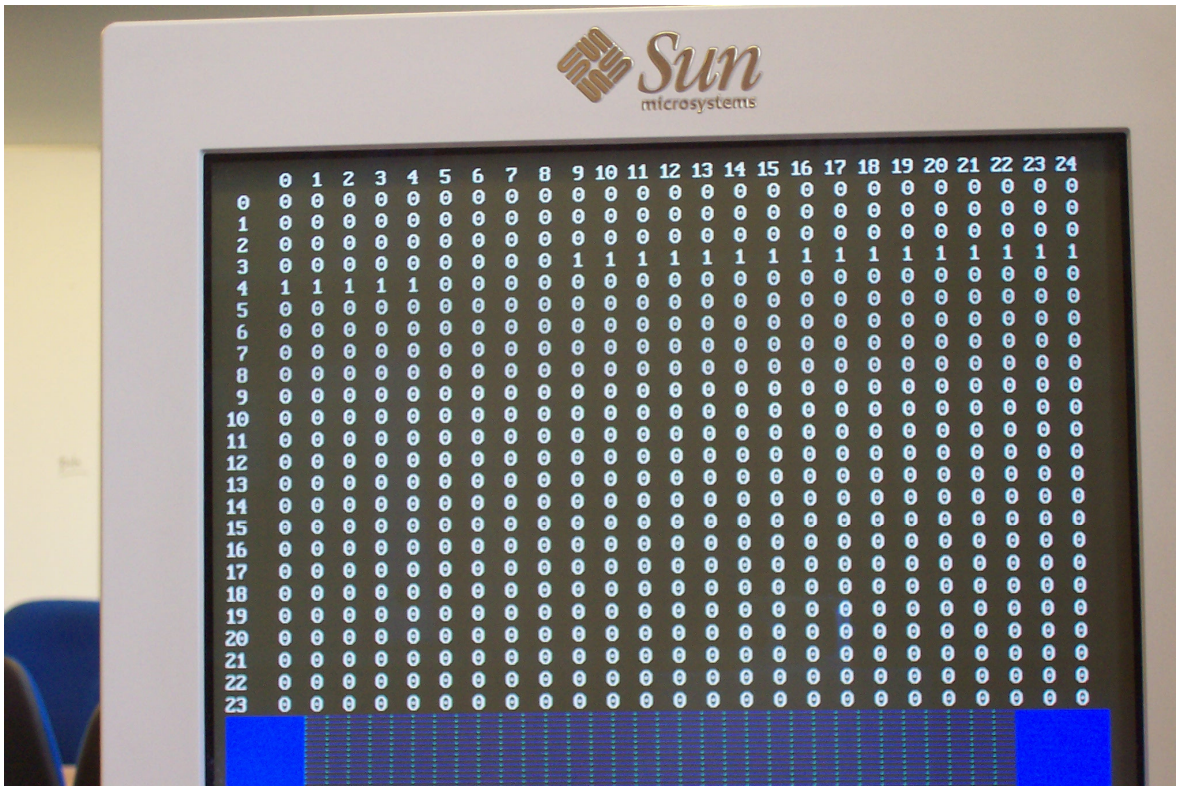


Figura 6.21: Resultados obtenidos para la señal *dUNEQ* en el test 06

Resultados obtenidos:

- Se muestran en las figuras 6.20 y 6.21.

Análisis de los resultados:

- Observando los resultados obtenidos se puede verificar que cuando se reciben cinco tramas consecutivas con el octeto *C2* con un valor de *FFh* se activa la señal *VC-AIS* (trama 34).

- Al recibir cinco tramas consecutivas con el octeto *C2* con un valor de *0h*, se activa la señal *dUNEQ* (trama 84).

- Al recibir cinco tramas consecutivas con el octeto *C2* con un valor distinto de *FFh* y *0h*, se desactivan tanto la señal *VC-AIS* como la *dUNEQ* (tramas 55 y 105).

Conclusión: Comparando los resultados obtenidos con los esperados, se puede decir que el sistema cumple con las especificaciones.

6.3.3.7 Test 07

Objetivos:

- Comprobar la activación/desactivación de la señal *HO-PATH-RDI*.

Procedimiento:

- Se van a generar tramas en las que se variará el valor del bit 3 del octeto *G1*.

- La tabla 6.16 muestra el valor del octeto *G1* para cada trama.

Resultados esperados:

- Hay que comprobar el valor de la señal *HO-PATH-RDI*.

- La tabla 6.16 muestra los resultados esperados.

ESTÍMULOS		RESULTADOS ESPERADOS
Tramas	G1[3]	HO-PATH-RDI
0-19	0	0
20-21	1	0
22-30	1	1
31-32	0	1
33-99	0	0
100-101	1	0
102-110	1	1
111-112	0	1
Resto	0	0

Tabla 6.16: Estímulos y resultados esperados

Resultados obtenidos:

- Se muestran en la figura 6.22.

Figura 6.22: Resultados obtenidos para la señal *HO-PATH-RDI* en el test 07

Análisis de los resultados:

- Observando los resultados obtenidos se puede verificar que cuando se reciben tres tramas consecutivas con el bit 3 del octeto *G1* con un valor de *1b* se activa la señal *HO-PATH-RDI* (tramas 22 y 102).

- Al recibir tres tramas consecutivas con el bit 3 del octeto *G1* con un valor de *0b*, se desactiva la señal *HO-PATH-RDI* (tramas 33 y 113).

Conclusión: Comparando los resultados obtenidos con los esperados, se puede decir que el sistema cumple con las especificaciones.

6.3.3.8 Test 08

Objetivos:

- Comprobar la activación/desactivación de las señales *error_B1*, *error_B2* y *error_B3*.

Procedimiento:

- Se van a generar tramas a las que se les calcularán el *B1*, *B2* y *B3* y luego se les introducirán errores a algunas.

- Con los conmutadores *SW1* y *SW0* de la placa (pines *G12* y *F12* de la *FPGA*) se seleccionará la señal a visualizar en el monitor.

- La tabla 6.17 muestra en qué tramas y en qué parte de las mismas se han introducido los errores.

Resultados esperados:

- Hay que comprobar el valor de las señales *error_B1*, *error_B2* y *error_B3*.

- La tabla 6.17 muestra los resultados esperados. Consta de las siguientes columnas:

  - *Tramas*: indica las tramas donde se va a introducir un error.

  - *Parte*: indica la parte de la trama donde se va a introducir el error.

  - *error_B1*, *error_B2* y *error_B3*: indican en que trama se deben activar dichas señales.

ESTÍMULOS		**RESULTADOS ESPERADOS**		
Tramas	**Parte**	**error_B1**	**error_B2**	**error_B3**
10	*VC*	11	11	11
52	*MSOH*	53	52* y 53	
101	*VC*	102	102	102
125	*RSOH*	126		
302	*RSOH*	303		
315	*MSOH*	316	316	

* El error se introdujo en uno de los octetos de *B2*

Tabla 6.17: Estímulos y resultados esperados

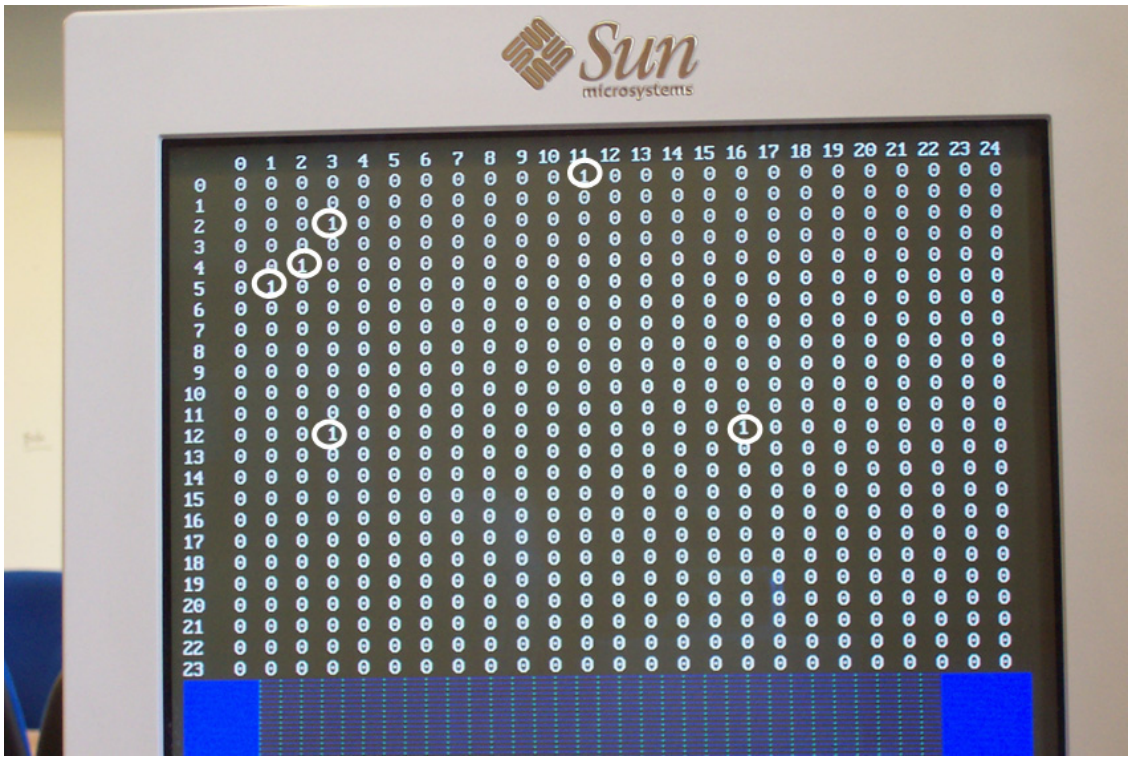


Figura 6.23: Resultados obtenidos para la señal *error_B1* en el test 08

Figura 6.24: Resultados obtenidos para la señal *error_B2* en el test 08

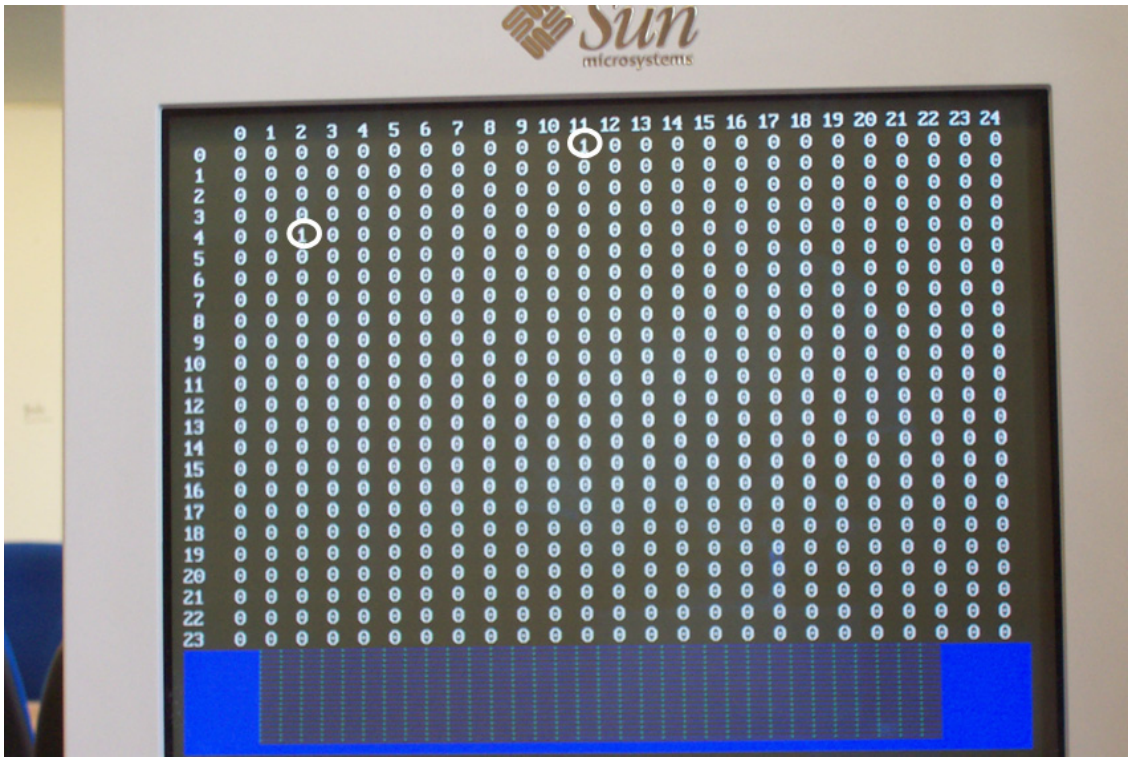


Figura 6.25: Resultados obtenidos para la señal *error_B3* en el test 08

Resultados obtenidos:

- Se muestran en las figuras 6.23, 6.24 y 6.25.

Análisis de los resultados:

- Observando los resultados obtenidos se puede verificar que cuando se recibe una trama con un error en la *RSOH*, se activa la señal *error_B1* (tramas 125 y 302).

- Si el error se ha producido en la *MSOH,* se activan las señales *error_B1* y *error_B2* (tramas 52 y 315).

- Si el error se ha producido en el *VC,* se activan las tres señales (tramas 10 y 101).

Conclusión: Comparando los resultados obtenidos con los esperados, se puede decir que el sistema cumple con las especificaciones.

6.4 Resultado de la síntesis del sistema

En la tabla 6.18 se muestra el resumen de los recursos de la *FPGA* utilizados al sintetizar solamente el sistema *receptor* completo. Cabe destacar que en ningún momento se supera el máximo de los recursos disponibles.

Resumen de la Utilización del Dispositivo				
Utilización de la Lógica	**Usado**	**Disponible**	**Utilización**	**Notas**
Número de *Flip Flops* de *Slices*	1,224	15,360	7%	
Número de *LUT*'s de 4 entradas	2,760	15,360	17%	
Distribución de la Lógica				
Número de *Slices* ocupadas	1,612	7,680	20%	
Número de *Slices* que contienen únicamente lógica relacionada	1,612	1,612	100%	
Número de Slices que contienen lógica no relacionada	0	1,612	0%	
Número total de *LUT*'s de 4 entradas	2,890	15,360	18%	
Número utilizado como lógica	2,760			
Número utilizado como *route-thru*	130			
Número de *bonded IOB*'s	93	173	53%	
Flip Flops de *IOB*'s	67			
Número de *GCLK*'s	1	8	12%	
Número total de puertas equivalentes para el diseño	29,837			
Número de puertas *JTAG* adicionales para *IOB*'s	4,464			

Tabla 6.18: Recursos de la *FPGA* utilizados al sintetizar el módulo *receptor*

PARTE IV
CONCLUSIONES

Capítulo 7

Conclusiones

En este capítulo se presentan las conclusiones que se desprenden de este Proyecto Fin de Carrera así como los posibles trabajos futuros a realizar a partir del mismo.

7.1 Conclusiones

El presente Proyecto Fin de Carrera se ha centrado en un transceptor que procesa tramas *STM-16* de 2,48832 Gbps que transportan *VC-4-16c*'s, y más concretamente en el receptor, es decir, en la parte del sistema que se encarga de extraer los contenedores. El principal objetivo de este proyecto ha sido la descripción en *verilog* del receptor *SDH* y su posterior verificación *hardware* en la placa de prototipado *Diligent Spartan-3 Starter Board*.

A nivel de módulo, se ha comprobado que cada uno responde a los distintos estímulos generando las señales de salida esperadas y conmutando correctamente entre sus distintos estados de funcionamiento, si es que éste se rige por una máquina de estados. El número total de tests realizados ha sido 57.

En la verificación a nivel de sistema se han realizado 8 tests con el objeto de verificar que el sistema extrae correctamente los contenedores de la trama, y genera las distintas señales y alarmas.

Una vez realizadas las verificaciones comentadas anteriormente, se ha llegado a la implementación de un sistema que satisface las especificaciones iniciales, cumpliéndose de esta forma los objetivos establecidos en este Proyecto Fin de Carrera.

A continuación se enumeran, a modo de resumen, las principales características del transceptor:

- Procesado a nivel de palabra (4 octetos) de tramas *SDH STM-16* que transportan *VC-4-16c*.
- Extracción de los contenedores de las mismas.
- Captura los octetos de las distintas taras.
- Generación de la siguientes señales:
  - Señales de sincronización: *sinc*, *LOF* y *OOF*.
  - Señales de indicación de alarma: *MS-AIS*, *AU-AIS* y *VC-AIS*.
  - Señales de error *BIP*: *error_B1*, *error_B2* y *error_B3*.
  - Señales de indicación de defecto remoto: *MS-RDI* y *HO-PATH-RDI*.

  - Señal de pérdida de puntero: *AU_LOP*.

  - Señal de no equipado: *dUNEQ*.

  - Señal de interrupción *int* cuando se active *error_B1*, *error_B2*, *error_B3*, *MS_RDI*, *AU_LOP*, *HO_PATH_RDI* ó *dUNEQ*.

- Protocolo *Wishbone* para la lectura por parte de un sistema externo de:

  - Los octetos capturados.

  - Las señales *error_B1*, *error_B2*, *error_B3*, *MS_RDI*, *AU_LOP*, *HO_PATH_RDI* ó *dUNEQ* cuando se active la señal *int*.

7.2 Trabajos futuros

A continuación se detallan posibles trabajos futuros que se pueden realizar a partir de este Proyecto.

Verificación del receptor

A la hora de verificar el receptor dos han sido las dificultades principales encontradas. En primer lugar y como ya se ha comentado antes en este capítulo, según la metodología, las tareas de diseño y verificación deben ser realizadas por personas distintas, lo cual no ha sido posible debido a las características particulares de un Proyecto Fin de Carrera. Además, debido a la limitación temporal del mismo, la verificación a nivel de sistema no ha sido lo más exhaustiva posible, limitándose a comprobar el correcto funcionamiento de las principales características del sistema.

Por todo esto, es recomendable que en el futuro, un ingeniero externo a este proyecto realice una verificación paralela más completa.

Implementar distintas versiones del receptor

El receptor diseñado en el presente proyecto procesa tramas *STM-16* que transportan *VC-4-16c*. Realizando las modificaciones oportunas en el mismo, pueden obtenerse distintos receptores capaces de procesar otras tramas a distintas velocidades.

Implementación del transceptor

Cada uno de los restantes subsistemas del transceptor *SDH* visto en el capítulo 1 pueden ser implementados individualmente en proyectos separados, de la misma forma que se ha diseñado el receptor.

Otro proyecto consistiría en la unión de todos los subsistemas para dar lugar, por fin, al sistema completo.

Implementación de un multiplexor de inserción/extracción

Una red *SDH* está formada de distintos elementos, entre los que se encuentra el multiplexor de inserción/extracción. Este elemento permite extraer e insertar señales de distinto tipo al flujo de datos *SDH* de alta velocidad, como por ejemplo, señales *PDH* y señales *SDH* de menor velocidad, haciendo posible configurar estructuras en anillo.

Para realizar esta función de extraer e insertar señales puede utilizarse el transceptor *SDH* del capítulo 1.

Implementación de transceptores de aplicación específica

Puesto que la *SDH* permite el transporte de diferentes tipos de tráfico, empaquetándolos en contenedores, otro trabajo futuro a realizar podría ser la implementación de transceptores de aplicación específica, como por ejemplo, un transceptor *ATM* o uno *IP* sobre *SDH*. Es decir, las células *ATM* y los paquetes *IP* pueden ser transmitidos a través de una red *SDH*, empaquetándolos previamente en contenedores.

PARTE V
BIBLIOGRAFÍA

Bibliografía

[1] Ming-Chwan Chow. *"Understanding SONET/SDH. Standars and Applications"*. Andan Publisher, 1995.

[2] Manuel Pascual Lastra. *"Jerarquía Digital Síncrona (conceptos básicos, equipos y estructuras de red)"*. Libro de formación de Telefónica. Febrero, 1997.

[3] Recomendación *UIT-T G.707*. *"Interfaz de nodo de red para la Jerarquía Digital Síncrona"*. Aprobada el 19 de marzo de 1996.

[4] Recomendación *UIT-T G.831*. *"Capacidades de gestión de las redes de transporte basadas en la Jerarquía Digital Síncrona"* Aprobada el 27 de agosto de 1996.

[5] Byeong Gi Lee y Seok Chang Kim. *"Scrambling Techniques for Digital Transmision, Telecommunications Network and Computer System"*. SpringeLverlay, 1995.

[6] A.D. Houghton *"The Engineer's Error Coding Handbook"*. Chapman & Hall, 1997.

[7] Samir Palnitkar. *"Verilog® HDL: A Guide to Digital Design and Sythensis"*. Prentice Hall, 1996.

[8] Donald E. Thomas y Philip Moorby. *"The Verilog Hardware Description Language"*. Kluwer Academic Publishers, 1993.

[9] Cliff Cummings. *"Nonblocking Assignments in Verilog Synthesis; Coding Styles that Kill!"*. Sunburst Design, Inc.

[10] Jacob Nielsen. *"Coding style for TeraStream Verilog Design"*. Vitesse Semiconductor Corporation, 2000.

[11] Xilinx *"Spartan-3 FPGA Family: Complete Data Sheet"*. 2006. [ONLINE]: *http://direct.xilinx.com/bvdocs/publications/ds099.pdf*.

[12] Xilinx *"Spartan-3 Starter Kit Board User Guide"*. 2005. [ONLINE]: *http://www.digilentinc.com/Data/Products/S3BOARD/S3BOARD-rm.pdf*.

[13] OpenCores WebSite: *http://www.opencores.org*.

[14] Proyecto de Fin de Carrera: *"Estudio y Descripción Sintetizable de un Transceptor SDH para Redes de Alta Velocidad"*. Autor: D. Rubén Pascual Arteaga Mesa. Tutores: Dr. Roberto Esper-Chaín Falcón, Dr. Félix B. Tobajas Guerrero. 2002.

PARTE VI
PRESUPUESTO

Presupuesto

Este capítulo está destinado al cálculo del coste económico que supone la realización de este Proyecto Fin de Carrera.

Introducción

El cómputo de los costes de ingeniería de este proyecto ha sido elaborado a partir de la normativa: *"Baremos orientativos para el cálculo de honorarios en el año 2006"*, publicada por el Colegio Oficial de Ingenieros Técnicos de Telecomunicación.

Esta propuesta establece que para trabajos tarificados según el tiempo empleado en su ejecución, sea aplicada la siguiente fórmula:

$$H = H_n \times 65 + H_e \times 78$$

Donde: H = Honorarios

H_n = Horas trabajadas en jornada laboral

H_e = Horas trabajadas fuera de la jornada laboral

Estos honorarios tendrán una reducción en función del número de horas, aplicando los coeficientes de reducción (C) establecidos en la siguiente tabla:

Número de horas (h)	Coeficiente
≤ 36	1,00
36-72	0,90
72-108	0,80
108-144	0,70
144-180	0,65
180-360	0,60
360-510	0,55
510-720	0,50
720-1080	0,45
> 1080	0,40

Los honorarios que se obtengan por la aplicación de la clave *H* se reducirán a medida que aumente el número de horas trabajadas, a cuyo efecto serán multiplicados por los coeficientes reductores con arreglo a la tabla anterior.

Cálculo del presupuesto

Para justificar los gastos que se han realizado, se procede a desglosar el montante total del presupuesto del proyecto en seis apartados, donde el sexto corresponde con el importe total del mismo.

Costes de ingeniería

Coste asociado al tiempo invertido para el desarrollo del presente proyecto. La tarifa aplicada coincide con la estipulada para un ingeniero junior, que consiste en 65 €/hora. Se consideran 8 horas diarias, del tipo de jornada laboral normal, por lo que el cálculo de honorarios es:

Costes de Ingeniería			
Concepto	**Tiempo**	**Coste mensual**	**Importe**
Análisis del problema	2 meses	40 x 4 x 65 = 10.400 €	20.800,00 €
Desarrollo	5 meses	40 x 4 x 65 = 10.400 €	52.000,00 €
Documentación	2 meses	40 x 4 x 65 = 10.400 €	20.800,00 €
Coste			93.600,00 €
Coste total Aplicando el coeficiente de reducción C = 0,4 para trabajos valorados en más de 1.080 horas			**37.440,00 €**

En definitiva, se necesitaron en total 1.400 horas, ascendiendo el coste total referente a costes de ingeniería, a la cantidad de **37.440,00 €**.

Recursos *hardware* y *software*

Los recursos *hardware* y *software* utilizados para la realización de este proyecto incluyen los ya mencionados en el apartado 4.3: *Metodología de diseño empleada en este proyecto*.

Coste de los recursos *hardware* y *software*			
Concepto	**Tiempo de Uso**	**Coste**	**Importe**
Estación de Trabajo SunBlade 150	9 meses	3.366,66 €/3 años	841,67 €
Monitor Sun MicroSystems 21" Flat CRT Monitor Model# GDM-5510	2 meses	117.8 €/3 años	6,54 €
Placa *Digilent Spartan-3 Starter Board*	2 meses	150,00 €	150,00 €
Entorno de diseño *Cadence Design Systems*: Incluye *Verilog-XL 05.10.003-s* y *SimVision 05.10-s12*	3 meses	Lincencia: 2.400 € Mantenimiento: 1.400,00 €/año	2.400 € 350 €
Microsoft Windows XP	4 meses	160,00 €	160,00 €
Microsoft Office XP	2 meses	639,00 €	639,00 €
Coste total			**4.547,21 €**

Personal de laboratorio

Este es el coste generado por el personal empleado para el mantenimiento de las herramientas y las estructuras necesarias. El personal está compuesto por dos técnicos a tiempo completo para un total de cien usuarios.

Coste del personal de laboratorio		
Concepto	**Coste**	**Importe**
Dos técnicos a tiempo completo en proporción a 100 usuarios en un tiempo de 9 meses	36.061,00 €/año	270,46 €
Coste total		**270,46 €**

Material fungible

A esta sección corresponden los gastos asociados al material utilizado para la realización del proyecto, como son el papel, el tóner de la impresora, *CD-ROM*'s para copias de seguridad, etc. Este valor asciende a la cantidad de **250,00 €**.

Importe total

En esta sección se lleva a cabo el cálculo total del coste de este proyecto, en función de los costes parciales indicados en las secciones anteriores. 69649,58

Coste de los recursos *hardware* y *software*	
Concepto	**Importe**
Costes de ingeniería	37.440,00 €
Recursos *hardware* y *software*	4547,21 €
Personal de laboratorio	270,46 €
Material fungible	250,00 €
Coste total	**42.507,67 €**

PRESUPUESTO

D. Oliverio Ramírez Santana declara que:

El proyecto *"Síntesis e Implementación de un Receptor SDH en FPGA Spartan-3"* asciende a un total de CUARENTA Y DOS MIL QUINIENTOS SIETE EUROS Y SESENTA Y SIETE CÉNTIMOS (42.507,67 €).

Fdo.: Oliverio Ramírez Santana
Las Palmas de Gran Canaria, 2007

PARTE VII
APÉNDICES

Apéndice A

Contenido del *CD-ROM*

El contenido del *CD-ROM* se ha dividido en cuatro directorios principales: uno para el formato electrónico de la memoria, otro para la descripción *RTL* del sistema, otro para su síntesis y el último para la verificación del mismo.

Directorio *memoria*

Contiene la memoria del proyecto en formato *pdf* (*Portable Document Format*).

Directorio *RTL*

En este directorio se presenta la descripción *RTL* del sistema completo y de cada uno de los módulos que lo forman.

El archivo que contiene el módulo *receptor.v* se almacena en este directorio y el resto se organizan en 7 subdirectorios, como muestra la figura A.1. Cada subdirectorio contiene la descripción *RTL* del módulo con su mismo nombre y de los submódulos de los que éste último consta.

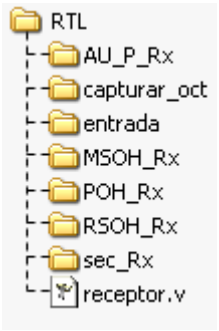


Figura A.1: Contenido del directorio *RTL*

Directorio *síntesis*

En este directorio se incluyen todos los módulos que componen el receptor y un subdirectorio denominado *sintetiza*, que almacena los archivos generados por las herramientas incluidas en el *software ISE* de *Xilinx* al crear un proyecto y sintetizar el receptor, entre los que destaca el archivo *T_receptor_summary.html*, que es una tabla resumen de los recursos de la *FPGA* utilizados.

Directorio *verificacion*

Este directorio se divide a su vez en otros dos: el directorio *modulo* y el directorio *sistema*, cuyos contenidos se detallan a continuación.

modulo

En este directorio se almacenan todos los tests realizados a nivel de módulo (57 en total) organizados en 57 subdirectorios conteniendo cada uno los archivos correspondientes a un test.

En la figura A.2 se puede ver el contenido de uno de estos subdirectorios, que contiene el fichero del módulo a testear (en el ejemplo *calcula_B1.v*), el fichero *generador.v* del módulo con el mismo nombre y el fichero *tb_top.v* del módulo de nivel superior que conecta a los dos anteriores. Además hay también un fichero llamado *descripcion.txt* con una breve descripción del test.

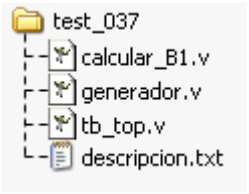


Figura A.2: Contenido de uno de los subdirectorios del directorio *modulo*

sistema

Este directorio esta subdividido en 8 subdirectorios más correspondientes a los 8 tests realizados a nivel de sistema.

Cada subdirectorio contiene todos los archivos de todos los módulos que componen el receptor y de los módulos necesarios para realizar el test. También hay una carpeta llamada *sintetiza*, donde se almacenan todos los archivos generados por las herramientas incluidas en el *software ISE* de *Xilinx* al crear un proyecto y sintetizar e implementar el sistema, entre los que se incluye el fichero *bitstream* necesario para configurar la *FPGA*. Por último existe otra carpeta denominada *resultado*, conteniendo las fotografías de los resultados mostrados en el monitor en la verificación *hardware*.

Apéndice B

Acrónimos

AIS: Señal de Indicación de Alarma

APS: Conmutación de Protección Automática

ATM: Modo de Transferencia Asíncrono

AU: Unidad Administrativa

AU-AIS: Señal de Indicación de Alarma de Puntero

AUG: Grupo de Unidades Administrativas

AU-LOP: Pérdida de Puntero

B6ZS: Código Bipolar de Sustitución de Seis Ceros

B8ZS: Código Bipolar de Sustitución de Ocho Ceros

BIP-X: Paridad con Entrelazado de *X* Bits

C: Contenedor

CAD: Diseño Asistido por Ordenador

CCITT: Comité Consultivo Internacional de Telefonía y Telegrafía. Actualmente *UIT-T*

CEPT-N: Señal Digital de la *UIT-T* de *Nivel N*

CLB: Bloques Lógicos Configurables

CMI: Inversión de Marcas Codificadas

CRU: Unidad de Recuperación de Reloj

DCC: Canales de Comunicaciones de Datos

DCM: Controladores de Reloj Digital

DDR: Doble Velocidad de Datos

DS-N: Señal Digital de Nivel *N*

dUNEQ: No Equipado. No hay configurada una conexión

ECC: Canal de Comunicación Embebido

EOC: Canal de Comunicaciones Embebido

ETSI: Instituto de Telecomunicaciones Europeo de Estandarización

FDDI: Interfaz de Fibra de Datos Distribuidos

FPGA: Red de Puertas Programables

HDB3: Bipolar de Alta Densidad 3

HDL: Lenguaje de Descripción Hardware

HO-PATH-RDI: Indicación de Defecto Remoto del *VC-4*

HS: Sincronismo Horizontal

IEEE: Instituto de Ingenieros Eléctricos y Eletrónicos

IOB: Bloques de Entrada y Salida

LED: Diodo Emisor de Luz

LOF: Pérdida de Trama

LOS: Pérdida de Señal

LSB: Bit/Byte menos significativo

LUT: Tabla de Verdad

MAN: Red de Área Metropolitana

MDF: Multiplexación por División en Frecuencia

MDT: Multiplexación por División en el Tiempo

MS-AIS: Señal de Indicación de Alarma de la Sección de Multiplexación

MSB: Bit/Byte más significativo

MSOH: Tara de Sección de Multiplexación

MSP: Protección para la Sección de Multiplexación

MS-RDI: Indicación de Defecto Remoto de la Sección de Multiplexación

NDF: Bandera de Nuevos Datos

NRZ: No Retorno a Cero

OC-N: Portadora Óptica de Nivel *N*

OH: Redundancia o Tara

OOF: Fuera de Trama

PDH: Jerarquía Digital Plesiócrona

PLD: Dispositivos Lógicos Programables

PLL: Bucle Enganchado en Fase

POH: Tara de Trayecto

PROM: Memoria de Sólo Lectura Programable

PS/2: Puerto serie 2

RAM: Memoria de Acceso Aleatorio

RDSI: Red Digital de Servicios Integrados

ROM: Memoria de Sólo Lectura

RSOH: Tara de Sección de Regeneración

RTL: Lógica de Transferencia de Registro

SD: Degradación de la Señal

SDH: Jerarquía Digital Síncrona

SF: Fallo de Señal

SOH: Tara de Sección

SONET: Red Óptica Síncrona

SPLD: Dispositivos Lógicos Programables Sencillos

STE: Supertrama Extendida

STM-N: Módulo de Transporte Síncrono de Nivel *N*

STS: Señal de Transporte Síncrono

TCM: Monitorización de Conexiones Tándem

TU: Unidad Tributaria

TUG: Grupo de Unidades Tributarias

U: Unidad

UIT-T: Unión Internacional de Telecomunicaciones - Sector de Estandarización de Telecomunicaciones

VC: Contenedor Virtual

VC-AIS: Señal de Indicación de Alarma de *VC*

VDC: Voltios de Corriente Continua

VGA: Matriz Gráfica de Vídeo

VHDL: Lenguaje de Descripción *Hardware* para Circuitos Integrados de Muy Alta Velocidad

VS: Sincronismo Vertical

Apéndice C

Protocolo *Wishbone B.3*

OpenCores es un grupo de estudiantes, profesionales, empresas y, en general, de cualquier persona interesada en el desarrollo *hardware*, cuya actividad se lleva a cabo en su sitio *web*: *http://www.opencores.org*.

Estas personas contribuyen al desarrollo de *OpenCores* aportando módulos digitales llamados núcleos, escritos en cualquier lenguaje de descripción *hardware*, preferiblemente *Verilog* o *VHDL* puesto que son los más utilizados. De esta forma, cualquier persona que necesite un núcleo que realice alguna función para incluirlo en su propio diseño puede obtenerlo de la *web* de *OpenCores* totalmente gratis. *OpenCores* es un grupo dinámico, es decir, no hay ninguna afiliación formal entre las personas que contribuyen.

El bus *SoC* (sistemas en chip) más utilizado para la interconexión entre los núcleos es el *Wishbone*, principalmente porque es totalmente gratis. Su propósito es fomentar la reutilización de los diseños para aliviar problemas de integración *SoC*. Esto se consigue

creando una interfaz común entre núcleos, lo que mejora la portabilidad y fiabilidad del sistema. Adoptando un esquema de interconexión estándar, los núcleos pueden integrarse más rápido y fácil.

En los siguientes apartados se describen las especificaciones del protocolo *Wishbone*, revisión *B.3* utilizado en este proyecto. Para más información pueden consultarse la referencias [12] y [13] de la bibliografía.

C.1 Interfaz *Wishbone*

En la figura C.1 se muestra la interfaz del esclavo. Todas las señales son síncronas con el flanco de subida de la señal de reloj *clk_Rx*, incluida *mr*, y activadas a nivel alto. A continuación se describen dichas señales:

- *mr*: Reset de entrada. Se ha utilizado el mismo que para el receptor. Resetea la interfaz *Wishbone*.

- *clk_Rx*: Reloj de entrada. Se ha utilizado el mismo que para el receptor. Coordina todas las actividades de la lógica interna en la interfaz *Wishbone*.

- *ADR_I[4:0]*: Vector de direcciones de entrada. Se utiliza para pasar una dirección binaria de 5 bits. Está dirección sólo es válida cuando *mr* no esté activa y *CYC_I* y *STB_I* sí lo estén.

- *DAT_I[7:0]*: Vector de datos de entrada. Se utiliza para recibir datos binarios de 8 bits cuando *CYC_I*, *STB_I* y *WE_I* estén activas y *mr* no.

- *DAT_O[7:0]*: Vector de datos de salida. Se utiliza para recibir datos binarios de 8 bits cuando *CYC_I* y *STB_I* estén activas y *mr* y *WE_I* no. El dato será válido cuando *ACK_O* esté activa.

- *WE_I*: Señal de habilitación de escritura de entrada. Indica ciclo de escritura cuando está activa y de lectura en caso contrario. Sólo se tendrá en cuenta cuando *mr* no está activa y *CYC_I* y *STB_I* sí lo están.

- *STB_I*: Señal *strobe* de entrada. Se utiliza para capacitar a otras señales de la interfaz. *mr* no esté activa y *CYC_I* sí lo esté.

- *ACK_O*: Señal de reconocimiento de salida. Se activa para indica la terminación normal de un ciclo de bus, siempre que lo estén las señales *CYC_I* y *STB_I*.

- *CYC_I*: Señal de ciclo de entrada. Cuando se activa, indica que un ciclo de bus válido está en progreso manteniéndose activa durante todo el ciclo de bus. El esclavo sólo tendrá en cuenta el resto de señales del protocolo *Wishbone* cuando *CYC_I* permanezca activa.

En la tabla C.1 se muestran los nombres para las señales utilizados en este proyecto y su señal correspondiente en el protocolo *Wishbone*.

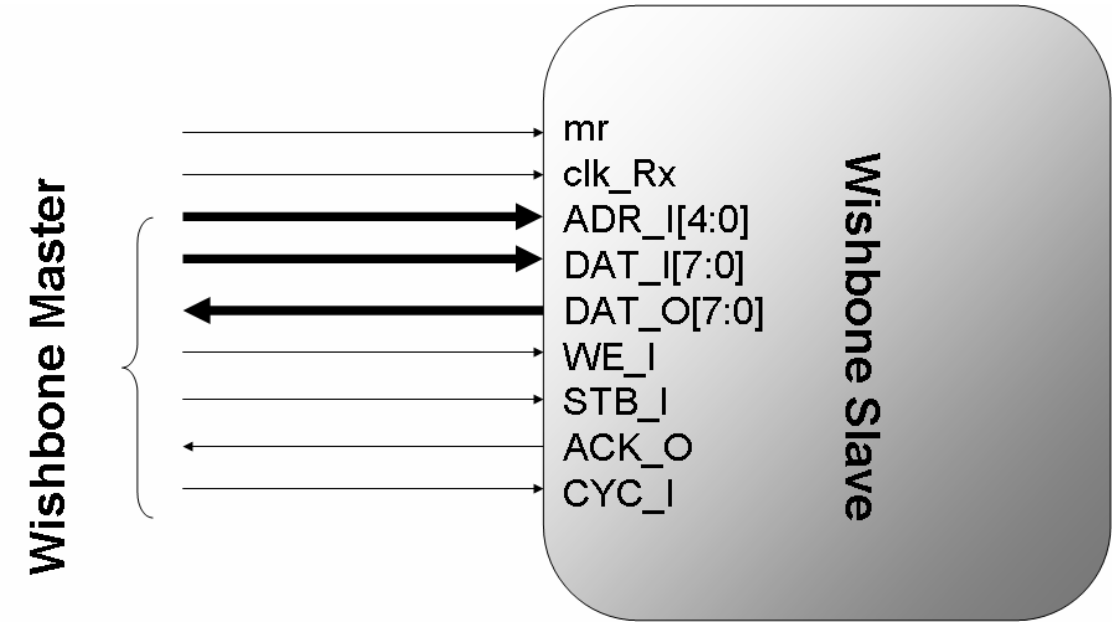


Figura C.1: Interfaz del protocolo *Wishbone*

Nombre	Señal *Wishbone*
clk_Rx	CLK_I
mr	RST_I
ADR_I[4:0]	ADR_I(4..0)
DAT_I[7:0]	DAT_I(7..0)
DAT_O[7:0]	DAT_O(7..0)
WE_I	WE_I
SEL_I	SEL_I
STB_I	STB_I
ACK_O	ACK_O
CYC_I	CYC_I

Tabla C.1: Nombre de las señales y señal *Wishbone* correspondiente

C.2 Ciclos de operación

En los ciclos de operación se utiliza un protocolo *handshaking* entre el maestro y el esclavo, como el mostrado en la figura C.2. Cuando el maestro está preparado para la transferencia de un dato activa la señal *STB_I*. El receptor responderá activando la señal *ACK_O* cuando esté preparado también. Se realiza la transferencia y el maestro desactiva *STB_I* para finalizar la operación.

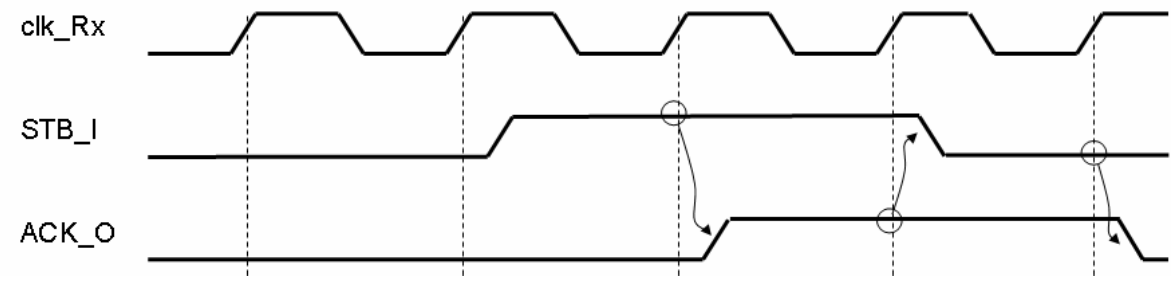


Figura C.2: Protocolo *Handshaking*

C.2.1 Ciclo de lectura

La figura C.3 muestra un ciclo de lectura. El protocolo funciona de la siguiente forma:

- Flanco de reloj 0:
  - El maestro presenta una dirección válida en *ADR_I[4:0]*.
  - El maestro desactiva *WE_I* para indicar un ciclo de lectura.
  - El maestro activa *CYC_I* para indicar el comienzo de un ciclo.
  - El maestro activa *STB_I* para indicar el comienzo de la primera fase.

- Flanco de reloj 1:
  - El esclavo responde activando la señal *ACK_O*.
  - El esclavo presenta un dato válido en *DAT_O[7:0]*.

- Flanco de reloj 2:
  - Al estar *ACK_O*, el maestro captura el dato en *DAT_O[7:0]*.
  - El maestro presenta otra dirección válida en *ADR_I[4:0]* para realizar otra lectura, manteniendo *CYC_I* y *STB_I* activadas y *WE_I* desactivada.

- Flanco de reloj 3:
  - El esclavo mantiene la señal *ACK_O* activada.
  - El esclavo presenta otro dato válido en *DAT_O[7:0]*.

- Flanco de reloj 4:
  - Al estar *ACK_O*, el maestro captura el dato en *DAT_O[7:0]*.
  - El maestro mantiene *CYC_I* activada para no finalizar todavía el ciclo.
  - El maestro desactiva *STB_I* para introducir estados de espera (-*EE*-).

- Flanco de reloj 5:
  - El receptor responde desactivando *ACK_O*.

- Flanco de reloj 6:
  - El maestro presenta otra dirección válida en *ADR_I[4:0]* para realizar otra lectura, manteniendo *CYC_I* activada y *WE_I* desactivada.
  - El maestro activa *STB_I*.
-
- Flanco de reloj 7:
  - El esclavo responde activando la señal *ACK_O*.
  - El esclavo presenta un dato válido en *DAT_O[7:0]*.

- Flanco de reloj 8:
  - Al estar *ACK_O*, el maestro captura el dato en *DAT_O[7:0]*.
  - El maestro presenta otra dirección válida en *ADR_I[4:0]* para realizar otra lectura, manteniendo *CYC_I* y *STB_I* activadas y *WE_I* desactivada.

- Flanco de reloj 9:
  - El esclavo mantiene la señal *ACK_O* activada.
  - El esclavo presenta otro dato válido en *DAT_O[7:0]*.

- Flanco de reloj 10:
  - Al estar *ACK_O*, el maestro captura el dato en *DAT_O[7:0]*.
  - El maestro finaliza el ciclo desactivando *CYC_I* y *STB_I*.

- Flanco de reloj 11:
  - El esclavo responde desactivando *ACK_0*.

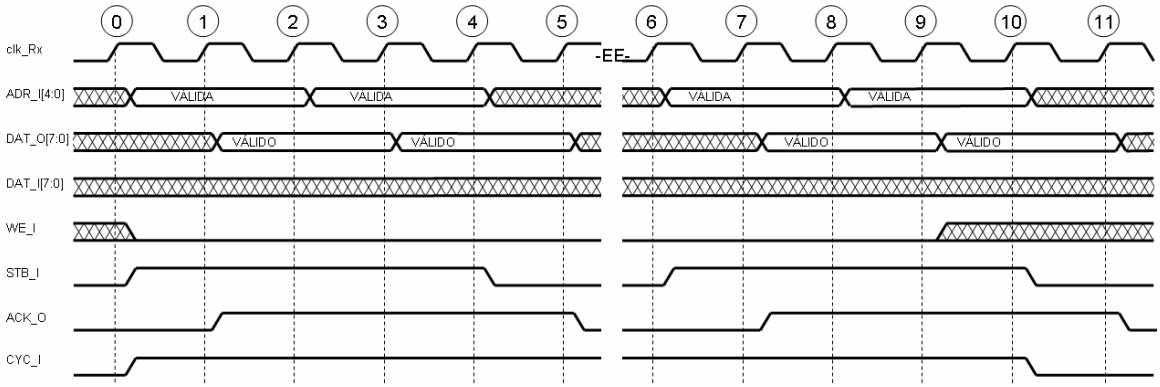


Figura C.3: Ciclo de lectura

C.2.2 Ciclo de escritura

La figura C.4 muestra un ciclo de escritura. El protocolo funciona de la siguiente forma:

- Flanco de reloj 0:
  - El maestro presenta una dirección válida en *ADR_I[4:0]*.
  - El maestro presenta un dato válido en *DAT_I[7:0]*.
  - El maestro activa *WE_I* para indicar un ciclo de escritura.
  - El maestro activa *CYC_I* para indicar el comienzo de un ciclo.
  - El maestro activa *STB_I* para indicar el comienzo de la primera fase.

- Flanco de reloj 1:
  - El esclavo responde activando la señal *ACK_O*.

- Flanco de reloj 2:
  - El esclavo captura el dato en *DAT_I[7:0]*.
  - El maestro presenta otra dirección válida en *ADR_I[4:0]* y otro dato válido en *DAT_I[7:0]* para realizar otra lectura, manteniendo *CYC_I*, *STB_I* y *WE_I* activadas.

- Flanco de reloj 3:
  - El esclavo mantiene la señal *ACK_O* activada.

- Flanco de reloj 4:
  - El esclavo captura el dato en *DAT_I[7:0]*.
  - El maestro mantiene *CYC_I* activada para no finalizar todavía el ciclo.
  - El maestro desactiva *STB_I* para introducir estados de espera (*-EE-*).

- Flanco de reloj 5:
  - El receptor responde desactivando *ACK_O*.

- Flanco de reloj 6:
  - El maestro presenta otra dirección válida en *ADR_I[4:0]* y otro dato válido en *DAT_I[7:0]* para realizar otra lectura, manteniendo *CYC_I* y *WE_I* activadas.
  - El maestro activa *STB_I*.

- Flanco de reloj 7:
  - El esclavo responde activando la señal *ACK_O*.

- Flanco de reloj 8:
  - El esclavo captura el dato en *DAT_I[7:0]*.
  - El maestro presenta otra dirección válida en *ADR_I[4:0]* y otro dato válido en *DAT_I[7:0]* para realizar otra lectura, manteniendo *CYC_I*, *STB_I* y *WE_I* activadas.

- Flanco de reloj 9:
  - El esclavo mantiene la señal *ACK_O* activada.

- Flanco de reloj 10:
  - El esclavo captura el dato en *DAT_I[7:0]*.
  - El maestro finaliza el ciclo desactivando *CYC_I* y *STB_I*.

- Flanco de reloj 11:
  - El esclavo responde desactivando *ACK_0*.

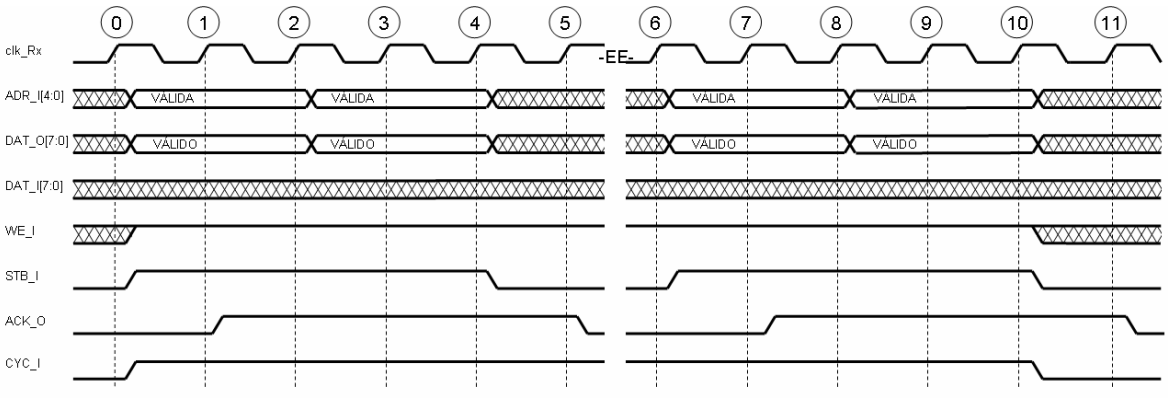


Figura C.4: Ciclo de escritura

Apéndice D

Pliego de condiciones

El objetivo de este pliego de condiciones es dar los requisitos *hardware* y *software* y el procedimiento necesarios para poder llevar a cabo las simulaciones y las verificaciones *hardware* de los tests incluidos en el CD-ROM del presente Proyecto Fin de Carrera.

D.1 Requisitos

En la tabla D.1 se detallan los recursos *hardware* y *software* utilizados para realizar los tests.

	Recursos *hardware*	Recursos *software*
Verificación simulada	- Estación de Trabajo *SunBlade 150*	- Sistema operativo *Solaris* - Entorno de diseño *Cadence Design Systems*: Incluye *Verilog-XL 05.10.003-s* y *SimVision 05.10-s12*
Verificación *hardware*	- Estación de Trabajo *SunBlade 150* - Monitor *Sun MicroSystems* 21" *Flat CRT Monitor Model# GDM-5510* - Placa *Digilent Spartan-3 Starter Board*	- Sistema operativo *Solaris* - Sistema operativo *Microsoft Windows XP* - Herramienta de síntesis *Xilinx ISE 8.1.03i* - Herramienta de configuración *iMPACT 6.3i*

Tabla D.1: Recursos utilizados para realizar los tests

D.2 Procedimiento para realizar los tests

Para llegar a realizar con éxito los tests incluidos en el CD-ROM hay que seguir una serie de pasos, que dependerán de si la verificación es simulada o *hardware*. En los dos siguientes apartados se explica en detalle el procedimiento para ambas. Para cada paso de cada procedimiento se enumeran los recursos necesarios y los pasos en que, a su vez, se dividen.

D.2.1 Verificación simulada

Los pasos a seguir para realizar una simulación son dos: primero se simula el test y después se visualizan los resultados obtenidos. A continuación se explicarán dichos pasos con la ayuda de un ejemplo, el test 033 situado en el directorio *verificacion\modulo\test_033* del CD-ROM.

D.2 PROCEDIMIENTO PARA REALIZAR LOS TESTS

1º *Simulación*

Recursos:

- Estación de trabajo *SunBlade 150*.
- Sistema operativo *Solaris*.
- Compilador y simulador *Verilog-XL 05.10.003-s*.

Procedimiento

- En primer lugar hay que copiar el directorio *test_033* en una unidad de disco de la estación de trabajo, ya que la simulación va a generar algunos archivos.

- Seguidamente se abre una consola de *Solaris* y desde la línea de comandos se ejecuta el simulador *Verilog*, pasándole como argumentos los archivos que forman parte de la simulación:

```
verilog tb_top.v generador.v estados_entrada.v
```

- Cuando el *software* termine de simular dará como resultado el archivo *tb.trn*, que no es más que una base de datos que incluye todas las señales de la simulación así como los valores que toman las mismas.

2º *Visualizar los resultados*

Recursos:

- Estación de trabajo *SunBlade 150*.
- Sistema operativo *Solaris*.
- Visor de formas de onda *Simvision 05.10-s12*.

Procedimiento

- Primera se ejecuta la aplicación que permite visualizar la base de datos obtenida en el apartado anterior, el *Simvision*, tecleando lo siguiente en la línea de comandos de la consola:

```
simvision
```

Seguidamente aparece en pantalla el explorador de diseño (ver figura D.1).

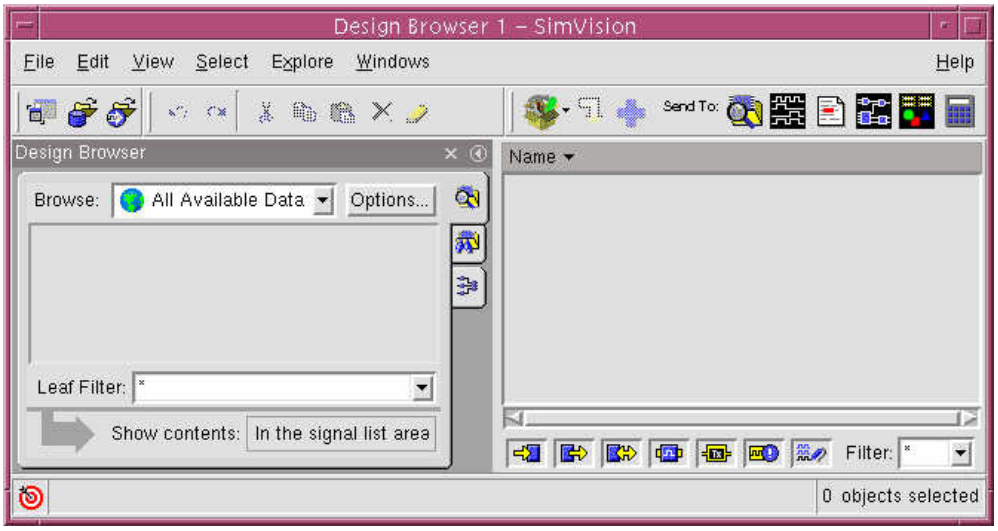


Figura D.1: Explorador de diseño del *Simvision*

- Lo siguiente será abrir el archivo *tb.trn* obtenido en el paso de simulación. Para ello hay que ejecutar el comando *Open a database…* del menú *File*. Se selecciona la base de datos *tb.trn* y se pulsa el botón *Open*. Ahora el explorador de diseño tiene la apariencia que muestra la figura D.2, donde, a la izquierda del mismo, aparecen los módulos que forman parte del sistema (del test, en este caso) y, a la derecha, las señales del módulo que esté seleccionado.

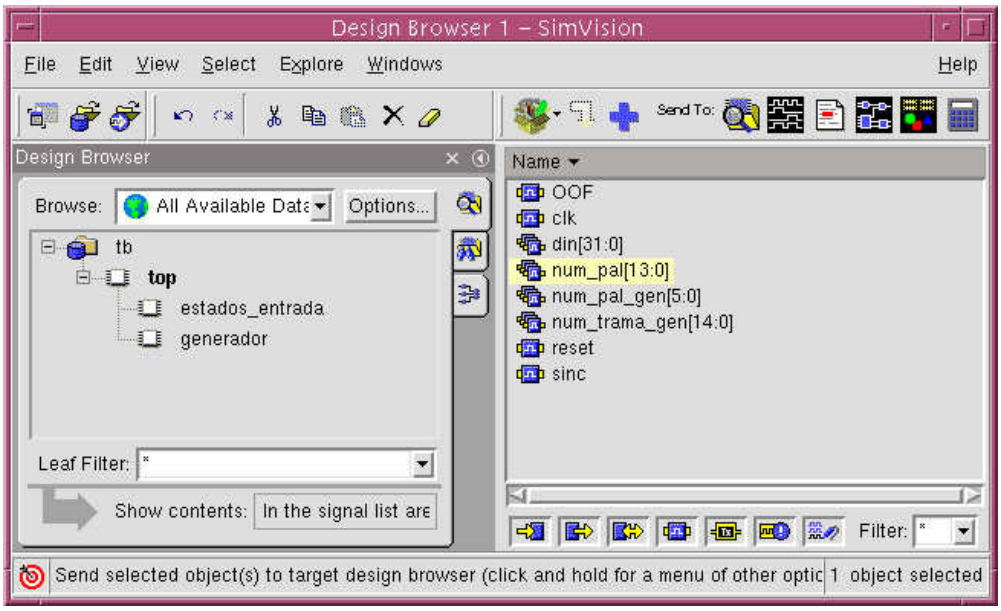


Figura D.2: Explorador de diseño del *Simvision* tras abrir una base de datos

- Para ver la forma de onda en el tiempo de las señales que se desean verificar, hay que seleccionarlas en el explorador de diseño y seguidamente hacer clic en el icono  , tras lo cual, se abre una ventana nueva denominada visor de formas de onda (figura D.3), donde se muestran las formas de ondas en el tiempo de las señales seleccionadas. Si se desean añadir más señales, no hay más que volver al explorador de diseño, seleccionarlas y volver a hacer clic en el icono  .

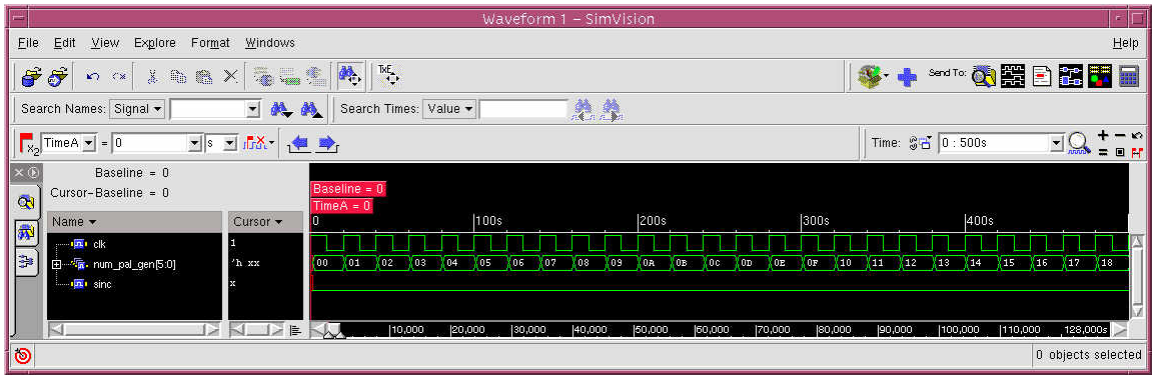


Figura D.3: visor de formas de onda del *Simvision*

D.2.2 Verificación *hardware*

Los pasos a seguir para realizar la verificación *hardware* son cuatro: primero se sintetiza el sistema que compone el test, después se implementa el diseño, más tarde se genera el fichero de programación de la *FPGA* y por último se programa para visualizar los resultados obtenidos. Puesto que en el CD-ROM se incluyen todos los ficheros generados en cada uno de estos pasos, se pueden omitir los tres primeros y pasar directamente al cuarto para ver los resultados.

A continuación se explicarán dichos pasos con la ayuda de un ejemplo, el test 02 situado en el directorio *verificacion\sistema\test_02* del CD-ROM.

1º *Síntesis*

Recursos:

- Estación de trabajo *SunBlade 150*.
- Sistema operativo *Solaris*.
- Herramienta de síntesis *Xilinx ISE 8.1.03i*.

Procedimiento

- En primer lugar hay que copiar el directorio *test_02* en una unidad de disco de la estación de trabajo, ya que en los pasos de síntesis, de implementación y generación del fichero de programación se van a generar algunos archivos.

- Seguidamente se abre una consola de *Solaris* y desde la líneas de comandos se ejecuta el *ISE*:

```
ISE
```

  Se abrirá el explorador de proyectos (Figura D.4).

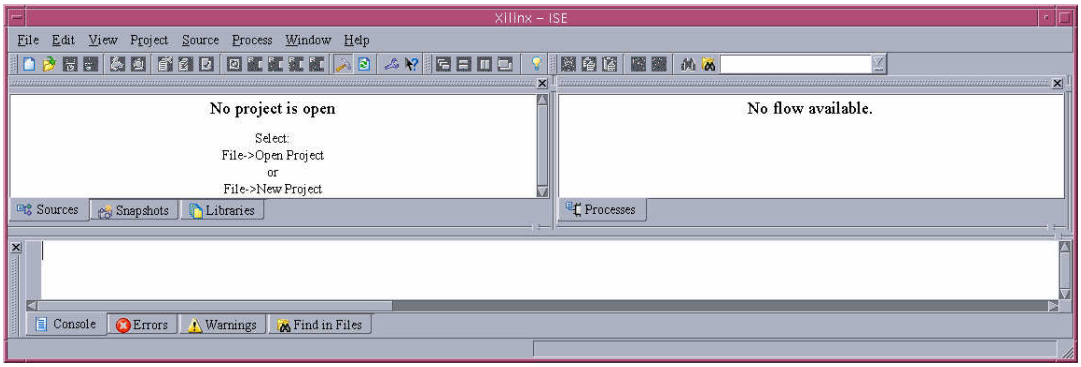


Figura D.4: Explorador de proyectos del *ISE*

- Ahora hay que abrir el archivo de proyecto *verifica.ise*, situado en la carpeta *verifica*, seleccionando el comando *Open Project…* del menú *File*. El explorador de proyectos tendrá la apariencia mostrada en la figura D.5.

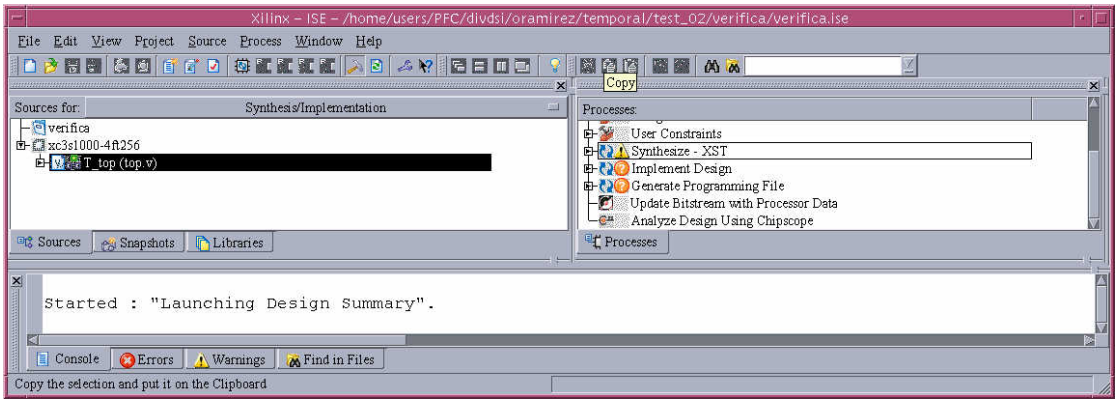


Figura D.5: Explorador de proyectos del *ISE* tras abrir un proyecto

- Seleccionando en la ventana *fuentes* (a la izquierda de la figura D.5) el módulo de nivel superior del test, *T_top*, aparecerá en la ventana *procesos* (a la derecha de la figura D.5) los procesos que se pueden ejecutar sobre el mismo.

- Para sintetizar el sistema habrá que hacer clic con el botón derecho del ratón sobre el proceso *Synthesize - XST* y seleccionar el comando *Rerun* del menú emergente que aparece.

- La síntesis habrá terminado cuando aparezca en la ventana *Transcripción* el siguiente mensaje:

```
Process "Synthesize" completed successfully
```

2º *Implementar el diseño*

Recursos:

- Estación de trabajo *SunBlade 150*.
- Sistema operativo *Solaris*.
- Herramienta de síntesis *Xilinx ISE 8.1.03i*.

Procedimiento

- Para implementar el diseño se selecciona el módulo *T_top*, en la ventana *fuentes*, se hace clic con el botón derecho del ratón sobre el proceso *Implement Design*, en la ventana *procesos*, y se selecciona el comando *Rerun* del menú emergente que aparece. Con esto se logra que se realice la traducción, el mapeo y el *Placement&Route* del diseño.

- La implementación habrá terminado cuando aparezca en la ventana *Transcripción* el siguiente mensaje:

```
Process "Generate Post-Place & Route Static Timing" completed
successfully
```

3º *Generar el fichero de programación*

Recursos:

- Estación de trabajo *SunBlade 150*.
- Sistema operativo *Solaris*.

- Herramienta de síntesis *Xilinx ISE 8.1.03i*.

Procedimiento

- Para generar el fichero de programación se selecciona el módulo *T_top*, en la ventana *fuentes*, se hace clic con el botón derecho del ratón sobre el proceso *Generate Programming File*, en la ventana *procesos*, y se selecciona el comando *Rerun* del menú emergente que aparece.

- La generación del fichero de programación habrá terminado cuando aparezca en la ventana *Transcripción* el siguiente mensaje:

```
Process "Generate Programming File" completed successfully
```

  En este instante se habrá creado el fichero *T_top.bit* que será cargado en la *FPGA* para su programación.

4º Programación de la *FPGA*

Recursos:

- Estación de trabajo *SunBlade 150*.
- Monitor *Sun MicroSystems* 21".
- Placa de prototipado Placa *Digilent Spartan-3 Starter Board*, que incluye el cable *JTAG*-paralelo de bajo coste.
- Sistema operativo *Microsoft Windows XP*.
- Herramienta de configuración *iMPACT 6.3i*.

Procedimiento

- En primer lugar hay que conectar el puerto paralelo de la tarjeta *SunPCI* de la estación de trabajo al puerto *JTAG* de la placa de prototipado, mediante el cable de bajo coste, y el monitor al puerto *VGA* de la misma.

- Seguidamente, desde el *Windows XP*, se ejecuta el *software iMPACT*. Aparecerá un asistente en el que habrá que seleccionar las siguientes opciones:

  - *Configure Devices*
  - *Boundary-Scan Mode*
  - *Automatically connect to cable and identify Boundary-Scan chain*

- El *software* detectará dos dispositivos, la *FPGA* y la memoria *PROM*, de los que dispone la placa. Aparecerá en pantalla la ventana de la figura D.6, donde se encuentra un símbolo para la *FPGA* y otro para la memoria, y el *software* preguntará si se desea asociar un archivo a cada uno de los dispositivos. Se responde sí y se selecciona el fichero *T_top.bit* generado en el paso anterior y situado en la carpeta *verifica*. Cuando pida el archivo para la memoria se cancela.

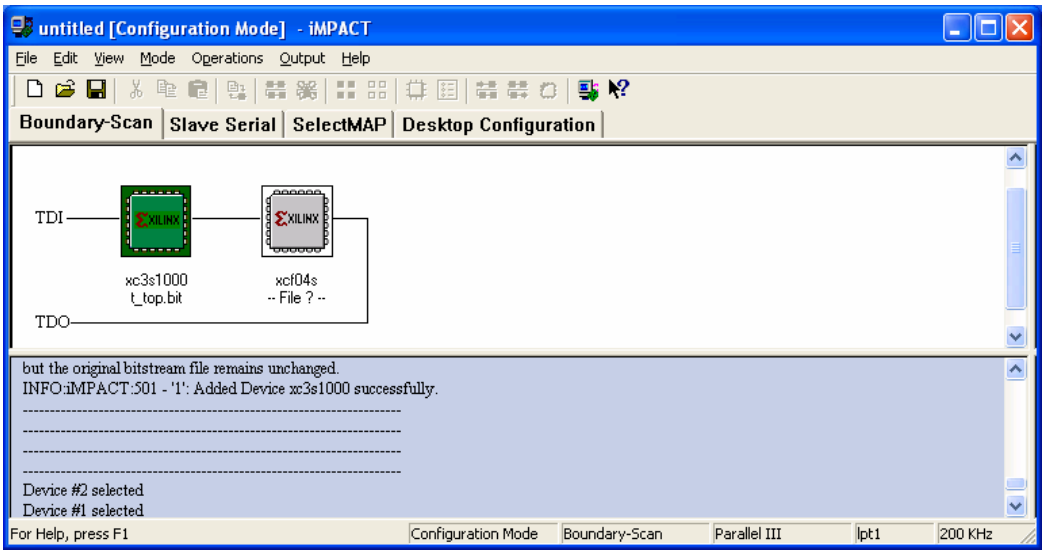


Figura D.6: Herramienta *iMPACT* tras detectar los dos dispositivos

- Ahora que el *iMPACT* ya ha detectado la *FPGA* y le ha asociado un fichero, se procede a su programación. Para ello se hace clic con el botón derecho sobre el símbolo de la *FPGA* y se selecciona el comando *Program…* del menú emergente. En la ventana que aparece se acepta pulsando sobre el botón *Ok* y se

espera a que finalice la programación, indicándose en pantalla con el mensaje de la figura D.7.

Figura D.7: Mensaje mostrado por el *iMPACT* tras la programación de la placa

- En este momento aparecerá en el monitor conectado a la placa de prototipado los resultados del test (figura D.8).

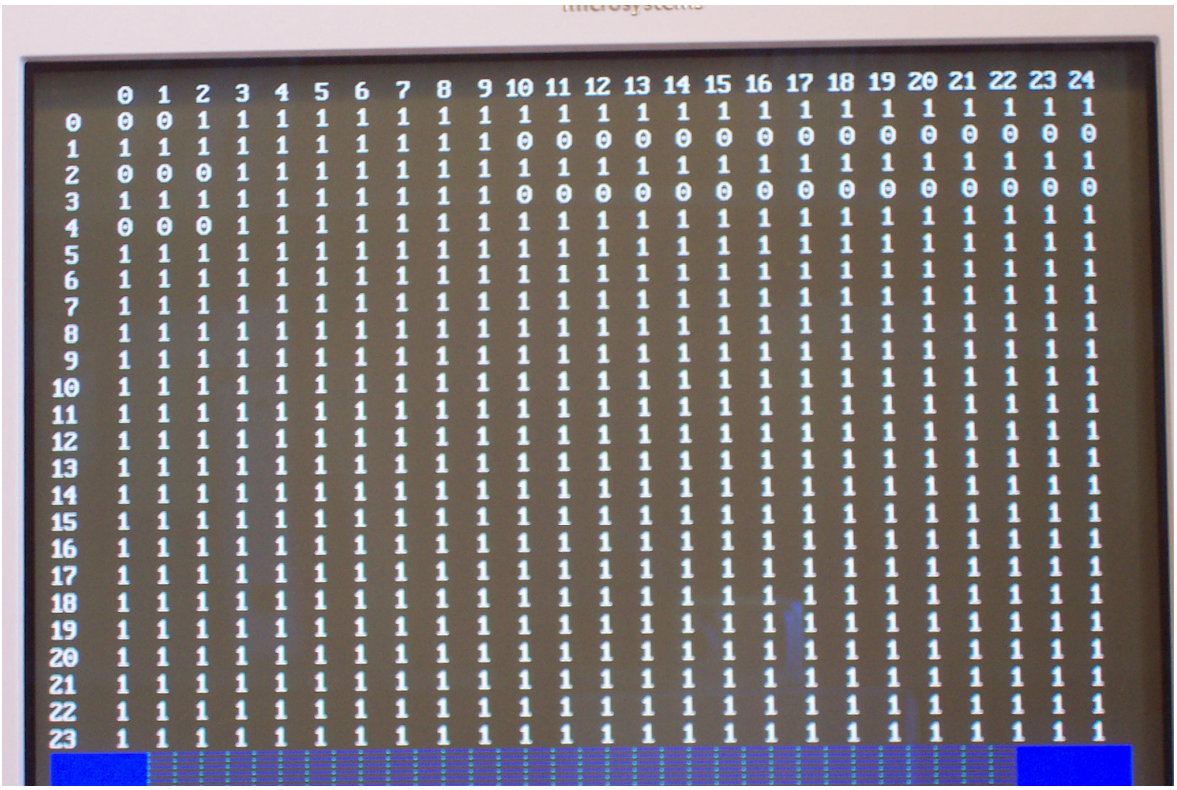


Figura D.8: Resultados de la verificación *hardware*.