Filtro adaptativo tolerante a fallos

José Carlos González Salas

Grado en Ingeniería Informática de Computadores

Facultad de Informática

Departamento DACYA

Universidad Complutense de Madrid

Trabajo fin de Grado en Ingeniería Informática de Computadores

Madrid, 19 de junio de 2014

Director: Óscar Garnica Alcázar

Codirector: Juan Lanchares Dávila

Autorización de difusión y utilización

El autor de este proyecto autoriza a la Universidad Complutense de Madrid a difundir y utilizar el presente trabajo de investigación, tanto la aplicación como la memoria, únicamente con fines académicos, no comerciales y mencionando expresamente a sus autores. También autorizan a la Biblioteca de la UCM a depositar el trabajo en el Archivo Institucional E-Prints Complutense.

Autor

José Carlos González Salas

Fecha

19/06/2014

Índice

Índice de figuras

Índice de tablas

Índice de abreviaturas

ACC - Accumulator

Ack - Acknowledge

ADC - Analog-to-Digital Converter

C2 – Two's Complement

CSMA/CD - Carrier Sense Multiple Access with Collision Detection

DS - Delay Subtract

DSP - Digital Signal Processing

DUT - Design Under Test

DVI-VGA - Digital Visual Interface-Video Graphics Array

E/S - Entrada/Salida

En - Enable

FF - Flip-Flop

FIFO - First In First Out

FMC - FPGA Mezzaine Connectors

FPGA - Field Programmable Gate Array

FSM – Finite State Machine

HPD - High-Pass Deconvolver

Id - Identifier

IP - Intelectual Property

LUT - Lookup Table

MAC - Media Access Controller

MHz - Mega Hercio

MSB - Most Significant Bit

Mux - Multiplexor

NoC - Network on Chip

Ns - Nanosegundos

PHA - Pulse High Analysis

PLB - Processor Local Bus

PSA - Pulse Shape analysis

RAM - Random Access Memory

RTL – Register-Transfer Level

RoHS - Restriction of Hazardous Substances

SCL - Serial Clock

SDA - Serial Data

Slv - Slave

SoC - System on Chip

Tb - Testbench

TCL - Tool Command Language

USB - Universal Serial Bus

V - Voltio

VHDL - Very high speed cirtuit Hardware Description Language

Wr - Write

Resumen

Los detectores de partículas (como el conformador trapezoidal utilizado en este proyecto) son utilizados en una gran variedad de aplicaciones (aceleradores de partículas, satélites artificiales,…). Esta clase de sistemas contienen diversos tipos de filtros digitales y uno de los más ampliamente utilizados es el conformador trapezoidal. Debido a su campo de aplicación, esta clase de sistemas están sometidos a grandes dosis de radiación que pueden llegar a producir daños en el sustrato físico sobre el que se ha diseñado los circuitos y consecuentemente producir fallos en el funcionamiento de dicho circuito. Este proyecto consiste en implementar este tipo de conformador sobre una FPGA Virtex-6 y dotarle de tolerancia a fallos utilizando la capacidad de reconfiguración dinámica parcial que posee esta familia de FPGAs.

Así, se rediseñará el filtro para que sea posible cambiar la ubicación de sus componentes en tiempo de ejecución cuando se detecte un daño en el sustrato físico de la FPGA sobre el que está ubicado. Al reubicar dicho componente en una región de la FPGA libre de daños conseguiremos que el dispositivo pueda seguir operando tras la aparición y detección del fallo.

En el proyecto se han abordado tres tareas:

1. Por un lado el diseño del conformador trapezoidal clásico siguiendo la descripción proporcionada por su creador. Se ha verificado su funcionamiento mediante simulación funcional y se ha sintetizado para medir los recursos de la FPGA que requiere.
2. A continuación se ha rediseñado el conformador para que sea posible cambiar la ubicación física de sus componentes y que siga funcionando correctamente, dotándole de esta forma de tolerancia a fallos. La ubicación de los componentes es una característica física del diseño, pero que sea posible cambiarla en tiempo de ejecución sin afectar a la funcionalidad del diseño requiere adaptaciones a nivel lógico. Ello nos obligó a rediseñar el conformador: generar un bus común para las comunicaciones entre los módulos, el diseño de un árbitro de bus y los correspondientes esclavos y el rediseño de los componentes iniciales del filtro para adaptarse a su nueva estructura. Se ha verificado su funcionamiento mediante simulación funcional y se ha sintetizado para medir los recursos de la FPGA que requiere.
3. Finalmente, se utilizó la herramienta PlanAhead para crear distintas configuraciones del diseño, cada una de las cuales es una reconfiguración parcial del mismo. Para ello se implementó una parte estática gracias a la cual los dispositivos dinámicos (susceptibles a cambios en la configuración) pudiesen comunicarse entre ellos. Esta parte estática se implementó como una topología de comunicación que utilizaba dos buses, y contiene el árbitro de bus y los esclavos que hacen de interfaces con los dispositivos dinámicos. De nuevo, se ha verificado el funcionamiento de cada configuración mediante simulación, pero en este caso la simulación es post Place&Route para tener que simular cualquier modificación causada por la distinta ubicación y rutado de los módulos dentro de la FPGA.

Palabras clave: Conformador trapezoidal, FPGA, PlanAhead, Reconfiguración dinámica parcial, Tolerancia a fallos, Topología de comunicación.

Abstract

Particle detectors (such as trapezoidal shaper used for this project) are used in a variety of applications (particle accelerators, artificial satellites...). Such systems include various types of digital filters and one of the most widely used, is the trapezoidal shaper. Due to its scope, such systems are subjected to large doses of radiation that can produce damage to the physical substrate on which the circuit is designed and consequently produce malfunction of the circuit. This project is to implement this type of shaper on a Virtex-6 FPGA and provide it with fault tolerance using partial dynamic reconfiguration capability possessed by this family of FPGAs. Thus, it will provide the ability to filter some of its components can be relocated when a fault is detected in the physical substrate of the FPGA, so that the device can continue to operate after the occurrence and failure detection.

So, will redesign the filter to be able to change the location of its components at run time when damage to the physical substrate of the FPGA on which it is located is detected. By relocating the component in a region of the FPGA harmless get the device to continue operating after the occurrence and failure detection.

The project has addressed three tasks:

1. On one hand the trapezoidal shaping classic design following the description provided by its creator. Operation has been verified by functional simulation and synthesized to measure the FPGA resources required.
2. Then the shaper was redesign to provide it with fault tolerance. This forced to redesign of the design: creating a common bus for communication between modules, the design of a bus master and slave and the corresponding initial redesign of filter components to suit its new structure. Operation has been verified by functional simulation and synthesized to measure the FPGA resources required.
3. Finally, the tool to create PlanAhead different design configurations, each of which is a partial reconfiguration of the same is used. To do a static part through which the dynamic devices (susceptible to configuration changes) could communicate with them was implemented. This static part is implemented as a communication topology used two buses, and contains the bus arbiter and slaves that make as an interface for the dynamic devices. Again, it has been verified to work each configuration by simulation, but in this case the simulation is Place & Route port to have to pretend any changes caused by the different location and routing modules within the FPGA.

Keywords: Communication topology, Fault tolerant, FPGA, Partial Dynamic reconfiguration, PlanAhead, Trapezoidal shaper.

Capítulo 1

Introducción

1.1. Conformación de señal en detectores de partículas

Los detectores de partículas son utilizados en multitud de aplicaciones, desde satélites hasta aceleradores de partículas. Estos sistemas constan de un detector, que puede estar fabricado utilizando distintas tecnologías (basados en semiconductor, centelleadores o gas), un preamplificador, un filtro analógico, un conversor analógico digital y por último un tipo especial de filtro digital conocido como conformador. La Figura 1 ilustra el diagrama de bloques de un detector de partículas típico.

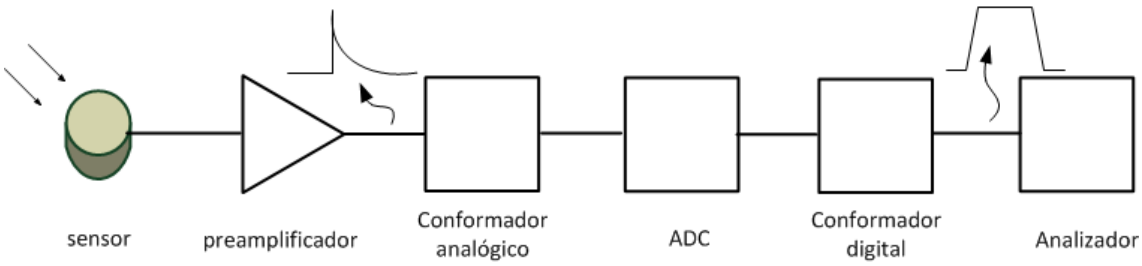

Figura 1 - Diagrama de bloques de una cadena de detección de partículas.

Cuando la partícula incide sobre el detector, este genera a su salida un señal exponencial decreciente (véase la Figura 1) y las sucesivas etapas van transformándola a fin de obtener una señal que permita caracterizar la partícula de forma más fidedigna. Las características de la partícula a detectar se determinarán a partir de unos parámetros que se miden sobre la forma de la señal generada a la salida del conformador. Así, el tipo y energía de las partículas incidentes pueden determinarse a partir del análisis de la altura del pulso (*Pulse Height Analysis*, PHA) o del análisis de la forma del pulso (*Pulse Shape Analysis*, PSA) de la señal generada por el conformador (véase Figura 2). En el primer tipo, la señal previamente conformada es analizada para obtener el valor máximo de la señal. En el segundo, se miden diversos parámetros de la señal a la salida del conformador, tales como la altura de la señal, el tiempo de subida, la duración del pulso o sus componentes en frecuencia para obtener una caracterización del tipo de partícula incidente.

Los componentes iniciales de la cadena (preamplificador, conformador analógico y conversor analógico digital) son circuitos analógicos y a partir de ahí son circuitos digitales. Como se puede apreciar en la figura existen conformadores analógicos y digitales [1]. De los primeros se pueden citar:

1. Diferenciador de CR o filtro de paso alto (CR *differentiator* or *high-pass filter*).
2. Integrador RC o filtro de paso bajo (CRC *integrator* or *low-pass filter*).
3. Conformador CR-RC (CR-RC *shaping*).
4. Gaussiano o conformador CR-(RC)" (*Gaussian or* CR-(RC)" *shaping*).

Entre los conformadores digitales, los más comunes son:

1. Conformador triangular: este tipo de conformador produce a su salida una señal triangular de altura proporcional a la altura del pulso exponencial decreciente que ha recibido a la entrada. La Figura 2 ilustra la salida típica de esta clase de conformadores.

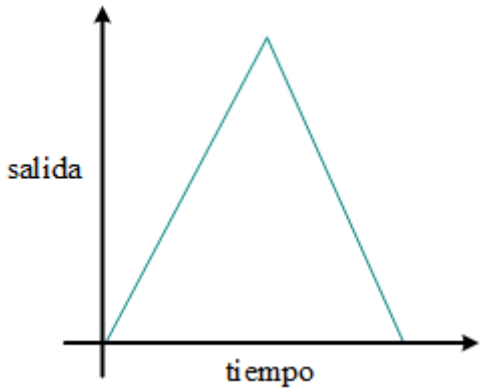

Figura 2 - Salida típica de conformador triangular.

2. Conformador trapezoidal: este tipo de conformador produce a su salida una señal trapezoidal de altura proporcional a la altura del pulso exponencial decreciente que ha recibido a la entrada. La Figura 3 ilustra la salida típica de esta clase de conformadores.

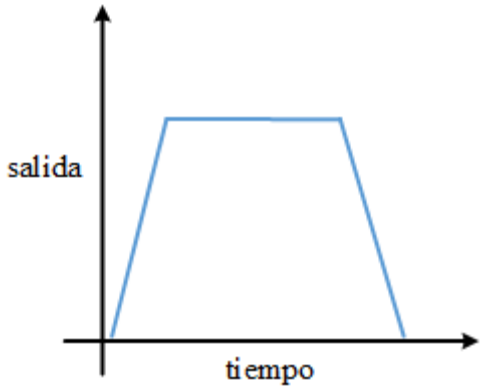

Figura 3 - Salida típica de conformador trapezoidal.

3. Conformador *Cusp-Like*. "En este conformador digital la señal exponencial se transforma en una forma simétrica" (véase Figura 4). La energía del pulso se mide como la altura del pico de la forma resultante [2].

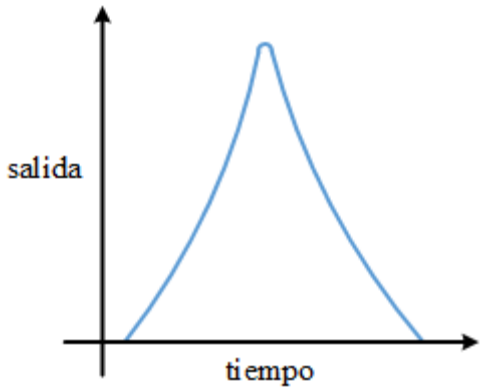

Figura 4 - Salida típica de conformador Cusp-Like.

Cada tipo de conformador tiene un comportamiento distinto frente a los distintos tipos de ruido presente en los sistemas de detección de partículas: ruido blanco serie, ruido blanco paralelo, ruido $\frac{1}{f}$ serie y ruido $\frac{1}{f}$ paralelo. La distribución espectral de estos tipos de ruido es uno de los factores a tener en cuenta en la elección del conformador. El presente trabajo se centra en el diseño de un conformador trapezoidal.

1.2. Tolerancia a fallos

"La computación tolerante a fallos abarca los métodos que permiten a los sistemas digitales realizar su función a pesar de los errores internos en hardware y software.

Los procesos industriales como la generación de energía, la transmisión y distribución, los bancos, las aerolíneas, los ferrocarriles y otras numerosas aplicaciones se basan cada vez más en los computadores para sus operaciones diarias. Los fallos en los computadores están ligados a grandes pérdidas económicas y en algunos casos incluso con peligro para la vida y la integridad física.

Dado que la tecnología y las leyes de la física limitan la fiabilidad de los componentes, un aumento de la fiabilidad sólo se puede lograr mediante la incorporación de elementos redundantes. Los equipos que incorporan redundancia para protección contra fallos son conocidas bajo el nombre genérico de sistemas tolerantes a fallos (*fault-tolerant computers*).

Un sistema tolerante a fallos tiene la propiedad única de que su fiabilidad generalmente es más alta que la fiabilidad de sus partes constituyentes [3]."

Los datos erróneos o la pérdida de datos pueden ser causados por un fallo en un ordenador. Dependiendo de la aplicación, puede ser más peligroso o costoso. Por lo tanto, antes de hablar de la tolerancia del computador, se debe considerar los fallos con respecto a ese equipo.

Los computadores tolerantes a fallos incorporan cierta redundancia. Dependiendo de la aplicación, esta redundancia puede usarse para detectar errores (a fin de evitar una salida

incorrecta) o para seguir trabajando (con el fin de evitar la falta de datos). De hecho, la detección de errores y corrección están estrechamente relacionados.

Hay distintos tipos de redundancia, como por ejemplo: la **redundancia masiva**, (también llamada "**redundancia estática**"), con este tipo de redundancia la tolerancia a fallos se consigue mediante numerosas unidades paralelas que realizan la misma función de forma independiente, el proceso está diseñado de una manera tal que puede tolerar el fallo de cualquiera de estas unidades sin tomar especial acciones para eliminar la unidad defectuosa o incluir una reparación.

Otro tipo de redundancia es la **redundancia de ahorro** (*sparing redundancy*) (también llamada "**redundancia dinámica**"), en la que una unidad hace el trabajo existiendo uno o varios repuestos, que podrían realizar la misma función. Después de un fallo de la unidad de trabajo, ésta se elimina y se utiliza una de repuesto [3]**.**

1.3. Reconfiguración Dinámica Parcial en FPGAs

Desde los años 80 ha habido un gran crecimiento en la implementación de sistemas sobre FPGAs. Estos sistemas dan paso a la idea de la reconfiguración con el fin de cambiar el sistema implementado en el momento deseado. Este concepto permite la reconfiguración y el cambio de funcionalidad de las áreas o módulos seleccionados de la FPGA, cuando esta se encuentra en operación, preservando la implementación de aquellas partes del diseño que permanecen inalterables [4].

Existen varios tipos de reconfiguración [5] [6]:

1. **Estática:** También denominada reconfiguración en tiempo de compilación. Implica parar el sistema y reiniciarlo con una nueva configuración.
2. **Dinámica:** Consiste en reconfigurar el sistema en tiempo de ejecución. Con el objetivo de obtener un equilibrio entre velocidad de ejecución y superficie se definen distintos tipos:
   2.1. **Total.** Tipo de reconfiguración en la que toda la lógica se puede modificar en tiempo de ejecución para hacer cambios en el diseño o bien recuperarse frente a fallos.
   2.2. **Parcial.** Tipo de reconfiguración en la que parte de la lógica puede configurarse en tiempo de ejecución mientras la lógica restante sigue realizando la computación de forma ininterrumpida. Dentro de la reconfiguración dinámica parcial pueden distinguirse varias tipos:
      2.2.1. **Diferencial** (*Small bit manipulation* ó *Difference-Based*). Hacer pequeños cambios en un diseño. Crear un fichero *bitstream* con las diferencias entre el diseño original y el nuevo diseño.
      2.2.2. **Modular** (*Module-Based*). Cuando los cambios son algo mayores se realizan por módulos interrelacionados. Cuando se modifica un módulo para cambiar el comportamiento del sistema se reconfigura únicamente la parte del diseño que contiene a dicho módulo.

En la actualidad está tomando un fuerte impulso la Reconfiguración Dinámica Parcial. La Reconfiguración Dinámica Parcial es la capacidad de modificar una parte de la lógica implementada sobre la FPGA en tiempo de ejecución sin necesidad de volver a configurar totalmente el dispositivo.

1.4. Topologías y protocolos de comunicación en chips

El diseño del filtro consiste en una colección de entidades o módulos lógicos que se comunican mediante conexiones. Cada uno de estos módulos estará ubicado en una región de la FPGA y las comunicaciones se realizarán, en una primera implementación, mediante conexiones punto a punto entre los distintos módulos (véase Figura 5).

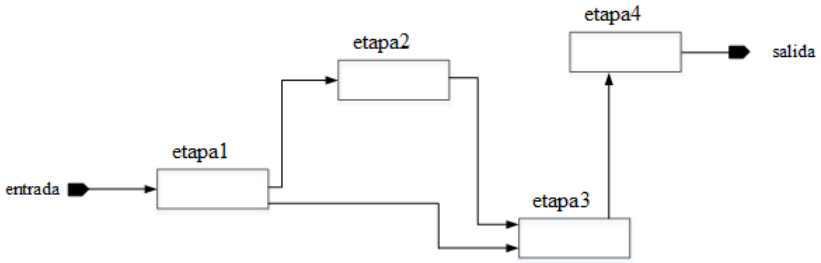

Figura 5 - Conexiones punto a punto.

La tolerancia a fallos en el sustrato va a ser implementada reubicando en una parte distinta de la FPGA y en tiempo de ejecución (reconfiguración dinámica parcial, véase Sección 1.3. Reconfiguración Dinámica Parcial en FPGAs) aquellos módulos del diseño ubicados sobre una región de la FPGA en la que se ha detectado un fallo. Para que el funcionamiento del sistema sea correcto independientemente de la ubicación de los módulos es necesario que la estructura de las conexiones sea tal que lleve la información entre ellos independientemente de su ubicación. Sin embargo, la reconfiguración dinámica no permite modificar la topología de comunicaciones en tiempo de ejecución. Por lo tanto, debemos idear una topología de comunicación que, sin ser modificada en tiempo de ejecución, permita mover un módulo de una región de la FPGA a otra y asegure que los datos correctos llegan al módulo deseado. Obviamente, una topología basada en conexiones punto a punto no satisface estos requisitos: en este tipo de conexiones los cables viajan desde una posición inicial hasta otra final y los módulos por ella conectados no pueden estar más que en las posiciones donde están el comienzo y el final de la conexión. No existe la posibilidad de mover el módulo de un sitio a otro tal.

El problema de utilizar conexiones punto a punto es que en caso de reubicar un módulo, las conexiones de éste no se reubican junto a él. En la Figura 6 se puede observar cómo reubicando la etapa 3 las conexiones que se tenían dejan de estar conectadas al módulo.

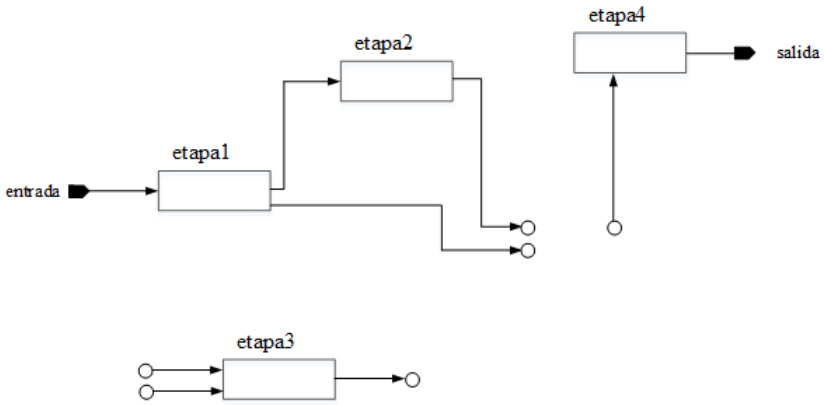

Figura 6 - Conexiones punto a punto reubicadas.

A continuación, se presentan distintas topologías y protocolos de comunicación que sí cumplen con estos requisitos y que hemos valorado para su implementación en este proyecto.

1.4.1. Bus compartido y topología Token Ring

Siendo ésta una solución sencilla con un bus compartido para todas las entidades de la topología en la que una entidad manda un testigo (*token*) a otra, y únicamente cuando se dispone del testigo se puede hacer uso del bus. El testigo pasa de una entidad a la siguiente, y una vez llega a la última entidad se vuelve a la primera, tal y como se muestra en la Figura 7, consiguiendo así una topología en anillo (*ring*).

Este tipo de topología tiene varias ventajas: no requiere de enrutamiento, es decir, no se crea un camino físico entre las entidades que se van a comunicar, sino que se utiliza el bus compartido. Requiere de poca cantidad de cable y es fácil extender su longitud ya que los nodos actúan como repetidores. Sin embargo es un sistema vulnerable a fallos (un fallo físico en un esclavo puede deshabilitar toda la red ya que no podrían pasar mensajes de uno a otro) y los elementos tienen un diseño complejo [7].

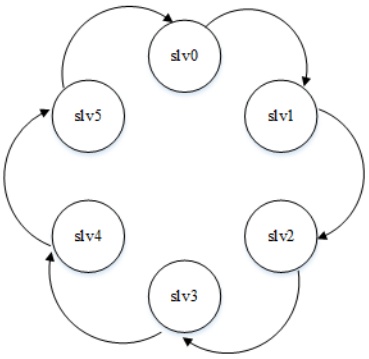

Figura 7 – Topología Token Ring. Las flechas ilustran el paso de testigos entre las distintas entidades de la red.

La comunicación se produce por un bus compartido por todas las entidades. En la Figura 8 se muestra un diagrama de bloques que representa unas entidades conectadas a un bus compartido por todas ellas. Se pueden observar seis entidades que leen y escriben en este bus.

Como se explicó anteriormente, sólo puede hacer uso del bus la entidad que disponga del *token*.

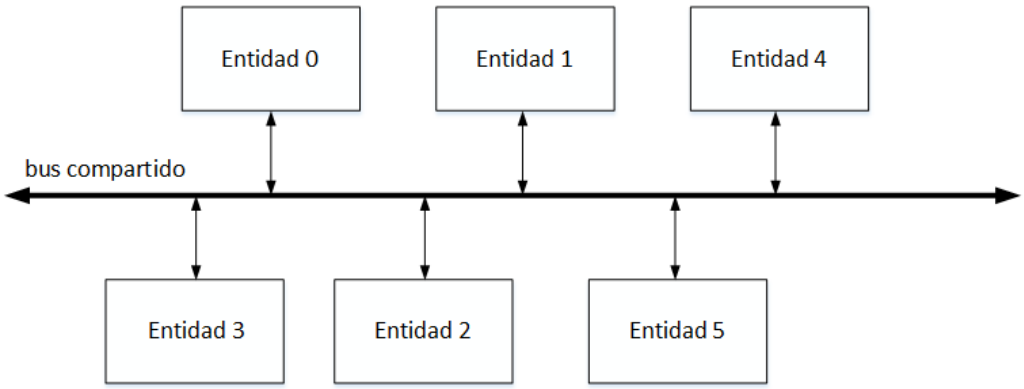

Figura 8 – Diagrama de bloques de un bus compartido.

1.4.2. Bus compartido y protocolo CSMA/CD

Otra opción estudiada fue el protocolo CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*) cuyo funcionamiento consiste en seguir el algoritmo que se muestra en la Figura 9 dotado con un bus compartido por todas las entidades [8].

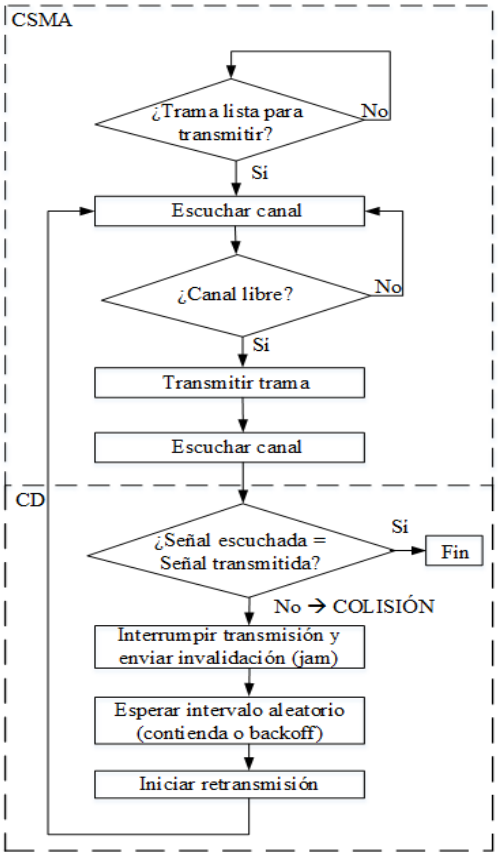

Figura 9 - Algoritmo CSMA/CD.

Esta solución presenta varios problemas para su aplicación en el presente proyecto:

1. Detectar una colisión en el espacio de transmisión ya que no se tiene la certidumbre de qué valor tomaría la señal en el momento de la colisión si sobre el mismo canal hay dos transmisores que intentan transmitir simultáneamente un 1 y un 0 lógicos. Determinar qué señal será observable en el canal no sencillo sin un estudio previo al comportamiento del canal a nivel eléctrico;
2. Nuestra aproximación requiere la implementación *hardware* de todo el algoritmo, lo que por sí solo supondría una carga de trabajo para un proyecto de fin de grado.

En resumen, esta solución no es viable en nuestro caso.

1.4.3. Bus compartido y árbitro con cola de prioridades

En esta solución existe una entidad (árbitro) que toma la decisión sobre qué entidades, o en nuestro caso qué componente, puede enviar datos por el bus y un bus que actúa como canal de comunicaciones compartido entre todos los componentes. El árbitro posee una estructura, llamada cola con prioridades, para guardar las peticiones de uso del bus emitidas por los componentes que desean mandar datos. En esta solución cada entidad tendría una prioridad fija. Así mismo, todas las entidades estarían conectadas al árbitro por conexiones punto a punto y únicamente cuando éste les concediese el bus podrían transmitir datos por el bus.

La principal ventaja de esta solución es su sencillez de diseño: por un lado se necesita una entidad de árbitro que estará conectada a los dispositivos y que asignará con prioridad fija el uso del bus compartido, y por otro lado el diseño de la lógica de petición del bus en cada una de las entidades.

No obstante tiene el inconveniente de ser más vulnerable a fallos en alguno de sus componentes que aquellas soluciones en las que la decisión de asignación del bus está descentralizada (por ejemplo, CSMA/CD). Así, un único daño en el árbitro que cause un mal funcionamiento de la lógica de asignación del bus llevará a un fallo global de todo el sistema puesto que alguna o ninguna entidad recibirá la asignación del bus para poder transmitir.

1.4.4. Bus similar a PLB

Otra aproximación que se estudió fue el uso de un bus similar a un bus PLB (*Processor Local Bus*) en el que se escriben ciertos valores en el bus indicando el estado en el que se encuentra el bus, o bien libre, o bien ocupado (véase la Figura 10) [9].

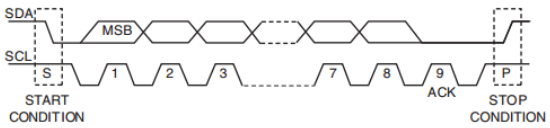

Figura 10 - Transferencia de datos en un bus PLB.

La línea SCL indica el reloj serie (*serial clock*) y la línea SDA los datos serie (*serial data*). Ambas líneas transportan datos bidireccionalmente entre los dispositivos conectados. Los números en

la línea SCL indican el bloque transmitido, los datos se transmiten con el bit más significativo (MSB) primero.

Se observa que cuando la línea SDA cambia de uno a cero mientras la línea SCL se mantiene constante a uno, se inicia la comunicación. Para detectar la condición de parada se observa que la línea SDA cambia de cero a uno mientras la línea SCL se mantiene constante a uno.

Cuando un dispositivo ve que el bus está libre lo solicita, indicando que empieza a transmitir datos (*START CONDITION*), transmite los datos e indica que ha terminado de utilizar el bus (*STOP CONDITION*).

Se pensó en implementar esta solución debido a que la biblioteca que proporciona ISE aporta este bus como un componente (IP). Un problema de ese componente es que es demasiado sofisticado para un diseño sencillo.

En esta idea existe la posibilidad de que dos entidades soliciten el bus a la vez produciendo así una colisión, y ya que no es posible detectar dicha colisión se rechazó esta opción, igual que en el protocolo CSMA/CD.

1.4.5. Network on Chip (NoC)

La mejor aproximación a este problema era el uso de una *Network on Chip* (NoC). Esta arquitectura se basa en conectar los bloques IP (*Intelectual Property*) en una plataforma Soc (*System on Chip*) mediante pequeños *routers* que controlan las comunicaciones entre las distintas IPs [10].

Esta idea vino desde los multiprocesadores de gran envergadura y las redes de computación distribuidas. La naturaleza escalable y modular de las NoCs y su soporte de la comunicación eficiente *on-chip* guían hacia implementaciones basadas en sistemas NoC. Aunque las tecnologías actuales de redes están bien desarrolladas y el soporte a las funciones es excelente, sus configuraciones y su implementación son complicadas, lo cual hace difícil implementarlo como metodología de interconexión en *chips* [11].

El uso de NoCs puede reducir la complejidad del diseño de cables para mejorar la velocidad, la potencia, el ruido…, gracias a su estructura regular. Una NoC ofrece una separación entre la computación y la comunicación, soporta modularidad y reutilización de IPs vía interfaces estándar. Manejan problemas de sincronización, e incluso sirven como plataforma para sistemas de *tests.*

En la Figura 11 se muestra una arquitectura de *Network on Chip* [12].

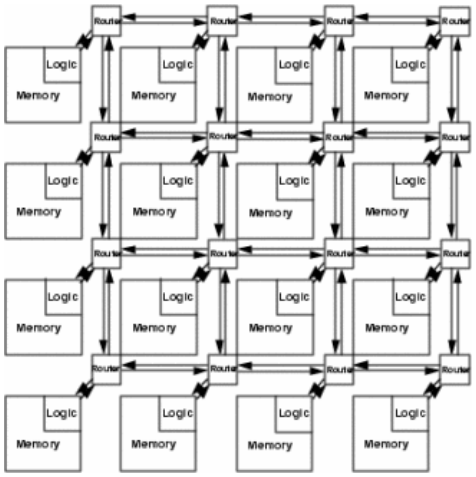

Figura 11 - Arquitectura de Network on Chip.

1.4.6. Conclusiones

Finalmente se decidió utilizar dos buses compartidos y una topología con árbitro con cola de prioridades (Sección 1.4.3. Bus compartido y árbitro con cola de prioridades) con las siguientes variaciones:

1. **Se han implementado dos buses:** uno por el que viaja el dato (bus de datos) y otro por el que viaja el identificador de la entidad receptora (bus de identificación). El bus de datos es utilizado por los módulos conectados al bus para enviar los datos de uno a otro. El bus de identificación es usado para que la entidad emisora indique el destinatario del dato enviado por el bus de datos. Un esquema de dicha topología se muestra en la figura de más adelante.

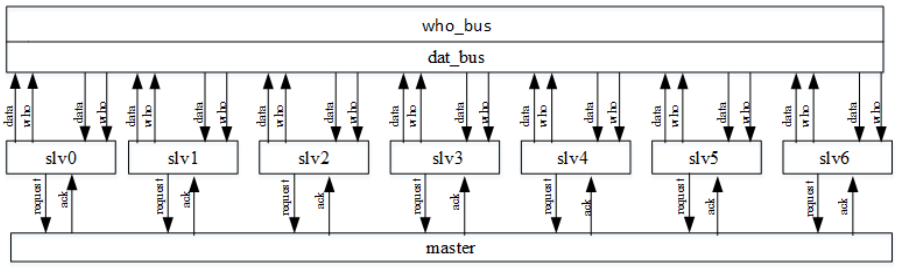

Figura 12 - Diagrama de bloques de la topología implementada.

Por lo tanto cada entidad conectada debe estar configurada con un código único que sirva para identificarla como receptor de los datos presentes en el bus de datos. Además cada entidad debe conocer el código de identificación de la entidad a la que debe enviar los datos.

2. **El árbitro implementa un esquema de asignación de bus basado en *token ring*:** de forma secuencial y cíclica va asignando el uso del bus a los distintos módulos siempre y cuando estos lo hayan solicitado. La concesión del bus a los esclavos por parte del master sigue una topología *token ring* en la que se otorga el bus al primer esclavo, a continuación al segundo, y consecutivamente hasta el último esclavo, y se vuelve conceder al primer esclavo, completando así la topología en anillo.

Durante la evaluación de las distintas topologías se desecharon las restantes opciones, alguna de las cuales en principio parecen más adecuadas al problema bajo estudio (por ejemplo NoC) porque:

1. La carga de trabajo de realizar estas opciones era muy alta para un proyecto con un tiempo limitado en el que sólo trabajó una persona. Algo similar ocurre con las otras opciones: aportarían mayor escalabilidad con el número de módulos conectados o mayor tolerancia a fallos o una solución más elegante a este problema pero suponen una carga de trabajo elevada para una sola persona.
2. El objetivo del presente trabajo es dotar de tolerancia a fallos (usando reconfiguración parcial) al conformador trapezoidal y comprobar el correcto funcionamiento de éste para diferentes configuraciones tal y como se explicará más adelante. Para satisfacer este objetivo es suficiente con implementar la topología de conexión más sencilla que satisfaga los requisitos enunciados al comienzo de la Sección 1.4. Topologías y protocolos de comunicación en chips.

1.5. Xilinx ML-605 Evaluation Board

En este proyecto se utilizó la tarjeta de evaluación Xilinx ML-605 (*Xilinx ML-605 Evaluation Board*) como plataforma de desarrollo. En esta plataforma se incluyen todos los componentes hardware básicos, herramientas de diseño, IPs, y referencias a diseños verificados de sistemas que requieren de alto rendimiento, conectividad serie e interconexión avanzada a la memoria. Los diseños verificados incluidos y los estándar industriales FPGA *Mezzainie Conncectors* (FMC) pueden ser escalados y personalizados con tarjetas auxiliares [13].

Esta plataforma de desarrollo utiliza una FPGA Virtex 6 XC6VLX240T-1FFG1156. Los caracteres a la izquierda del guión (XC6VLX240T) indican el tipo de dispositivo. Los caracteres a derecha indican: el -1 indica el grado de velocidad, los grados de velocidad existentes son -3, -2, -1 y -1L. El grado de velocidad -1L es el más lento y el -3 el grado más veloz. La placa sobre la que se trabajó no está dotada del grado -1L, por lo que en esta placa en concreto, el grado más lento es -1. Por esto, la placa utilizada está dotada del grado de velocidad más lento. “El grado de la velocidad influye en una gran variedad de temporización parámetros en el FPGA, incluyendo tejido (*slice*), *multiplier*/DSP48x, Block RAM, parámetros de E / S y otros recursos [14].”

FF se refiere al tipo de paquete, y en este caso indica que se utiliza la tecnología *flip-chip* [15] que como método de empaque para *chips*, reduce el tamaño del circuito integrado a la mínima expresión, convirtiéndolo en una pequeña pieza de silicio con diminutas conexiones eléctricas.

La letra G indica que los componentes son libres de plomo (*Pb-free*) que cumplen con la directiva RoHS (*Restriction of Hazardous Substances*) de la Unión Europea [16]. Y por último se indica el número de pines (1156).

Esta FPGA *Virtex* 6 consta de 241.152 celdas de lógica, 37.680 *slices* (cada *slice* contiene cuatro LUTs y ocho *flip-flops*), y un máximo de 600 pines de entrada/salida disponibles para el usuario

y 300 pares diferenciales (*differential pairs*) [17]. También consta de un bloque MAC (*Media Access Controller*) de Ethernet 10/100/1000 Mb/s.

Consta de opciones de configuración sensibles, como por ejemplo, soporte *multi-bitstream* dedicado a la lógica reconfigurable, por lo cual soporta la reconfiguración parcial.

A nivel de tarjeta se conecta a una fuente de alimentación universal de 12V, tiene dos cables USB para activar la configuración del dispositivo, tiene conexión a Ethernet, y por último, consta de un adaptador DVI-VGA para mostrar imágenes [18].

Capítulo 2

Conformador trapezoidal

Este capítulo está dedicado a presentar el conformador trapezoidal: el sistema que se usará a lo largo del proyecto y que se irá modificando en los sucesivos capítulos para dotarle de tolerancia a fallos. En particular, este capítulo está dedicado a presentar las ecuaciones que describen su funcionamiento, los parámetros que modifican su comportamiento y su significado, y la descripción y el diseño de los distintos módulos que lo constituyen.

Este diseño ha sido realizado a partir de [19].

2.1. Descripción del conformador trapezoidal

Como ya se mencionó en la Sección 1.1. Conformación de señal en detectores de partículas, en el presente trabajo se aborda el diseño e implementación de un conformador trapezoidal.

El conformador trapezoidal recibe a su entrada las señales digitalizadas, provenientes del ADC (*Analog-to-Digital Converter*), de un pulso exponencial decreciente y lo transforma en un pulso trapezoidal simétrico (véase Figura 3). El algoritmo recursivo que transforma un pulso exponencial digital, v(n), en un pulso trapezoidal simétrico, s(n), viene dado por las siguientes ecuaciones:

$$d^{k,j}(n) = v(n) - v(n-k) - v(n-1) + v(n-k-l) \qquad (\mathbf{1})$$

$$p(n) = p(n-1) + d^{k,l}(n), \quad n \geq 0 \qquad (\mathbf{2})$$

$$r(n) = p(n) + Md^{k,l}(n) \qquad (\mathbf{3})$$

$$s(n) = s(n-1) + r(n), \quad n \geq 0 \qquad (\mathbf{4})$$

donde v(n), p(n) y s(n) son cero para valores de n negativos (n < 0).

El parámetro M únicamente depende de la constante de tiempo, tau, τ del pulso exponencial y el período de muestreo T_{clk} del pulso digital y viene dado por:

$$M = e^{\frac{T_{clk}}{\tau}} - 1 \qquad (\mathbf{5})$$

Para los valores de $\tau / T_{clk} > 5$, la ecuación (5) se puede aproximar a:

$$M \simeq \frac{\tau}{T_{clk}} - 0.5$$

Si definimos la ecuación (6)

$$d^k(n) = v(n) - v(n-k) \qquad (6)$$

entonces la ecuación (1) puede expresarse haciendo uso de la ecuación (6) como:

$$d^{k,l}(n) = d^k(n) - d^k(n-l) \qquad (7)$$

Se observa que la ecuación (7) es la ecuación (1) renombrada para usar la ecuación (6) con dos distintos argumentos: por un lado n, y por el otro lado n-l.

En la ecuación (6) la señal d(n) se obtiene como la resta de la señal de entrada en el instante actual, v(n), menos la señal de entrada recibida hace k muestras, es decir, hace k ciclos de reloj. De manera similar, la ecuación (7) utiliza por un lado esta misma ecuación d(n) con la que se resta v(n) en el instante actual con v(n-k) hace k ciclos. Y por otro lado esta misma ecuación hace l ciclos, d(n-l), que da como resultado la resta de v(n-l) hace l ciclos y v(n-l-k) hace k+l ciclos.

Estas ecuaciones producen a la salida del conformador una señal trapezoidal. La duración del flanco ascendente y descendente de la forma trapezoidal viene dada por el valor mínimo de k y l (min(k,l)) y la duración de la parte plana viene dada por el valor absoluto de la diferencia entre k y l (abs(l - k)). En figura que se muestra más adelante se observa la representación de los parámetros en una salida típica:

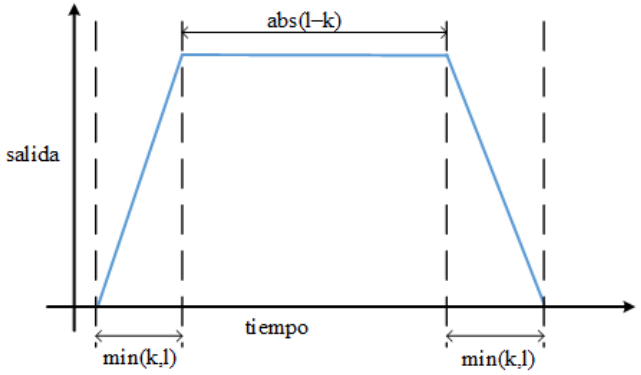

Figura 13 - Representación gráfico parámetros k y l.

En la Figura 14 se observa el diagrama de bloques de la unidad que implementa la ecuación (6). A este bloque se le llama unidad de retardo-resta (*Delay-Subtract unit* - DS). Consta de un elemento de retardo y un restador.

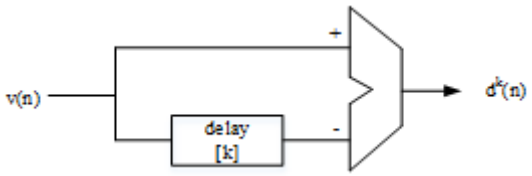

Figura 14 - Diagrama de bloques DS.

Tal y como se ha explicado, la ecuación (1) puede expresarse como la ecuación (7) que hace uso de la ecuación (6) con dos parámetros: por un lado n, para restar la entrada en el instante actual con la entrada hace k ciclos, y por otro lado n-l, para restar la entrada hace l ciclos con la entrada hace k+l ciclos. Teniendo en cuenta que la ecuación (6) se implementa con una entidad DS, la ecuación (7), y por tanto la (1), se implementan como dos bloques DS consecutivos (uno configurado con un retardo de k ciclos, y otro con un retardo de l ciclos) tal y como se muestra en la Figura 15. Al primero de los bloques DS le llamaremos DS1 y al segundo DS2.

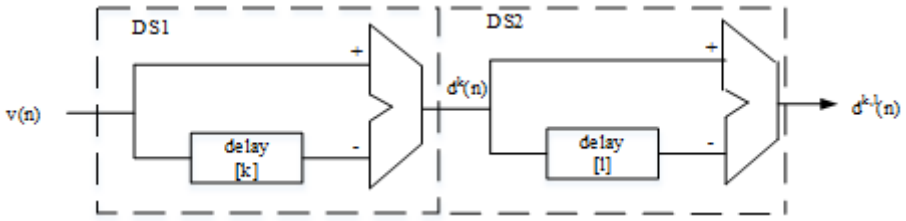

Figura 15 - Diagrama de bloques que implementa la ecuación (1).

En la Figura 16 se muestra el filtro paso alto de deconvolución (*high-pass filter deconvolver* - HPD) que implementa las ecuaciones **(2)** y **(3)**. El parámetro de la multiplicación M viene dado por la ecuación **(5)**. En esta figura se puede observar la presencia de dos parámetros, m_1 y m_2, que debe satisfacer la relación dada por la siguiente ecuación:

$$M = \frac{m_1}{m_2}$$

El orden de los módulos DS1 y DS2 no interfieren en la salida, tampoco lo hace el orden de colocación del módulo HPD, pero en el caso de que este módulo se sitúe antes que los módulos DS1 y/o DS2 se deberá tener cuidado ya que puede causar problemas de desbordamiento. En este diseño se ha elegido colocar primero la entidad DS1, después DS2 y a continuación HPD para evitar estos problemas.

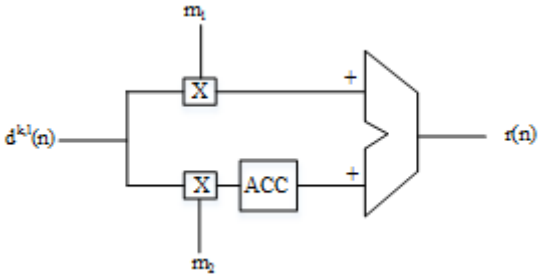

Figura 16 - Diagrama de bloques HPD.

El último bloque del conformador trapezoidal es un acumulador (ACC) que implementa la ecuación (4). Esta ecuación realiza la suma acumulada de la señal r(n) generada por el módulo HPD y la almacena en la variable s(n) que es la salida del conformador. El acumulador debe ser capaz de generar salidas del mayor ancho que pueda producir una entrada. Esta unidad se implementa como un sumador y un registro para acumular la entrada con la salida anterior.

2.2. Diseño del conformador trapezoidal

En esta sección se explican las principales características del diseño del conformador trapezoidal que se ha implementado a partir de las ecuaciones y los diagramas de bloques explicados en la sección de más atrás.

Uno de los fines de esta implementación es conseguir un diseño adaptativo o evolutivo para que se puedan configurar los parámetros de entrada explicados en la sección anterior (k, l, m_1 y m_2). Por esta razón se define el módulo adicional de parámetros, no explicado en la sección anterior. Este módulo recibirá la configuración (conjunto de parámetros k, l, m_1 y m_2) que se va utilizar en el conformador y se encarga de enviar a cada módulo el valor de su parámetro.

Para implementar las ecuaciones del conformador trapezoidal enumeradas en el anterior apartado, se utiliza el esquema de la Figura 17. A continuación se explica con más detalle cada componente de este esquema. El conformador que se detalla a continuación es una variante del diseño simplificado en la sección de más atrás en la que los valores de los parámetros (k, l, m_1 y m_2) se pueden modificar gracias a un módulo de parámetros.

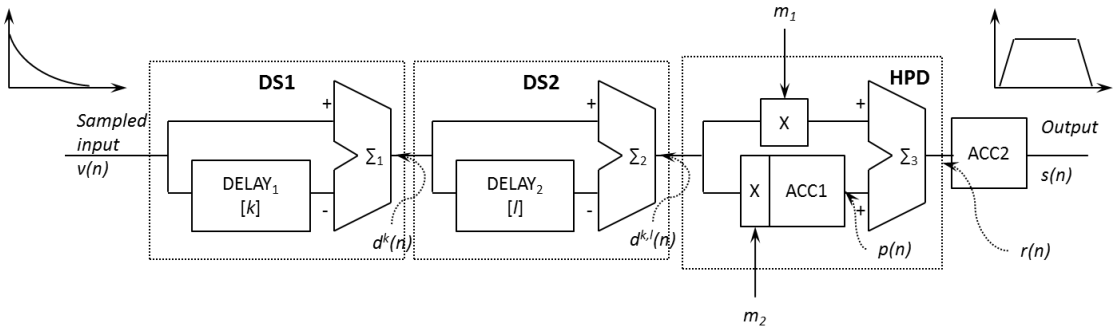

Figura 17 - Diagrama de bloques del conformador trapezoidal.

Cabe destacar que en esta implementación se ha utilizado una única entidad DS para los dispositivos DS1 y DS2 ya que son módulos con la misma lógica, pero con diferentes parámetros. Gracias a que se cuenta con el módulo parámetros no existe ningún problema en realizar un único módulo DS para estas dos entidades.

Por último, se tomaron ciertas decisiones de diseño globales: todos los *resets* utilizados en este proyecto son síncronos ya que no está recomendado utilizarlos de forma asíncrona (a pesar de que esté soportado por dispositivos Xilinx) porque la funcionalidad secuencial en los componentes RAM y en los bloques DSP únicamente pueden tener *resets* síncronos [20]. Otra decisión global es que se utiliza un único dominio de reloj y que la sincronización del reloj es por flanco de subida. Los números están representados en complemento a dos (C2). Por último la señal de *reset* se activa a baja.

2.2.1. Módulo DS

Esta entidad debe implementar la ecuación (6) de la sección 2.1. Descripción del conformador trapezoidal o lo que es igual el bloque DS de la Figura 18. Este módulo se ha implementado como un registro de desplazamiento (como un *delay*) de una anchura de 14 bits. Para que este *delay* sea programable cada salida del registro se conecta a la entrada de un multiplexor, en el que la señal de selección es el parámetro k (para el módulo *ds1*) o l (para el módulo *ds2*). El máximo *delay* es de 64, luego el multiplexor tiene 64 entradas y los parámetros k y l se representan con 6 bits. Y un restador que resta la señal de entrada y la salida del multiplexor (véase Figura 17)

Con esta entidad se da una implementación tanto al dispositivo DS1 como al dispositivo DS2, es decir, una implementación a la ecuación (7) de la sección 2.1. Descripción del conformador trapezoidal, que como ya se explicó podía ser representada como la concatenación de dos módulos DS (DS1 y DS2).

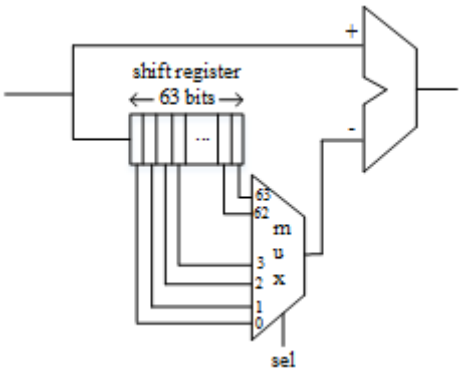

Figura 18 - Implementación del módulo DS.

Los puertos de este módulo se muestran en la siguiente tabla:

Nombre	I/O	Descripción
clk	In	Reloj del sistema
rst_n	In	*Reset* síncrono activo a baja
din	In	Datos de entrada al módulo
delay_sel	In	Configuración de entrada al dispositivo (k para DS1 y l para DS2)
en	In	Señal de *enable* (que viene desde el puerto *valid* del conformador) con el fin de que los registros carguen datos únicamente cuando el dato sea válido
dout	Out	Datos de salida del módulo

Tabla 1 - Puertos de entrada salida del módulo DS.

La entrada de datos se conecta al módulo DS1 (al cual se le proporciona el parámetro k, que selecciona el retardo del registro de desplazamiento, por la entrada *delay_sel*) y su salida se conecta con la entrada del módulo DS2 (al cual se le proporciona el parámetro l, que selecciona el retardo del registro de desplazamiento, por la entrada *delay_sel*).

La definición de los genéricos de este módulo se muestra en la siguiente tabla:

Nombre	Valor	Descripción
g_indata_width	14	Ancho de datos de entrada
g_outdata_width	37	Ancho de datos de salida

Tabla 2 - Genéricos del módulo DS.

2.2.2. Módulo HPD

Este módulo se ha diseñado para implementar las ecuaciones **(2)** y **(3)** de la sección 2.1. La diferencia de la implementación de esta entidad con el diagrama de bloques explicado en la

sección 2.1. Descripción del conformador trapezoidal es que se le proporcionan los parámetros m1 y m2 desde una entidad de parámetros que se explicará más adelante.

La entrada a este módulo se divide en dos caminos: en uno se multiplica por el parámetro m1 y se extiende a la anchura deseada; en el otro camino se multiplica por el parámetro m2, se extiende a la anchura deseada y entra en un acumulador. Finalmente estos dos caminos se suman obteniéndose así la salida. Se puede observar esta implementación en la Figura 19.

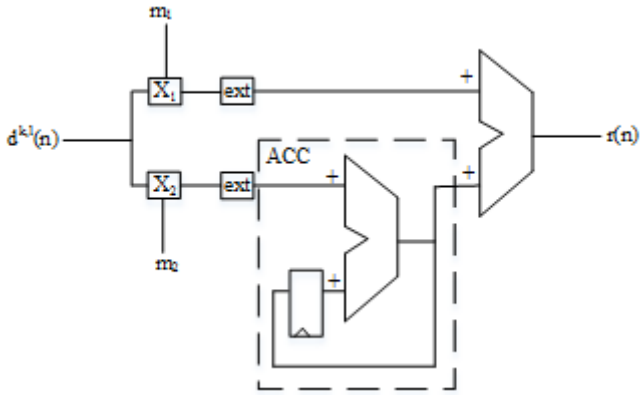

Figura 19 - Implementación del módulo HPD.

Los puertos de este módulo se muestran en la siguiente tabla:

Nombre	I/O	Descripción
clk	in	Reloj del sistema
rst_n	in	*Reset* síncrono activo a baja
din	in	Datos de entrada al módulo
m1	in	Parámetro m1 de configuración
m2	in	Parámetro m2 de configuración
en	in	Señal de *enable* (que viene desde el puerto *valid* del conformador) con el fin de que los registros carguen datos únicamente cuando el dato sea válido
dout	out	Datos de salida del módulo

Tabla 3 - Puertos de entrada salida del módulo HPD.

La entrada a este módulo es proporcionada por la salida del módulo DS2 que se explicó anteriormente.

Los genéricos de este módulo son los siguientes:

Nombre	Valor	Descripción
g_indata_width	14	Ancho de datos de entrada
g_outdata_width	37	Ancho de datos de salida

Tabla 4 – Genéricos del módulo HPD.

2.2.3. Módulo ACC

Este módulo es el que a partir de la salida del módulo HPD genera la salida del conformador trapezoidal conforme a la ecuación (4) de la sección 2.1. Descripción del conformador trapezoidal. La implementación de este módulo coincide con el diagrama de bloques que se indicó en la sección 2.1. Descripción del conformador trapezoidal. La figura que se muestra más adelante representa la implementación del módulo ACC.

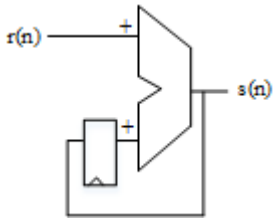

Figura 20 - Implementación del módulo ACC.

Los puertos de este módulo se muestran en la Tabla 5. Se puede observar que no hay ningún puerto de entrada de configuración, esto es porque este módulo no consta de ningún parámetro de entrada de configuración.

Nombre	I/O	Descripción
clk	in	Reloj del sistema
rst_n	in	*Reset* síncrono activo a baja
acc_in	in	Datos de entrada al módulo
en	in	Señal de *enable* (que viene desde el puerto *valid* del conformador) con el fin de que los registros carguen datos únicamente cuando el dato sea válido
acc	out	Datos de salida del módulo ACC y del conformador trapezoidal

Tabla 5 - Puertos del módulo ACC.

Este módulo únicamente necesita conocer la anchura de los datos de salida, ya que la entrada (que viene de la salida del módulo HPD) es de la misma anchura que los datos de salida (véase Tabla 6).

Nombre	Valor	Descripción
g_outdata_width	37	Ancho de datos de salida

Tabla 6 - Genéricos del módulo ACC.

2.2.4. Parámetros

Como ya mencionamos con anterioridad, uno de los objetivos de este diseño es conseguir un diseño adaptativo en el que se puedan modificar los parámetros del conformador (k, l, m_1 y m_2). Así, el conformador tiene como una entrada un vector de configuración en la que están concatenados los diferentes parámetros. Este módulo se encarga de extraer los parámetros de ese vector y entregarle a cada entidad el parámetro adecuado. En un futuro estos parámetros serán proporcionados por un módulo que se encargará de buscar los mejores parámetros cuando se produzca una degeneración en el sensor (véase Figura 1) y de esta forma seleccionar los parámetros que regeneran al sistema.

En este vector de configuración se encuentra en los bits más significativos el parámetro k, a continuación se encuentra el parámetro l, el parámetro m_1 y por último, en los bits menos significativos, el parámetro m_2. Este módulo consiste en dividir los distintos parámetros (k, l, m_1 y m_2) del vector de configuración que se proporciona, teniendo en cuenta que los parámetros están dispuestos como se explicó antes.

Los puertos de este módulo se pueden ver en la siguiente tabla:

Nombre	I/O	Descripción
chrom	In	vector de configuración que consta de los parámetros k, l, m_1 y m_2
k	out	Parámetro k de salida hacia el dispositivo DS1
l	out	Parámetro l de salida hacia el dispositivo DS2
m1	out	Parámetro m1 de salida hacia el dispositivo HPD
m2	out	Parámetro m2 de salida hacia el dispositivo HPD

Tabla 7 - Puertos del módulo Parameters.

Por último, es necesario indicar a este módulo de la anchura del cromosoma y de los parámetros (k y l tienen la misma anchura, y m1 y m2 también). Estas anchuras se definen con los siguientes genéricos:

Nombre	Valor	Descripción
g_kl_width	6	Ancho de los parámetros k y l
g_m_width	14	Ancho de los parámetros m_1 y m_2
g_chrom_len	40	Ancho del cromosoma

Tabla 8 - Genéricos del módulo Parameters.

2.2.5. *Top level* - trapezoidal

Finalmente la entidad trapezoidal agrupa los módulos explicados en los apartados anteriores hasta formar el *top level*, es decir, hasta formar el conformador trapezoidal. En la Figura 21 se observa la implementación utilizada. Este conformador se ciñe a las ecuaciones de la sección 2.1 y a las particularidades que se ha indicado en los apartados anteriores (capacidad de configurar el conformador utilizando diferentes parámetros).

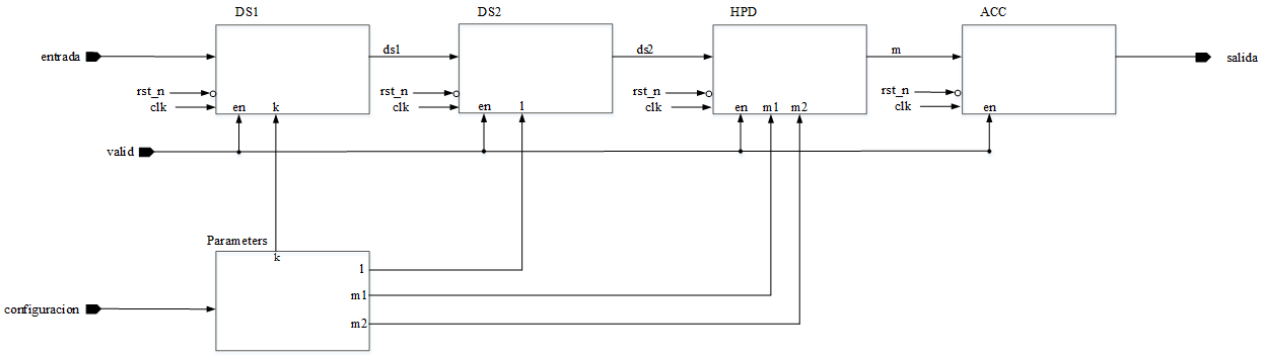

Figura 21 - Implementación conformador trapezoidal.

Los puertos de este módulo se muestran en la siguiente tabla:

Nombre	I/O	Descripción
clk	in	Reloj del sistema
rst_n	in	*Reset* síncrono activo a baja
entrada	in	Datos de entrada al conformador trapezoidal
configuracion	in	Configuración del conformador trapezoidal (k, l, m1 y m2)
valid	in	Puerto que indica que un dato es válido y que permite a los registros del sistema cargar el valor de entrada
salida	out	Datos de salida del conformador trapezoidal

Tabla 9 - Puertos del módulo trapezoidal (top-level).

Los genéricos de esta entidad se muestran a continuación:

Nombre	Valor	Descripción
g_indata_width	14	Ancho de los datos de entrada
g_chrom_len	40	Ancho de los parámetros de configuración
g_outdata_width	37	Ancho de los datos de salida

Tabla 10 - Genéricos del módulo trapezoidal (top-level).

2.3. Simulación

La simulación funcional, también conocida como verificación, del conformador trapezoidal se ha realizado utilizando la herramienta *QuestaSim* 10.0d de *MentorGraphics*. Además, para verificar el funcionamiento del conformador trapezoidal se ha realizado el diseño del banco de prueba (*testbench*) que se ilustra en la Figura 22. El banco de pruebas consta del conformador trapezoidal más dos nuevos módulos (*read_file* y *write_file*). La misión del módulo *read_file* es inyectar al conformador una señal exponencial decreciente que se encuentra en el fichero *stimuli*. El módulo *write_file* se encarga de recoger la señal de salida generada por el conformador y escribirla en el fichero *response*.

Además se le añaden un módulo *p_reset* y un módulo *p_clk* con el fin de generar la señal de *reset* y la señal del reloj (clk). También se añade un módulo *p_config* que proporciona la configuración al conformador trapezoidal. Por último está el módulo *p_wr_en* que genera la señal que indica cuándo es válido cada dato.

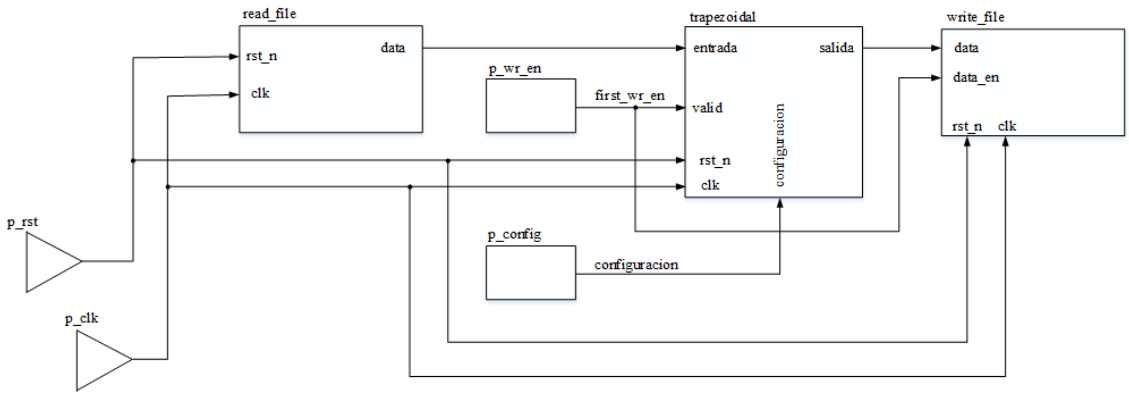

Figura 22 - Implementación tb_trapezoidal.

El módulo de lectura de fichero (*read_file*) funciona a una frecuencia ocho veces menor que el resto del sistema gracias a dos divisores de frecuencia (véase Figura 23). Trabaja a frecuencias más bajas ya que se debe dar tiempo tras cada lectura a que se procesen los datos en el sistema. Este módulo, a parte de los dos divisores de frecuencia (*p_clock_divider* y *p_clock_enable*), consta de un proceso de lectura (*p_read*) con el que se leen los datos del

fichero y un proceso que genera la señal *wr_en* de capacitación de escritura generada cada flanco de subida del reloj con la frecuencia dividida por ocho.

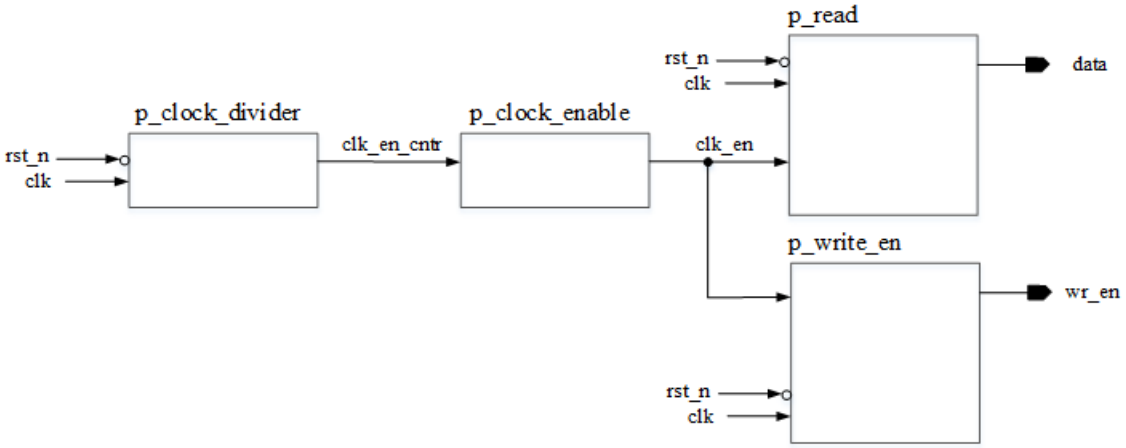

Figura 23 - Implementación módulo read_file.

Con el fin de obtener una representación gráfica de las entradas y de las salidas del conformador trapezoidal se hace uso de un script en *Matlab* (*print_stimuli.m*) que dibuja las entradas y las salidas (véanse la Figura 24 y la Figura 25). Se puede observar que la entrada es una función exponencial y la salida es un trapezoide (tal y como se esperaba). Como era de esperar el valor máximo de la salida (cercano a 10^{15}) está amplificado frente al de entrada (poco superior a 8000). En esta simulación se utilizaron los valores 63 para k, 8 para l, 19 para m1 y 2 para m2.

El flujo de datos representado por la siguiente ecuación:

$$v(n) = Ae^{-\frac{t}{\tau}}$$

Tomando los siguientes valores: A=20 y τ=200μs. Además estos valores se convierten a un vector de 14 bits en complemento a 2 (C2) y se escala la señal para utilizar todos los bits de éste vector. Estos datos de entrada se observan en la Figura 24.

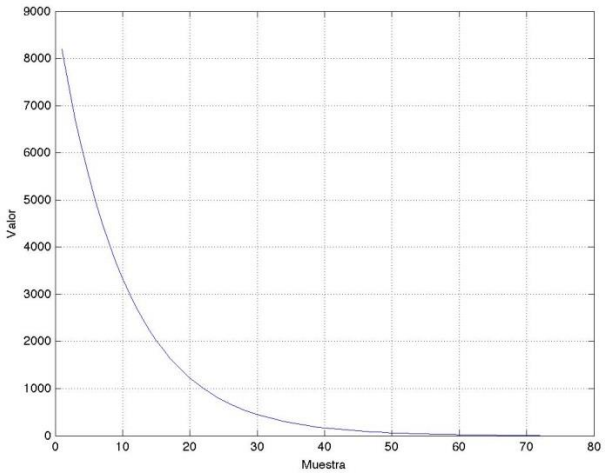

Figura 24 - Entrada conformador

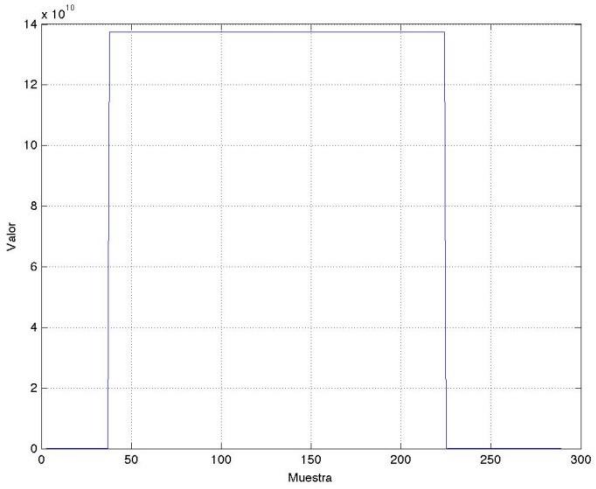

Figura 25 - Salida conformador

2.4. Síntesis

La siguiente etapa en el flujo de diseño del trapezoidal es realizar la síntesis del diseño. Para ello se creó el script que se comenta a continuación. Para realizar la síntesis se ha utilizado la herramienta Xilinx-ISE 14.1 (*Integrated Software Environment*).

2.4.1. Scripts de síntesis

Para sintetizar el conformador trapezoidal se creó el ejecutable *synth_trap*. Para este proyecto se creó un directorio T: para evitar problemas con los *path* de trabajo en el que se guardaron todos los archivos relacionados con el proyecto, en este ejecutable lo primero que se hace es ir al directorio en el que se encuentra la implementación del proyecto del conformador trapezoidal (*T:/trapezoidal_noc/impl/trapezoidal*), lo siguiente que se hace es que si no existe

el directorio *xst*, se crea y se crea el directorio *projnav.tmp* en este directorio. A continuación se ejecuta el siguiente script:

```
xst -intstyle ise -filter
"T:/trapezoidal_noc/tools/ise/trapezoidal/iseconfig/filter.filter" -ifn
"T:/trapezoidal_noc/impl/trapezoidal/trapezoidal.xst" -ofn
"T:/trapezoidal_noc/impl/trapezoidal/trapezoidal.syr"
```

La opción *-intstyle* sirve para indicar el formato de los informes generados por la herramienta de síntesis. En este caso *-inststyle ise* indica que el programa está siendo ejecutado como parte de un entorno de diseño integrado [21]. La opción *-filter* permite poner un archivo *filter.filter* en el que se indica qué tipos de mensaje de salida se desean filtrar. En este caso se ha dejado vacío dicho archivo de forma que la salida incluye todos los mensajes (*info*, *warnings* y *errors*) generados por la herramienta. Por último, con la opción *-ifn* se indica el archivo de entrada *.xst* que contiene las opciones, archivos fuentes y valores de genéricos necesarios para realizar la síntesis del conformador. A continuación se muestra el contenido de este archivo para después comentar cada una de sus líneas [22] (Apéndice A):

1. **`set -tmpdir "T:\trapezoidal_noc\impl\trapezoidal_noc\xst\projnav.tmp"`**
   Se crea el directorio donde se guardarán los archivos intermedios.

2. **`set -xsthdpdir "xst"`**
   Esta línea indica que se ha creado un directorio *xst* donde se guardarán los archivos de caché del *xst*.

3. **`Run`**
   Se lanza la síntesis.

4. **`-ifn trapezoidal.prj`**
   Se indica que la lista de fuentes que integran el proyecto se encuentran en el archive *trapezoidal.prj* (Apéndice B).

5. **`-ofn trapezoidal`**
   Indica que se use *trapezoidal* como nombre base para los archivos generados.

6. **`-ofmt NGC`**
   Indica que el formato de salida será *ngc* (archivo de *netlist* con información de restricciones).

7. **`-p xc6vlx240t-1-ff1156`**
   Se indica la placa utilizada.

8. **`-top trapezoidal`**
   Se establece el nombre de la entidad top.

9. **`-opt_mode Speed`**
   Objetivo de optimización (*Optimization Goal*) define la estrategia de optimización de la síntesis. En este caso se utiliza el valor *speed*, la prioridad de este valor es reducir el número de niveles de lógica e incrementar la frecuencia.

10. **`-opt_level 1`**

Esfuerzo de optimización (*Optimization Goal*) define el nivel de esfuerzo de optimización de la síntesis. El valor 1 indica un nivel normal de optimización. Se utiliza para conseguir un procesamiento muy rápido, especialmente para diseños con jerarquía.

11. -power NO
Reducción de potencia (*Power Reduction*) optimiza el diseño para consumir lo mínimo posible. En este caso no se ha utilizado esta opción.

12. -iuc NO
En este caso no se han utilizado ficheros de restricciones de síntesis utilizados.

13. -keep_hierarchy Yes
Si la jerarquía se mantiene durante la síntesis (como es en este caso), se preserva también la jerarquía en la implementación. Permite crear una netlist de simulación con la jerarquía deseada.

14. -netlist_hierarchy As_Optimized
Se utiliza para controlar la forma del archivo *ngc* generado. El valor *as_optimized* genera el archivo *ngc* del diseño optimizado manteniendo la jerarquía

15. -rtlview Yes
Se activa la opción para crear una vista del diseño RTL.

16. -glob_opt AllClockNets
Dependiendo de las metas de optimización globales (*Global Optimization Goal*) se pueden optimizar diferentes regiones del diseño. Con la opción A*llClockNets* de optimiza el período del diseño.

17. -read_cores YES
Utilizar la funcionalidad de leer los *cores* (como en este caso) habilita la capacidad de leer los archivos *edif* o *ngc* para estimar los tiempos de retardo.

18. -sd {"T:/trapezoidal_noc/impl/ip"}
Con esta línea se indica el path en el que se encuentras los cores *edif* o *ngc*.

19. -write_timing_constraints NO
Se deshabilida la escritura de restricciones de tiempo.

20. -cross_clock_analysis NO
Esta línea indica que no se permite el análisis de dominios inter-reloj mientras se optimizan los retardos.

21. -hierarchy_separator /
Se indica que el carácter separador de las jerarquías es /.

22. -bus_delimiter <>
Indica que el delimitador de bus se realiza mediante los caracteres <>.

23. -case Maintain
Indica cómo se escribirán los nombres de las instancias y las redes en las *netlist* finales. Con la opción *maintain* se mantienen los nombres.

24. -slice_utilization_ratio 100
Define el tamaño del área en números de slices, es decir, el tamaño del área es de 100 slices.

25. -bram_utilization_ratio 100
Indica el número máximo de bloques BRAM (100) que se pueden utilizar.

26. -dsp_utilization_ratio 100
Define el número máximo de *DSP* que se pueden utilizar, en este caso 100.

27. -lc Auto
Define la combinación globalmente de *LUTs* como automática.

28. -uc T:\trapezoidal_noc/impl/trapezoidal/trapezoidal.xcf
Indica el archive de restricciones de síntesis utilizado (el período del reloj del sistema es de 20 ns).

29. -reduce_control_sets Auto
Reducir el conjunto de control permite reducir el número de conjuntos de control, disminuyendo así el área ocupada. Esta opción toma el valor autómatico.

30. -fsm_extract YES -fsm_encoding Auto
La extracción automática de máquinas fitias de estados (FSM) habilita, en este caso, la extracción de máquinas de estados finitas y su respectiva optimización. La opción *fsm_encoding* indica la técnica de codificación para los estados de la FSM. En este caso, se pone como automático.

31. -safe_implementation No
Se deshabilita la opción para implementar máquinas de estado seguras. Para conseguir esta seguridad se añade lógica adicional.

32. -fsm_style LUT
Cambiar la forma de implementación de las máquinas de estados finitos puede hacer a éstas más rápidas (utilizando por ejemplo bloques RAM). En este diseño se eligió utilizar *LUTs* ya que la velocidad no era lo que se perseguía.

33. -ram_extract Yes
Se habilita la macro de inferencia RAM.

34. -ram_style Auto
Controla la forma en que el generador de macros implementa las macros RAM. En este caso se ha dejado con el valor automático.

35. -rom_extract Yes
Se habilita la macro de inferencia ROM.

36. -shreg_extract YES
Se habilita la macro de inferencia de los registros de desplazamiento (*shift register*).

37. -rom_style Auto
Controla la forma en que el generador de macros implementa las macros RAM. En este caso se ha dejado con el valor automático.

38. -auto_bram_packing NO
Se deshabilita la opción que permite empaquetar dos pequeños módulos BRAM en un único módulo BRAM de doble puerto.

39. -resource_sharing YES
Se habilita la opción de compartir operaciones aritméticas.

40. -async_to_sync NO
Se deshabilita la opción que permite pasar las señales asíncronas de *set* o *reset* a señales síncronas.

41. -shreg_min_size 2
Se indica que el tamaó mínimo de los registros de desplazamiento es dos.

42. -use_dsp48 Auto
La opción de incluir bloques *DSP48* se deja en modo automático.

43. -iobuf YES
Se permiten *buffers* de entrada/salida en el diseño.

44. -max_fanout 100
Se limita el máximo fanout a 100.

45. -bufg 32
Se permiten un máximo de 32 buffers en la señal de reloj (valor máximo permitido por la placa).Esta opción sirve para evitar la degradación de la señal de reloj en aquellas entidades lejanas al cristal de cuarzo que la genera.

46. -register_duplication YES
Se habilita la opción de duplicados de registros.

47. -register_balancing No
Se deshabilita la opción de balancear los registros. La principal meta de esta opción es repartir los registros y los *latches* por la placa para incrementar la frecuencia de reloj.

48. -optimize_primitives NO
No se permite la optimización de las primitivas de las librerías utilizadas.

49. -use_clock_enable Auto
La opción de activar o desactivar la capacitación de reloj en los registros (*flip-flip*) se deja con el valor por defecto (automático).

50. -use_sync_set Auto
La opción se utilizar los *sets* síncronos en los registros se deja con el valor por defecto (automático).

51. -use_sync_reset Auto
La opción se utilizar los *sets* síncronos en los registros se deja con el valor por defecto (automático).

52. -iob Auto

Esta opción empaqueta los registros en entradas/salidas para mejorar el retardo de camino entrada/salida.

53. -equivalent_register_removal YES
Se active la opción de borrar los registros que sean equivalentes en nivel RTL.

54. -slice_utilization_ratio_maxmargin
Define el número máximo de *slices* que se pueden utilizar, en este caso se deja la opción por defecto (100).

Por último se indica el archivo de salida *.syr* a generar en el que se volcarán los resultados de la síntesis con la opción *-ofn*.

2.4.2. Resultados de implementación

En esta sección se va a interpretar los datos devueltos por la síntesis en el archivo *trapezoidal.syr* generado al sintetizar. Lo primero que destaca de este archivo es que se utilizan 4810 registros, y una vez optimizada la lógica por la herramienta de Xilinx se utilizan 1866 registros.

También cabe destacar que se utilizan 93 buffers de entrada/salida (56 de entrada y 37 de salida). El período mínimo de reloj de este diseño es de 12.774ns, es decir, una frecuencia máxima de 78.284MHz, lo cual es una frecuencia de trabajo suficiente ya que los conformadores en el espacio están trabajando a 50MHz [23]. El camino crítico (12.774ns) se observa en la siguiente figura marcada en rojo:

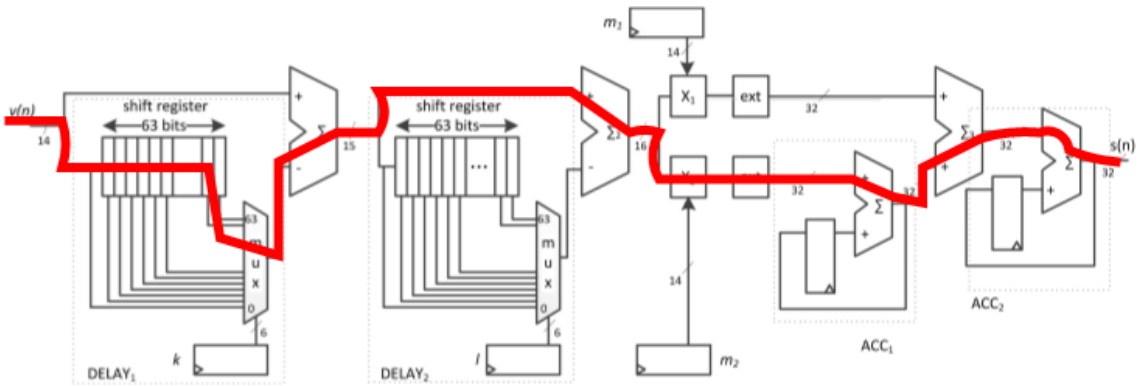

Figura 26 - Camino crítico Trapezoidal.

Este camino crítico se observa en el archivo *.syr* generado por la síntesis, la parte en la que se muestra se cita a continuación. En este archivo se observa que el camino crítico pasa a través del registro de desplazamiento de la entidad DS1 (*i_delay_*ds), que pasa por la entrada inferior al sumador/restador (*i_sub_sd:*sustraendo). Entra en la entidad DS2 y llega hasta el minuendo del sumador/restador (*i_sub_ds:*minuendo). En la entidad HPD pasa por el camino inferior (*i_hpd_*imult2). Y por último pasa por el primer puerto del acumulador de la etapa ACC (*i_add:i_adder_acc:*add). Cabe destacar que se observa que el 66.2% del tiempo se ejecuta lógica, mientras que el 37.1% restante es por el retardo de los cables.

```
Data Path: i_ds1/i_delay_ds/prog_del[26].i_del_reg/q_0 (FF) to
i_acc/i_reg_acc/q_36 (FF)
                              Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     end scope: 'i_ds1/i_delay_ds/prog_del[26].i_del_reg:q<0>'
     end scope: 'i_ds1/i_delay_ds:dout<0>'
     begin scope: 'i_ds1/i_sub_ds:sustraendo<0>'
     end scope: 'i_ds1/i_sub_ds:resta<12>'
     end scope: 'i_ds1:dout<12>'
     begin scope: 'i_ds2:din<12>'
     begin scope: 'i_ds2/i_sub_ds:minuendo<12>'
     end scope: 'i_ds2/i_sub_ds:resta<13>'
     end scope: 'i_ds2:dout<13>'
     begin scope: 'i_hpd:din<13>'
     begin scope: 'i_hpd/i_mult2:op1<13>'
     end scope: 'i_hpd/i_mult2:mult<0>'
     begin scope: 'i_hpd/i_extender_m2:unextended<0>'
     end scope: 'i_hpd/i_extender_m2:extended<0>'
     begin scope: 'i_hpd/i_acc1:acc_in<0>'
     begin scope: 'i_hpd/i_acc1/i_adder_acc:op1<0>'
     end scope: 'i_hpd/i_acc1/i_adder_acc:add<34>'
     end scope: 'i_hpd/i_acc1:acc<34>'
     begin scope: 'i_hpd/i_adder:op2<34>'
     end scope: 'i_hpd/i_adder:add<35>'
     end scope: 'i_hpd:dout<35>'
     begin scope: 'i_acc:acc_in<35>'
     end scope: 'i_acc/i_adder_acc:add<36>'
     begin scope: 'i_acc/i_reg_acc:d<36>'
    ----------------------------------------
    Total                     12.774ns (8.030ns logic, 4.744ns route)
                                       (62.9% logic, 37.1% route)
```

Capítulo 3

Conformador trapezoidal tolerante a fallos

En esta sección se va a tratar de dotar de tolerancia a fallos al diseño del conformador trapezoidal explicado en el Capítulo 2. Esta idea surgió ya que este tipo de dispositivos suelen ir en aparatos que reciben una gran cantidad de radiación, en satélites por ejemplo. Esta radiación hace que las partes de estos aparatos sean vulnerables a ser dañados físicamente y dejen de funcionar correctamente. En este proyecto se aporta una solución a este tipo de problemas tratando de reubicar la entidad que se encuentre en una zona de la placa dañada físicamente a otra parte que se encuentre en buen estado.

Particularmente, para el caso del conformador trapezoidal, que tiene cuatro entidades (DS1, DS2, HPD y ACC) como se explicó en el Capítulo 2 - Conformador trapezoidal, se le va a dotar de una entidad que permita la comunicación, gracias a la cual se podrán reubicar las distintas etapas en la placa.

En la Figura 27 se muestra una configuración de este diseño simplificado en el que toda la placa funciona correctamente. Se observa una entidad de comunicación, que es con la que se consigue que las etapas del conformador se comuniquen entre sí.

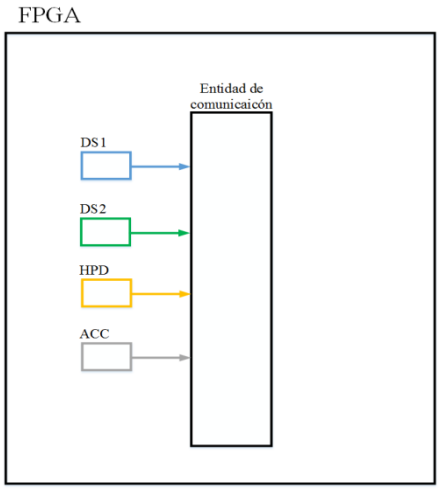

Figura 27 – Diagrama de bloques configuración sin fallos.

Cuando se produzca un fallo en una zona de la placa en la que esté situada alguna de las etapas del conformador trapezoidal, ésta etapa de reubicará a otra zona de la placa y se comunicará con el resto de etapas gracias a la entidad de comunicación. En la Figura 28 se muestra una configuración en la que se ha detectado un fallo en la zona de la placa donde se encontraba la etapa DS2, y esta etapa se reubica a otra zona de la placa.

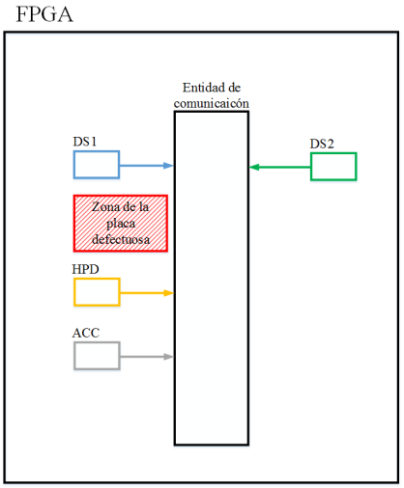

Figura 28 – Diagrama de bloques de configuración con fallos.

3.1. Diseño del conformador trapezoidal tolerante a fallos

En esta sección se va a explicar el diseño de la entidad de comunicación explicada en la sección anterior y cómo se comunican las etapas del conformador haciendo uso de esta entidad.

Esta entidad de comunicación se basa en la topología elegida en la sección 1.4.6. Conclusiones, consta de dos buses, uno para transferir los datos y otro para indicar a qué entidad va dirigido, las entidades que se conectarán a los buses lo harán mediante unas interfaces que tendrán un identificador propio y un identificador de la etapa a la que enviarán los datos. También consta de una entidad que arbitra el uso de dichos buses con el fin de evitar colisiones. En esta entidad se otorga un *token* y únicamente puede hacer uso del bus la entidad que tiene dicho *token*, entregándolo siguiendo una topología en anillo. Se empieza por el identificador cero.

Para conseguir que los componentes que se encuentren en zonas defectuosas de la placa se puedan conectar a la entidad de comunicación, será necesario incluir un número de interfaces para la conexión mayor que el número de etapas del conformador. De esta forma siempre habrá alguna interfaz libre para realizar la conexión de una etapa reubicada. Por esto, utilizamos una redundancia estática en la que se multiplican las entidades de conexión al bus. Es estática ya que es antes de ejecutar el sistema cuando se indica que se implementen más entidades de las necesarias.

Otro cambio necesario a realizar es la configuración de los parámetros del conformador. En el Capítulo 2 - Conformador trapezoidal se explicó que estos parámetros vienen dados por un módulo concreto que conecta mediante un vector los parámetros con sus etapas correspondientes. Debido a la posible reubicación de las etapas en la placa, esta opción es inviable, ya que el bus que conectaba el módulo que concede los parámetros no viaja junto a la etapa. La solución que se encontró a este problema fue incluir este módulo de configuración en la entidad de comunicación, siendo ésta la que distribuya la configuración correspondiente a cada etapa gracias al identificador de las mismas utilizando un multiplexor que se explicará más adelante.

Por último cabe destacar que fue necesario incrementar la funcionalidad de las distintas etapas, envolviéndolas así en unos *wrappers* (*ds1_wrapper*, *ds2_wrapper* y *hpd_wrapper*, *acc_wrapper*) siendo estas entidades las que se conectarán a las interfaces de comunicación del bus. Se han añadido dos funcionalidades a las etapas: añadir un registro para segmentar el comportamiento y añadir un módulo que generará una señal que viajará junto al dato indicando que éste es válido.

3.1.1. Topología de comunicación

La estructura de comunicación que se ha implementado es la descrita en la Sección 1.4.6 1.4.6. Conclusiones: árbitro con cola de prioridades y bus compartido. En la Figura 29 se presenta de nuevo la figura que ya mostramos en aquella sección con el propósito facilitar la lectura.

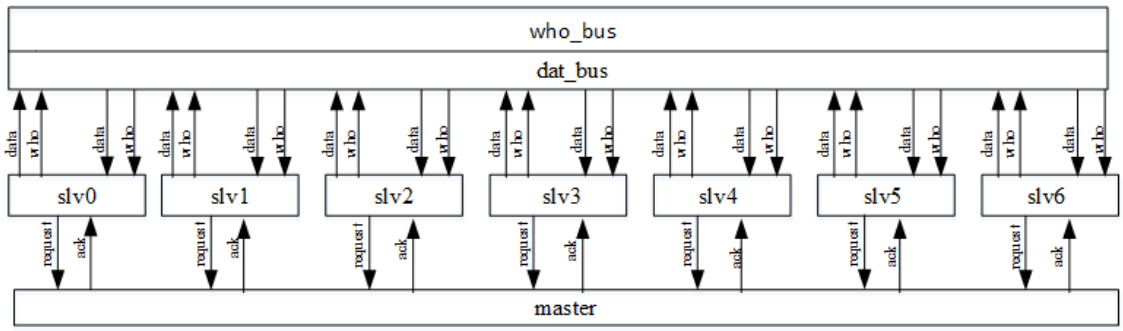

Figura 29 - Diagrama de bloques topología.

Como ya se explicó en la sección anterior, esta entidad consta de dos buses de acceso compartido a todos los módulos del diseño: el bus de datos (*data_bus*) y otro para transferir la identificación del módulo receptor (*who_bus*). Estos datos viajan a la vez, indicando así el contenido de *who_bus* a quién va dirigido el dato que contiene *data_bus*.

Esta topología utiliza una entidad árbitro (entidad master) que gestiona la concesión de los dos buses. En la concesión del bus, el árbitro implementa de forma virtual una topología en anillo como se explicará más adelante, asemejándose el comportamiento del bus al de la topología *token-ring*.

En definitiva, esta topología funciona según el siguiente algoritmo:

1. Un esclavo envía el dato por el bus de datos, y el identificador del receptor por el bus de identificadores. Estos datos viajan en sincronía.

2. Todos los esclavos leen el bus cada ciclo de reloj. Si el identificador que se lee del bus coincide con el identificador del esclavo que leyó el bus, significa que el dato es para ese esclavo.

3. Cuando la entidad conectada al esclavo que recibió el dato ha realizado los cálculos correspondientes, el esclavo realiza una petición del bus a la entidad master.

4. La entidad master va comprobando uno a uno todos los esclavos para ver si se ha solicitado el bus, y en caso de que se haya solicitado, se le concede.

5. Cuando la entidad que solicitó el bus reciba la concesión del mismo, transmitirá el dato por el bus de datos y el identificador del receptor por el bus de identificadores.

Tal y como se contó anteriormente se decidió utilizar este tipo de topología ya que la carga de trabajo para realizar la mejor solución (NoC, CSMA/CD...) era demasiado alta para un proyecto de este tipo y porque lo importante de este proyecto era comprobar la tolerancia a fallos en el conformador trapezoidal.

Cabe destacar que una señal que indicará si los datos son válidos o no viajará en sincronía con a los datos.

3.1.2. Paquete de constantes

Ya que en este proyecto se trabaja con constantes y tipos de datos, se decidió implantar un paquete de constantes para facilitar futuras modificaciones en el diseño. Los tipos y constantes declarados en este paquete pueden observarse en la Tabla 11.

Tipo	Nombre	Valor inicial	Descripción
Slv_id_constants			
constant	c_slv0	“000”	Identificador del esclavo 0 asociado a la etapa DS1
constant	c_slv1	“001”	Identificador del esclavo 1 asociado a la etapa DS2
constant	c_slv2	“010”	Identificador del esclavo 2 asociado a la etapa HPD
constant	c_slv3	“011”	Identificador del esclavo 3 asociado a la etapa ACC
constant	c_slv4	“100”	Identificador del esclavo 4 libre para la reubicación de cualquier etapa
constant	c_slv5	“101”	Identificador del esclavo 5 libre para la reubicación de cualquier etapa
Constantes de ancho de			

datos			
constant	c_data_width	37	Anchura de los datos
constant	c_who_width	3	Anchura de los identificadores
constant	c_default_width	8	Ancho por defecto
Constantes de retraso de la señal valid			
constant	c_delay_ds1	2	Constante de retardo de DS1
constant	c_delay_ds2	2	Constante de retardo de DS2
constant	c_delay_hpd	3	Constante de retardo de HPD
constant	c_delay_acc	2	Constante de retardo de ACC
Declaración de tipos			
subtype	data_t		Tipo de los datos
subtype	who_t		Tipo de los identificadores
Constantes de selección de configuración			
constant	c_conf_sel_width	28	Anchura de los datos de configuración

Tabla 11 – Constantes y tipos.

3.1.3. Master

Esta entidad es la que se encarga de decidir qué esclavo hace uso del bus compartido en qué instante de tiempo, con el fin de evitar que dos esclavos utilicen el bus a la vez.

Para conseguir este resultado se utiliza la máquina de estados finitos descrita en la Figura 30 y que consta de un estado por cada esclavo que tiene conectado. En este diseño esta entidad consta de seis esclavos: cuatro de ellos para las entidades DS1, DS2, HPD y ACC del conformador, y los dos restantes para las dos interfaces redundantes que se definieron en la Sección 3.1. Diseño del conformador trapezoidal tolerante a fallos el caso en que alguno de los otros falle.

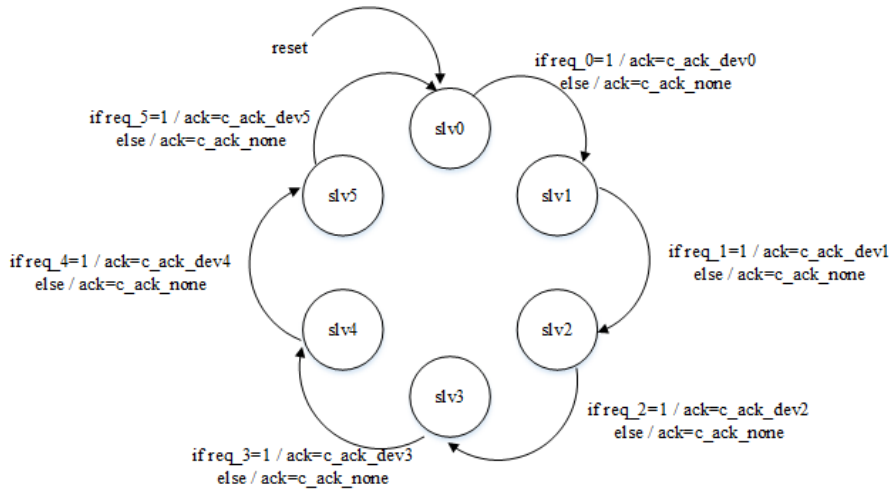

Figura 30 – Máquina de estados.

En la Tabla 12, que se muestra a continuación, se observan los puertos de esta entidad:

Nombre	I/O	Descripción
clk	in	Reloj del sistema
rst_n	in	Reset síncrono activo a baja
req_0	in	Señal de petición del bus del esclavo 0
req_1	in	Señal de petición del bus del esclavo 1
req_2	in	Señal de petición del bus del esclavo 2
req_3	in	Señal de petición del bus del esclavo 3
req_4	in	Señal de petición del bus del esclavo 4
req_5	in	Señal de petición del bus del esclavo 5
ack	out	Señal de concesión del bus 00000 -> concesión al esclavo 0 00001 -> concesión al esclavo 1 00010 -> concesión al esclavo 2 00100 -> concesión al esclavo 3 01000 -> concesión al esclavo 4 10000 -> concesión al esclavo 5

Tabla 12 – Puertos del módulo Master.

Esta entidad está dividida en tres procesos: el primero *p_next_state* que calcula el siguiente estado, el segundo *p_outputs* que calcula la salida, y por último *p_reg_state* que carga el siguiente estado. A continuación se van a explicar estos procesos con más detalle.

Cada ciclo de reloj se carga el estado siguiente (que sigue la topología de un anillo: asignando el estado de la siguiente entidad de esclavo, o de la primera en el caso de que el estado sea el del último esclavo) con el proceso *p_next_state*.

El proceso *p_outputs* carga el puerto de salida ack con el valor de la entidad del esclavo al que se le concede el bus. Se le concede el bus a aquél esclavo que tenga su línea de request activa y la entidad del master esté en el estado de dicho esclavo.

Por último el proceso *p_reg_state* carga cada ciclo de reloj el siguiente estado en el registro del estado actual. Éste registro se inicia al valor del estado del esclavo cero al hacer un reset.

3.1.4. Esclavo

Esta entidad es la que se encarga de escribir y leer los valores en ambos buses (data_bus y who_bus). Para ello está conectada, por un lado, a un módulo del conformador (DS1, DS2, HPD o ACC), y realiza las siguientes tareas:

1. Cuando tiene concedido el bus, escribe los datos proporcionados por el módulo en el bus de datos y el identificador de destinatario en el bus who_bus.
2. Si no tiene concedido el bus, monitoriza los valores en el bus de identificadores (who_bus) y cuando el valor que hay en dicho bus coincide con su identificador de esclavo, lee los datos y se los envía al módulo al que está conectado.

También recibe el vector de configuración (el vector que contiene todos los parámetros de configuración (k, l, m_1 y m_2)) desde el módulo de parámetros para ir a un multiplexor que escoge la configuración correspondiente al dispositivo que tiene conectado. En la siguiente figura se observa el diagrama de bloques de esta entidad:

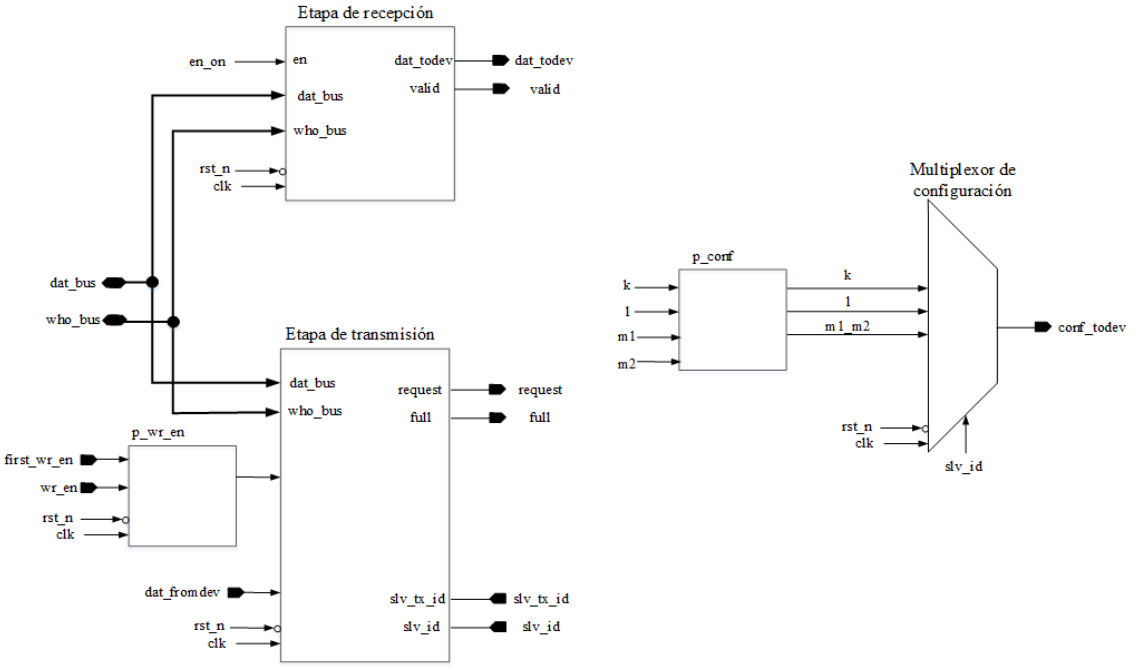

Figura 31 - Diagrama de bloques de la entidad esclavo (slv).

Los puertos de esta entidad se muestran la Tabla 13. El proceso *p_wr_en* genera una señal interna *wr_en_int* que tiene genera el primer dato y se le pasa como puerto de entrada a la entidad *slv_tx* (la etapa de transmisión).

Nombre	I/O	Descripción
clk	in	Reloj del sistema
rst_n	in	Reset síncrono activo a baja
Señales de escritura y error		
first_wr_en	in	Señal para escribir el primer dato en la FIFO del primer esclavo
wr_en	in	Señal para escribir en la FIFO
valid	out	Señal que viaja con el dato indicando que éste es válido
full	out	Señal de error porque la FIFO está llena
Señales del y hacia el master		
ack	in	Señal de concesión del bus 00000 -> concesión al esclavo 0 00001 -> concesión al esclavo 1 00010 -> concesión al esclavo 2 00100 -> concesión al esclavo 3 01000 -> concesión al esclavo 4

		10000 -> concesión al esclavo 5
request	out	Señal de salida para solicitar el bus (si la FIFO no está vacía se solicita el bus)
Datos de entrada y salida		
dat_fromdev	in	Datos entrantes desde el módulo
dat_todev	out	Datos de salida hacia el módulo
Señales de configuración		
k	in	Señal k de configuración desde el módulo parameters
l	in	Señal l de configuración desde el módulo parameters
m1	in	Señal m1 de configuración desde el módulo parameters
m2	in	Señal m2 de configuración desde el módulo parameters
conf_todev	out	Configuración correspondiente al dispositivo hacia el entidad conectada al esclavo
Identificadores		
slv_id	in	Identificador de esclavo procedente al dispositivo conectado
slv_tx_id	in	Identificador del esclavo a quién se le transmiten los datos procedente del dispositivo conectado
Buses		
dat_bus	inout	Bus de datos del que se leen y escriben los datos
who_bus	inout	Bus de identificadores en el que se leen y escriben los identificadores

Tabla 13 – Puertos de la entidad esclavo (slv).

Este módulo está conectado al dispositivo del conformador trapezoidal por los puertos *dat_fromdev* para leer los datos generados por éste, *dat_todev* para escribir los datos leídos del bus de datos, *valid* para indicar que el dato enviado es válido y *conf_todev* para enviar la configuración correspondiente. También está conectado con el módulo de parámetros para recibir todos los parámetros de configuración y enviar el correspondiente al dispositivo.

Esta entidad se divide en tres bloques: uno de recepción que se encarga de leer los buses, otro de transmisión que se encarga que escribir en los buses, y por último un multiplexor para enviar la configuración al módulo del conformador trapezoidal que está conectado al esclavo.

Esta entidad no necesita genéricos.

3.1.4.1. Bloque de recepción

Esta entidad es la que se encarga de leer los datos del bus. Se leen los dos buses (*data_bus* y *who_bus*) y se guardan sus datos en un registro cada ciclo de reloj, también se guarda en un registro en ese mismo ciclo de reloj el puerto de entrada con el identificador del esclavo. Cuando se han cargado tanto el identificador del esclavo, como el identificador del bus, se comparan generando así una señal *match* que indica si estos identificadores son iguales. Esta señal toma como valor uno si ambos identificadores son iguales, es decir, que el dato va dirigido a esta entidad, o cero si no son iguales (el dato no va dirigido a esta entidad).

Esta señal se conecta al puerto *enable* de un registro (*i_dat_todev*) que carga el dato leído del bus para enviarlo al módulo. Con esta señal también se genera la señal *valid* comprobando si ha habido un flanco de subida en la señal *match* (*p_match_rising*) (véase Figura 32).

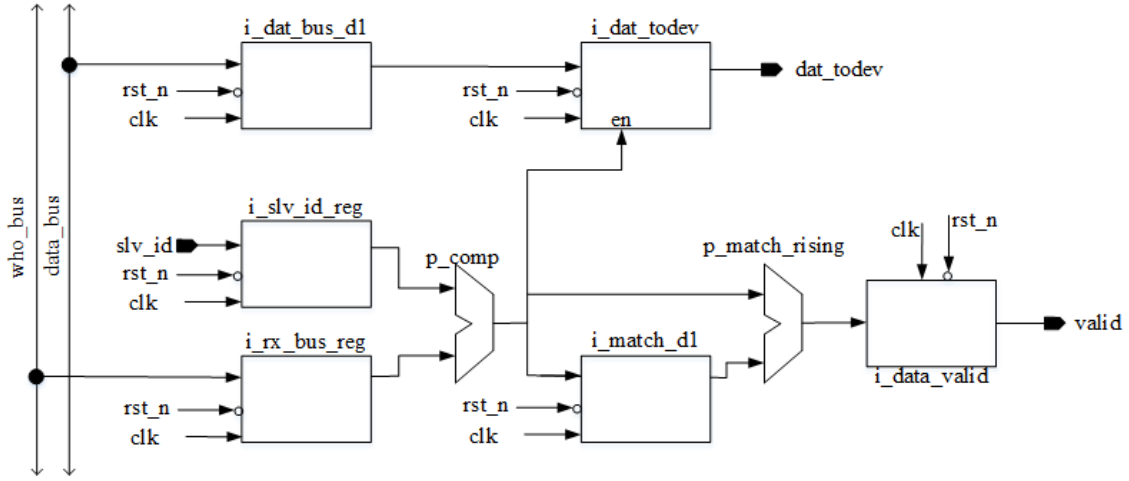

Figura 32 – Diagrama de bloques del bloque de recepción.

En la siguiente tabla se muestran los puertos del módulo slv_rx:

Nombre	I/O	Descripción
clk	in	Reloj del sistema
rst_n	in	*Reset* síncrono activo a baja
dat_bus	in	Bus de datos
who_bus	in	Bus de identificadores
en	in	Señal conectada a uno para que siempre se carguen los registros
slv_id	in	Identificador del esclavo
dat_todev	out	Datos leídos por el bus hacia el dispositivo
valid	out	Señal que indica que el dato enviado es válido

Tabla 14 – Puertos del bloque de recepción (slv_rx).

3.1.4.2. Bloque de transmisión

Esta etapa se encarga de escribir los datos generados por los dispositivos en los buses. Primero se escriben en una FIFO el dato generado por el dispositivo para no perder ningún dato. También se carga cada ciclo el identificador de transmisión en un registro.

Para leer de la FIFO se utiliza la señal *ack* generada por el master, señal que también se retrasa para formar la señal de carga los buffers tri-estado (*p_ack_d1*). Estos buffers propagan el valor de entrada cuando la señal de selección toma el valor 1, y si no colocan la salida a un valor de alta impedancia (Z). Cargan este valor para evitar colisiones. Los buffers se cargan: uno con el dato leído de la FIFO y otro con el identificador de transmisión para escribirlo en los buses.

Estos buffers tri-estado fueron implementados como un multiplexor. Una entrada de selección que permite cargar o bien la entrada al buffer o bien un valor de alta impedancia.

Por último esta entidad también se encarga de generar la señal de petición del bus (*request*) que vale uno si la FIFO no está vacía, y cero si está vacía.

El diagrama de bloques de esta etapa se muestra en la siguiente figura:

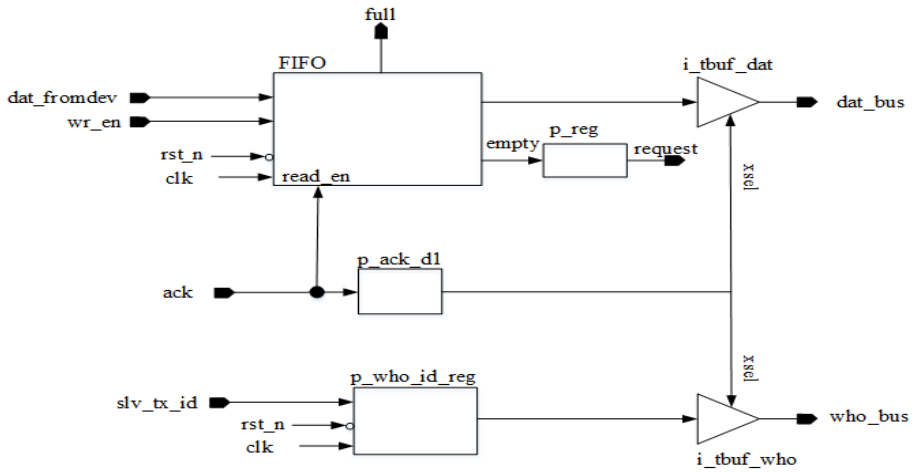

Figura 33 – Diagrama de bloques del bloque de transmisión.

En la siguiente tabla se muestran los puertos de esta etapa:

Nombre	I/O	Descripción
clk	in	Reloj del sistema
rst_n	in	*Reset* síncrono activo a baja
ack	in	Señal que habilita la lectura de la FIFO y la carga de los buffers tri-estado
wr_en	in	Señal que habilita la escritura en la FIFO
dat_fromdev	in	Datos que vienen desde el dispositivo
slv_tx_id	in	Identificador del esclavo al que se le envía el dato
request	out	Salida de concesión del bus
dat_bus	out	Bus de datos
who_bus	out	Bus de identificadores
full	out	Señal de error que indica que la FIFO está llena

Tabla 15 - Puertos del bloque de transmisión (slv_tx).

La entidad FIFO de la que consta este módulo fue generada utilizando la herramienta CoreGenerator. Fue generada como una entidad *shift/register* para que tuviese una baja complejidad a la hora de implementarla en el sistema y se le dotó de un ancho de 37 bits, con el fin de que tuviese la misma anchura que los datos de salida. Esta FIFO escribe y lee en un único ciclo de reloj.

3.1.4.3. Multiplexor de configuración

El módulo *configuration_mux* incluido en la entidad esclavo se encarga de proporcionar la configuración adecuada a cada dispositivos dependiendo de su id. La configuración adecuada es: el parámetro k al dispositivo *DS1* (con *id* 000), el parámetro l al dispositivo *DS2* (con *id* 001) y los parámetros m_1 y m_2 al dispositivo *HPPD* (con id 010).

Esta entidad se basa en un multiplexor que tiene tres entradas: una entrada es k, otra es l, y la otra entrada es una señal que tiene los parámetros m_1 y m_2. Todas estas señales tienen el mismo ancho (*c_conf_sel_width*). Con el identificador del dispositivo (*slv_id*) se selecciona una entrada u otra, correspondiendo así la configuración correcta a cada dispositivo.

3.1.5. Bus fabric

Esta entidad es la que da cuerpo a la topología de comunicación uniendo una entidad de Master, seis entidades de Slave (una para cada dispositivo DS1, DS2, HPD y ACC y las dos restantes para que sea posible la reubicación de las etapas), y los dos buses (data_bus y who_bus). En la Figura 34 puede observarse el diagrama de bloques correspondiente a esta entidad.

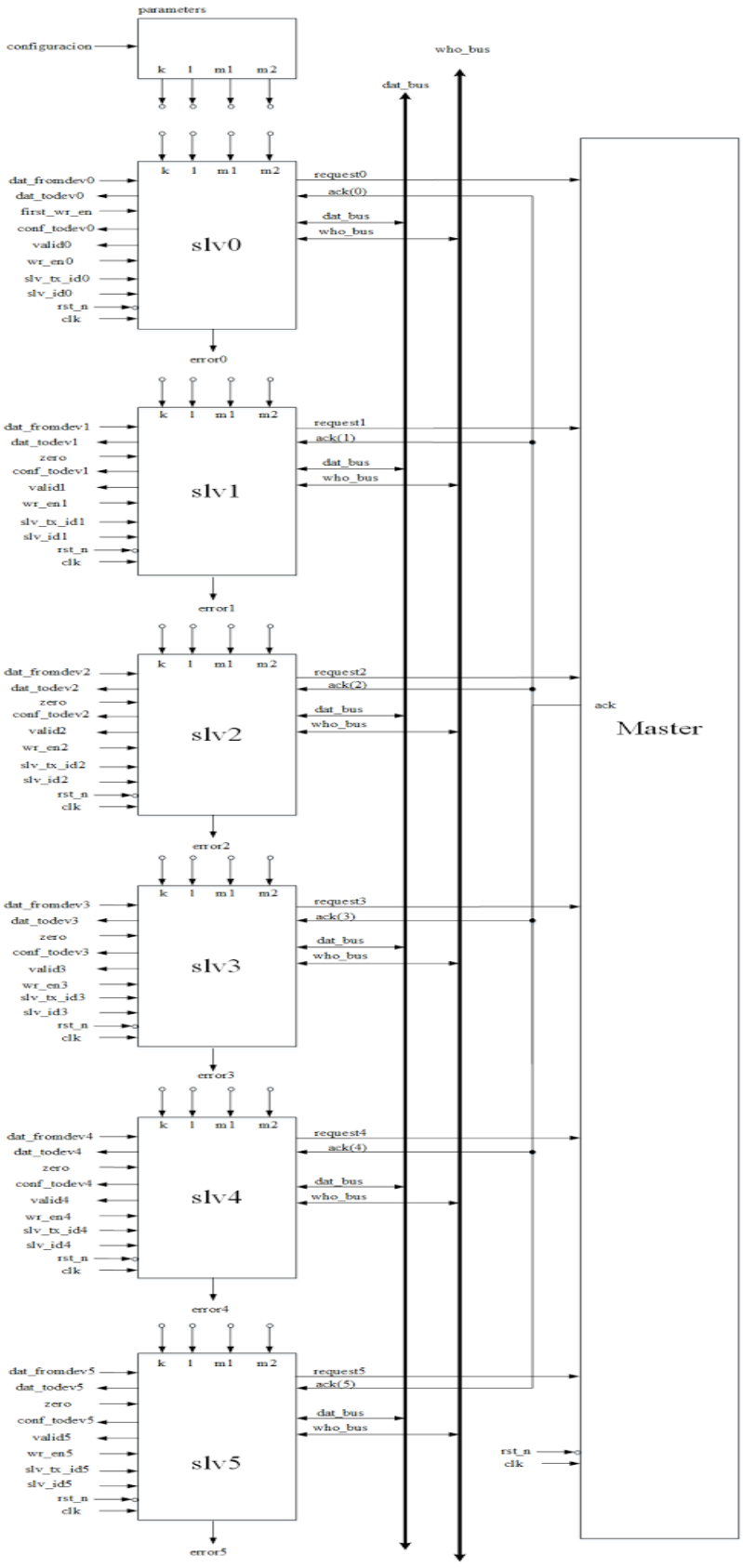

Figura 34 - Diagrama de bloques del módulo bus_fabric.

En la siguiente tabla se observan los puertos de este módulo:

Nombre	I/O	Comentario
clk	in	Reloj del sistema
rst_n	in	*Reset* síncrono activo a baja
configuracion	in	Configuración que lleva todos los parámetros (k, l, m1 y m2)
slv0		
dat_fromdev0	in	Datos de entrada desde el dispositivo 0
wr_en0	in	Señal que habilita la escritura en la FIFO 0
slv_tx_id0	in	Id destino
slv_id0	in	Id de esclavo
dat_todev0	out	Datos de salida hacia el dispositivo 0
conf_todev0	out	Configuración hacia el dispositivo 0
valid0	out	Señal que indica que el dato es válido
slv1		
dat_fromdev1	in	Datos de entrada desde el dispositivo 1
wr_en1	in	Señal que habilita la escritura en la FIFO 1
slv_tx_id1	in	Id destino
slv_i1	in	Id de esclavo
dat_todev1	out	Datos de salida hacia el dispositivo 1
conf_todev1	out	Configuración hacia el dispositivo 1
valid1	out	Señal que indica que el dato es válido
svl2		
dat_fromdev2	in	Datos de entrada desde el dispositivo 2
wr_en2	in	Señal que habilita la escritura en la FIFO 2
slv_tx_id2	in	Id destino
slv_id2	in	Id de esclavo
dat_todev2	out	Datos de salida hacia el dispositivo 2
conf_todev2	out	Configuración hacia el dispositivo 2
valid2	out	Señal que indica que el dato es válido
slv3		
dat_fromdev3	in	Datos de entrada desde el dispositivo 3
wr_en3	in	Señal que habilita la escritura en la FIFO 3
slv_tx_id3	in	Id destino
slv_id3	in	Id de esclavo
dat_todev3	out	Datos de salida hacia el dispositivo 3
conf_todev3	out	Configuración hacia el dispositivo 3
valid3	out	Señal que indica que el dato es válido
slv4		
dat_fromdev4	in	Datos de entrada desde el dispositivo 4
wr_en4	in	Señal que habilita la escritura en la FIFO 4
slv_tx_id4	in	Id destino
slv_id4	in	Id de esclavo
dat_todev4	out	Datos de salida hacia el dispositivo 4
conf_todev4	out	Configuración hacia el dispositivo 4
valid4	out	Señal que indica que el dato es válido

slv5		
dat_fromdev5	in	Datos de entrada desde el dispositivo 5
wr_en5	in	Señal que habilita la escritura en la FIFO 5
slv_tx_id5	in	Id destino
slv_id5	in	Id de esclavo
dat_todev5	out	Datos de salida hacia el dispositivo 5
conf_todev5	out	Configuración hacia el dispositivo 5
valid0	out	Señal que indica que el dato es válido

Tabla 16 - Puertos del módulo bus_fabric.

Cada esclavo está conectado a un módulo mediante sus puertos de entrada. Reciben datos desde el dispositivo por el puerto *dat_fromdev* a la vez que reciben el identificador del esclavo que es (*slv_id*) y el identificador del esclavo al que envían el mensaje (*slv_tx_id*). Con el puerto *wr_en* que viene desde el módulo del conformador se indica a la FIFO del esclavo de la etapa de transmisión que escriba un dato. Este puerto se consigue con la señal *valid* que genera el propio esclavo gracias al módulo *valid2wr* que se explica más adelante.

El esclavo proporciona al módulo los datos leídos del bus por el puerto *dat_todev* y la configuración del propio módulo por el puerto *conf_todev* gracias al multiplexor *configuration_mux*. El esclavo también genera una señal de *valid* indicando que el dato (*dat_todev*) que lee es válido mientras que la señal de *valid* está a uno.

El módulo *parameters* proporciona toda la configuración (k, l, m_1 y m_2) a los esclavos y éstos pasan dicha configuración al dispositivo correspondiente.

Los genéricos de este módulo se muestran en la siguiente tabla:

Nombre	Valor	Descripción
g_indata_width	14	Ancho de los datos de entrada
g_chrom_len	40	Ancho de los parámetros de configuración
g_outdata_width	37	Ancho de los datos de salida

Tabla 17 – Genéricos del módulo bus_fabric (top-level).

3.1.6. Valid to write

El módulo (*valid2wr*) se encuentra dentro de las entidades *ds_wrapper*, *hpd_wrapper* y *acc_wrapper*. Se encarga de generar la señal de escritura en la FIFO del esclavo a partir de la señal *valid* que le llega al dispositivo desde el esclavo.

La señal de escritura en la FIFO se genera retrasando la señal *valid* que viaja junto al dato leído tantos ciclos de reloj como se tarda en generar el dato de salida a partir del dato de entrada. Gracias a esto al dispositivo le entran un dato y una señal de válido, y le salen (en el mismo ciclo de reloj) un dato de salida y una señal de escritura en la FIFO. Los retardos asociados a cada dispositivo están en el paquete de constantes: *c_delayds1* y *c_delayds2* tienen como valor 2, *c_delay_hpd* tiene como valor 3 y *c_delay_acc* tiene como valor 2. En la Figura 35 se muestra un cronograma en el que se observa cómo la señal *wr_en* se retrasa 2 ciclos respecto a la señal *valid*.

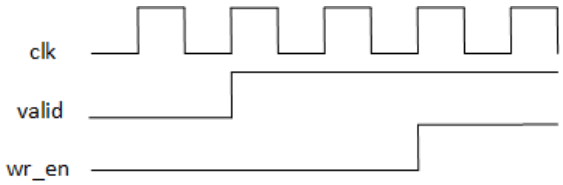

Figura 35 – Cronograma del módulo valid2wr.

Tal y como se observa en la Figura 36 este módulo está implementado como un registro y un *for generate* de tantos registros como el genérico de retardo (que toma el valor de las constantes antes mencionadas) menos uno debido al primer registro.

El proceso *p_wr_en* carga el valor del puerto *wr_en* con el valor de la posición del vector *wr_aux* indicada por el genérico *g_valid_delay*. Así se consigue cargar el valor *valid* retrasado *g_valid_delay* ciclos en el puerto *wr_en*.

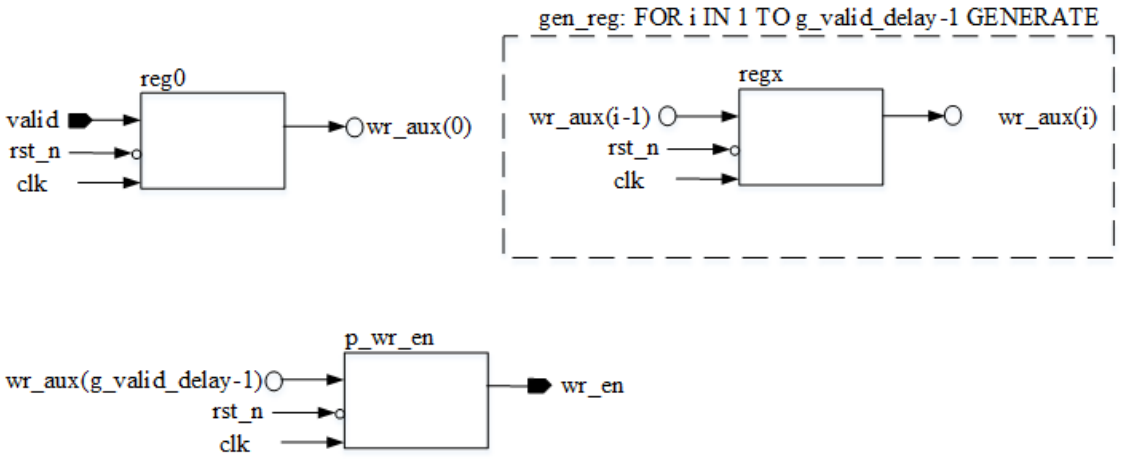

Figura 36 - Diagrama de bloques del módulo valid2wr.

En la siguiente tabla se indican los puertos de esta entidad:

Nombre	I/O	Comentario
clk	in	Reloj del sistema
rst_n	in	*Reset* síncrono activo a baja
valid	in	Señal de dato válido
wr_en	out	Señal de habilitación de escritura en la FIFO del esclavo

Tabla 18 - Puertos de la entidad valid2wr.

Esta entidad tiene los siguientes genéricos:

Nombre	Valor	Descripción
g_valid_delay	4	Número de ciclos que se retarda la señal *valid* de forma genérica (cuando se instancie esta entidad se le dará a éste genérico el valor de cada dispositivo *c_delay_ds1*, *c_delay_ds2*, *c_delay_hpd* o *c_delay_acc* que se encuentran en el paquete de constantes)

Tabla 19 - Genéricos del módulo valid2wr.

3.1.7. DS wrapper

Debido a que las entidades *DS1* y *DS2* tienen la misma implementación con parámetros diferentes no se crean dos entidades (*DS1* y *DS2*), sino una única entidad *DS* que da cuerpo a ambos módulos. Se tuvo que poner el mismo ancho en todas las señales, la única que hubo que cambiar fue la señal de entrada de datos, que para *DS1* era de una anchura de *g_indata_width* y para *DS2* era de *g_outdata_width*, por lo que se decidió hacer que el módulo *DS1* tuviera una entrada de datos de *g_outdata_width* bits de anchura.

Con el fin de extender la funcionalidad del módulo *DS* para permitir la comunicación con el bus se crea una entidad *ds_wrapper*. Para conseguir esta funcionalidad el módulo *ds_wrapper* contiene una entidad *DS* y una entidad *valid2wr*. Este módulo se diferencia en dos a la hora de la implementación: *ds1_wrapper* y *ds2_wrapper*. La diferencia se presenta en valor de los genéricos *g_valid_delay*, *g_slv_id* y *g_slv_tx_id*.

En la Figura 37 se observa el proceso *p_conf* que carga del puerto *conf_fromslv* (enviado desde el esclavo con el parámetro correspondiente: k para *DS1* y l para *DS2*) a una señal de la anchura indicada para el parámetro.

También se observa el proceso *p_slv_id* que carga en los puertos de salida *slv_id* y *slv_tx_id* los identificadores correspondientes. Por último se observa que la salida del módulo *DS* se conecta a un registro, esto es para conseguir que la señal *wr_en* y el dato van en sincronía al módulo *bus_fabric*.

El diagrama de bloques de este módulo se observa en la siguiente figura:

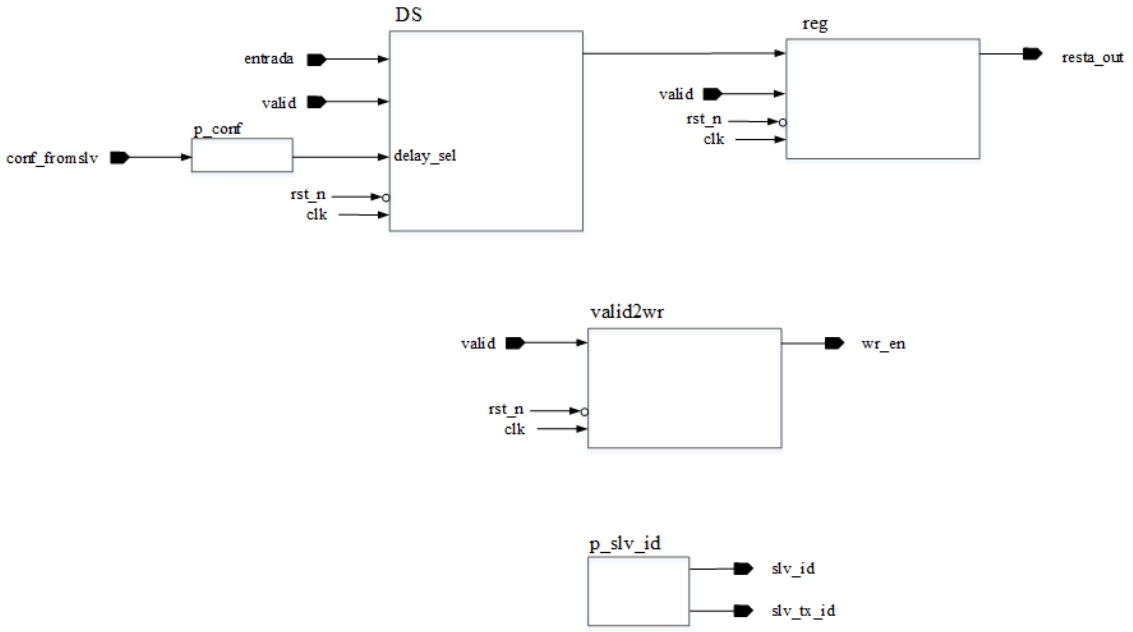

Figura 37 - Diagrama de bloques del módulo DS wrapper.

Los puertos de este módulo se muestran en la Tabla 20.

Nombre	I/O	Comentario
clk	in	Reloj del sistema
rst_n	in	*Reset* síncrono activo a baja
valid	in	Señal de dato válido que activa los registros del dispositivo
conf_fromslv	in	Configuración que se otorga al dispositivo desde el esclavo
entrada	in	Datos de entrada al dispositivo
slv_id	out	Identificador del esclavo que se envía al esclavo (este identificador se carga por los genéricos)
slv_tx_id	out	Identificador del esclavo al que se envía el dato (este identificador se carga por los genéricos)
resta_out	out	Datos de salida del dispositivo
wr_en	out	Señal de habilitación de escritura en la FIFO del esclavo

Tabla 20 - Puertos del módulo ds_wrapper.

Los identificadores (tanto el identificador del esclavo, como el identificador del esclavo a transmitir) se configuran con los genéricos (véase Tabla 21).

Nombre	Valor	Descripción
g_valid_delay	4	Número de ciclos que se retarda la señal *valid* de forma genérica (cuando se instancie esta entidad se le dará a éste genérico el valor de cada dispositivo *c_delay_ds1*, *c_delay_ds2*, *c_delay_hpd* o *c_delay_acc* que se encuentran en el paquete de constantes)
g_slv_id	"000"	Genérico que indica el identificador del esclavo
g_slv_tx_id	"000"	Genérico que indica el identificador del esclavo al que se le envía el dato

g_outdata_width	37	Ancho de los datos de salida
g_conf_sel_width	c_conf_sel_width	Anchura de la configuración del dispositivo

Tabla 21 - Genéricos del módulo ds_wrapper.

3.1.8. HPD wrapper

El módulo *hpd_wrapper* tiene una entidad *HPD* (explicada en la sección 2.2.2. Módulo HPD), una entidad *valid2wr* (para generar la señal *wr_en* a partir de la señal *valid* tal y como se explicó en el apartado 3.1.6. Valid to write) y dos procesos (*p_conf* que dada la configuración desde el esclavo la divide en los parámetros m1 y m2 y *p_slv_id* que carga los identificadores de esclavo). El diagrama de bloques se observa en la Figura 38.

También se observa que la salida del módulo *DS* se conecta a un registro, esto es para conseguir que la señal *wr_en* y el dato viajen juntos al módulo *bus_fabric*.

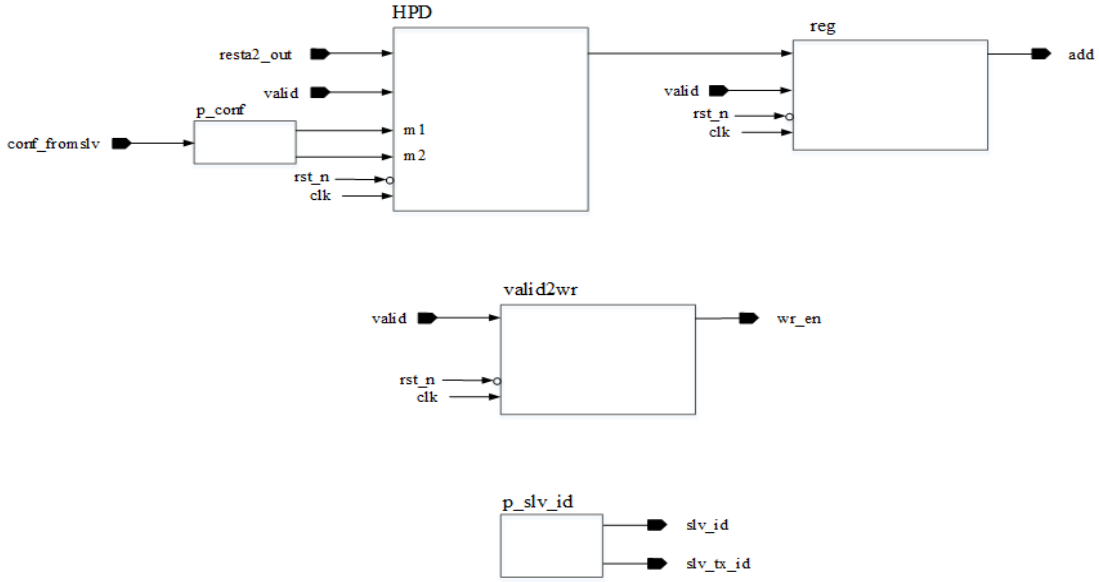

Figura 38 - Diagrama de bloques del módulo hpd_wrapper.

Sus puertos se muestran en la siguiente tabla:

Nombre	I/O	Comentario
clk	in	Reloj del sistema
rst_n	in	*Reset* síncrono activo a baja
valid	in	Señal de dato válido que activa los registros del dispositivo
conf_fromslv	in	Configuración que se otorga al dispositivo desde el esclavo
resta2_out	in	Datos de entrada al dispositivo
slv_id	out	Identificador del esclavo que se envía al esclavo (este identificador se carga por los genéricos)
slv_tx_id	out	Identificador del esclavo al que se envía el dato (este identificador se carga por los genéricos)
add	out	Datos de salida del dispositivo
wr_en	out	Señal de habilitación de escritura en la FIFO del esclavo

Tabla 22 - Puertos del módulo hpd_wrapper.

Por último, los genéricos de este módulo son los siguientes:

Nombre	Valor	Descripción
g_valid_delay	4	Número de ciclos que se retarda la señal *valid* de forma genérica (cuando se instancie esta entidad se le dará a éste genérico el valor de cada dispositivo *c_delay_ds1*, *c_delay_ds2*, *c_delay_hpd* o *c_delay_acc* que se encuentran en el paquete de constantes)
g_slv_id	"000"	Genérico que indica el identificador del esclavo
g_slv_tx_id	"000"	Genérico que indica el identificador del esclavo al que se le envía el dato
g_outdata_width	37	Ancho de los datos de salida
g_conf_sel_width	c_conf_sel_width	Anchura de la configuración del dispositivo

Tabla 23 – Genéricos del módulo hpd_wrapper.

3.1.9. ACC wrapper

El módulo *acc_wrapper* tiene una entidad acc (explicada en la sección 2.2.3. Módulo ACC), una entidad valid2wr (para generar la señal *wr_en* a partir de la señal *valid*) y un proceso (*p_slv_id*) para cargar los identificadores de los esclavos. No consta del proceso *p_conf* ya que este módulo no necesita ningún parámetro de configuración (véase Figura 39).

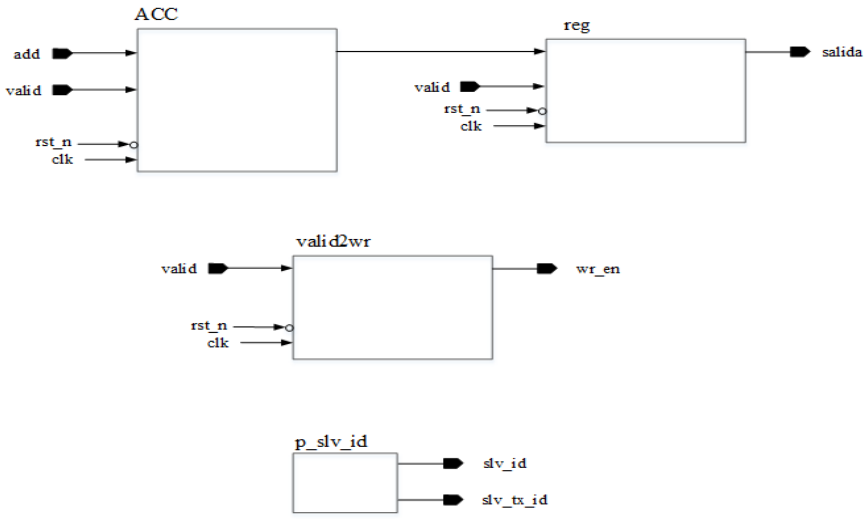

Figura 39 - Diagrama de bloques del módulo acc_wrapper.

Sus puertos se muestran en la siguiente tabla:

Nombre	I/O	Comentario
clk	in	Reloj del sistema
rst_n	in	*Reset* síncrono activo a baja
valid	in	Señal de dato válido que activa los registros del dispositivo
add	in	Datos de entrada al dispositivo
slv_id	out	Identificador del esclavo que se envía al esclavo (este identificador se carga por los genéricos)
slv_tx_id	out	Identificador del esclavo al que se envía el dato (este identificador se carga por los genéricos)
salida	out	Datos de salida del dispositivo
wr_en	out	Señal de habilitación de escritura en la FIFO del esclavo

Tabla 24 - Puertos del módulo acc_wrapper.

Los genéricos que utiliza este módulo se muestran en la siguiente tabla:

Nombre	Valor	Descripción
g_valid_delay	4	Número de ciclos que se retarda la señal *valid* de forma genérica (cuando se instancie esta entidad se le dará a éste genérico el valor de cada dispositivo *c_delay_ds1*, *c_delay_ds2*, *c_delay_hpd* o *c_delay_acc* que se encuentran en el paquete de constantes)
g_slv_id	"000"	Genérico que indica el identificador del esclavo
g_slv_tx_id	"000"	Genérico que indica el identificador del esclavo al que se le envía el dato
g_outdata_width	37	Ancho de los datos de salida

Tabla 25 - Genéricos del módulo acc_wrapper.

3.1.10. Black-boxes

En este diseño existe un error debido a que las entidades redundantes de conexión a los buses que se comentaron anteriormente tienen sus puertos desconectados. Para subsanar este error se decidió conectar unas entidades sin ningún tipo de lógica en su interior, consiguiendo así que los puertos de entrada de las entidades redundantes no estuvieses desconectados. Estas entidades que se crearon son las *black-boxes*. En la Figura 40 se observa su diagrama de bloques. Los procesos *p_wr_en*, *p_dat_tomaster*, *p_slv_id* y *p_slv_tx_id* inyectan valores inocuos para el sistema.

Los valores que se inyectan son unos, ya que no existe ninguna entidad con un identificador todo unos. También cabe destacar que así no habría posibilidad que dos esclavos tuviesen el mismo identificador.

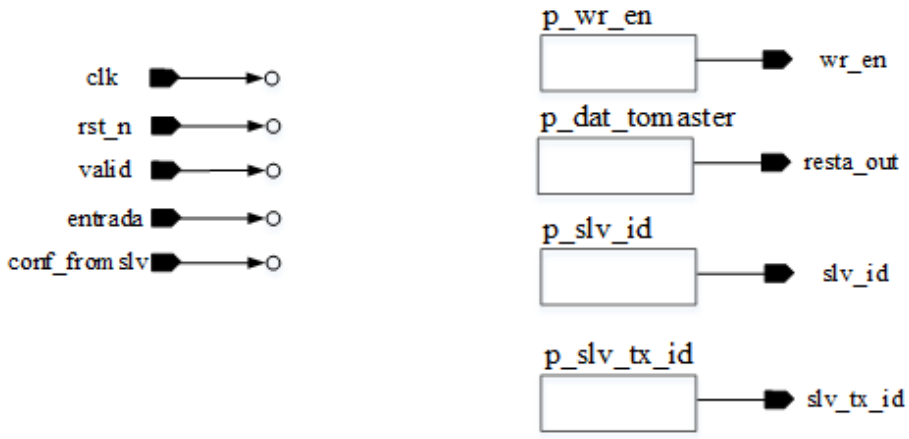

Figura 40 - Diagrama de bloques del módulo black-box.

Los puertos que tiene este módulo se muestran en la siguiente tabla:

Nombre	I/O	Comentario
clk	in	Reloj del sistema
rst_n	in	*Reset* síncrono activo a baja
valid	in	Señal de dato válido que activa los registros del dispositivo
conf_fromslv	in	Configuración que se otorga al dispositivo desde el esclavo
entrada	in	Datos de entrada al dispositivo
slv_id	out	Identificador del esclavo que se envía al esclavo (este identificador se carga por los genéricos)
slv_tx_id	out	Identificador del esclavo al que se envía el dato (este identificador se carga por los genéricos)
resta_out	out	Datos de salida del dispositivo
wr_en	out	Señal de habilitación de escritura en la FIFO del esclavo

Tabla 26 - Puertos del módulo black_box.

Por último, los genéricos utilizados en esta entidad son los siguientes:

Nombre	Valor	Descripción
g_valid_delay	4	Número de ciclos que se retarda la señal *valid* de forma genérica (cuando se instancie esta entidad se le dará a éste genérico el valor de cada dispositivo *c_delay_ds1*, *c_delay_ds2*, *c_delay_hpd* o *c_delay_acc* que se encuentran en el paquete de constantes)
g_slv_id	"000"	Genérico que indica el identificador del esclavo
g_slv_tx_id	"000"	Genérico que indica el identificador del esclavo al que se le envía el dato
g_outdata_width	37	Ancho de los datos de salida
g_conf_sel_width	c_conf_sel_width	Anchura de la configuración del dispositivo

Tabla 27 - Genéricos del módulo black_box.

3.1.11. Top level (trapezoidal_noc)

En la Figura 41 se muestra el diagrama de bloques del *trapezoidal_noc* con todos los componentes anteriormente explicados. Se puede observar que están conectados los componentes *DS1*, *DS2*, *HPD* y *ACC* a los cuatros primeros esclavos del módulo *bus_fabric* y que las *black-boxes* están conectadas a los dos esclavos restantes. El proceso *p_entrada_xtnd* extiende el puerto entrada de anchura *g_indata_width* a una anchura de *g_outdata_width*. El proceso *p_salida_aux* sirve para evitar que el puerto de entrada del esclavo conectado al dispositivo *acc_wrapper* esté desconectado.

En esta entidad se han incluido unas señales de errores. Hay una señal de error por cada esclavo. Estas señales provienen del puerto del esclavo que indica que la FIFO que está llena.

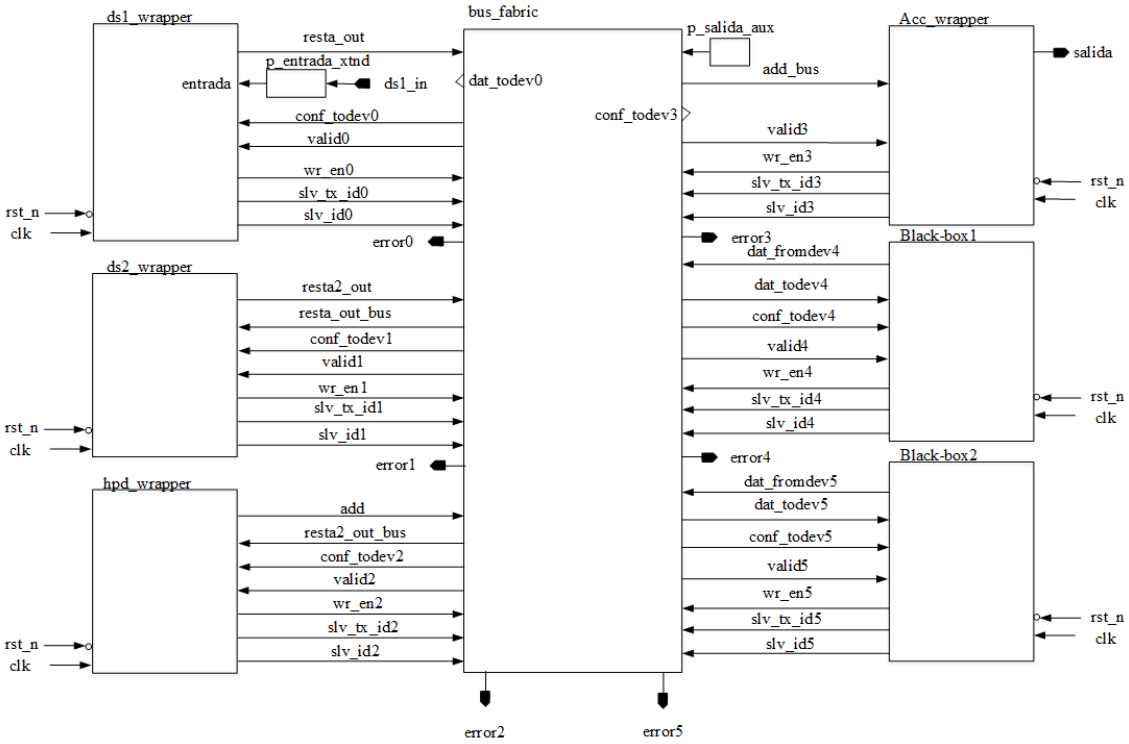

Figura 41 - Diagrama de bloques del módulo trapezoidal_noc.

Los puertos de la entidad *trapezoidal_noc* se muestran en la siguiente tabla:

Nombre	I/O	Comentario
clk	in	Reloj del sistema
rst_n	in	*Reset* síncrono activo a baja
ds1_in	in	Datos de entrada al conformador trapezoidal tolerante a fallos
configuración	in	Configuración de entrada con los parámetros k, l, m1 y m2
first_wr_en	in	Señal que indica la primera escritura
error0	out	Señal de error en el primer esclavo
error1	out	Señal de error en el segundo esclavo
error2	out	Señal de error en el tercer esclavo
error3	out	Señal de error en el cuarto esclavo
error4	out	Señal de error en el quinto esclavo
error5	out	Señal de error en el sexto esclavo
salida	out	Datos de salida del conformador trapezoidal tolerante a fallos
salida_en	out	Señal de escritura del dispositivo que otorga los datos de salida (sirve para indicar que el dato de salida es válido)

Tabla 28 - Puertos del módulo trapezoidal_noc.

Los genéricos de esta entidad son los siguientes:

Nombre	Valor	Descripción
g_indata_width	14	Ancho de los datos de entrada
g_chrom_len	14	Ancho de la configuración
g_outdata_width	37	Ancho de los datos de salida

Tabla 29 - Genéricos del módulo trapezoidal_noc.

Cabe destacar que los módulos atados a puertos de entrada o salida (*ds1_wrapper* y *acc_wrapper*) no pueden tener la característica de ser reconfigurables, ya que no se pueden reubicar si están atacados por un puerto. Esto es debido a que la conexión con los puertos es punto a punto (tal y como se explicó en la sección 1.4. Topologías y protocolos de comunicación en chips).

Por esto se debería haber optado por introducir los datos de entrada y de salida en la entidad estática *bus_fabric* y que fuese ésta la que le concediese estos datos a la entidad *ds1_wrapper* y a la entidad *acc_wrapper*. Atando así los puertos de entrada y salida al módulo estático que no estará dotado de reconfiguración. Definitivamente no se optó por esta modificación debido a la falta de tiempo.

3.2. Simulación

Con el fin de comprobar el correcto funcionamiento del diseño del conformador trapezoidal tolerante a fallos *trapezoidal_*noc se realizó una simulación (véase Figura 42). Esta simulación se realiza sobre la entidad *trapezoidal_noc* a la cual se la inyectan los datos leídos desde el fichero *stimuli.txt* y la cual escribe los datos de salida en el fichero *response.txt*. El proceso *p_config* proporciona la configuración (con los parámetros k, l, m_1 y m_2) al conformador trapezoidal tolerante a fallos (*trapezoidal_noc*).

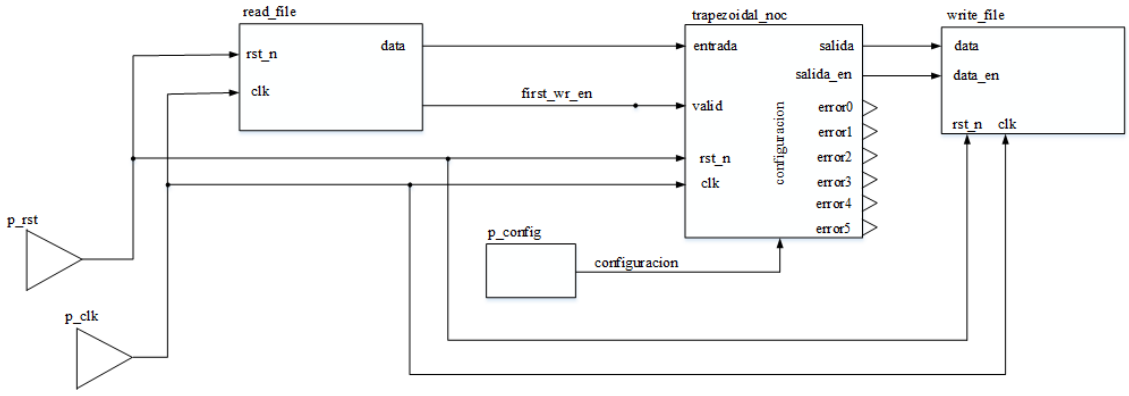

Figura 42 - Implementación del módulo tb_trapezoidal_noc.

Para obtener una representación gráfica de los resultados de la simulación se creó un script en *Matlab* (*print_stimuli.m*) que lee el fichero de datos de entrada (*stimuli.txt*) y los pinta como una función, y hace lo mismo para los datos de salida (*response.txt*). Estas funciones deberían ser: una función exponencial para los datos de salida y una función con forma de trapezoide

para los datos de salida (véanse la Figura 43 y la Figura 44). Tal y como se observa en la Figura 43, la salida forma una función con forma de trapezoide, exactamente igual a como se mostraba en la salida del conformador trapezoidal sin tolerancia a fallos, por lo cual se sabe que los datos generados por el conformador trapezoidal tolerante a fallos son los correctos.

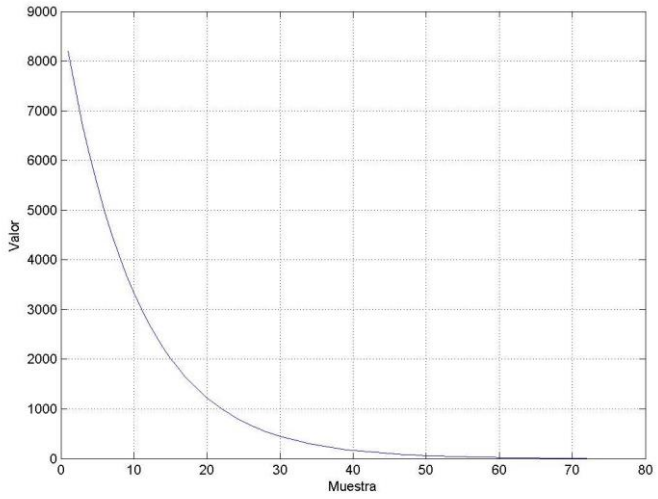

Figura 43 - Entrada conformador trapezoidal tolerante a fallos.

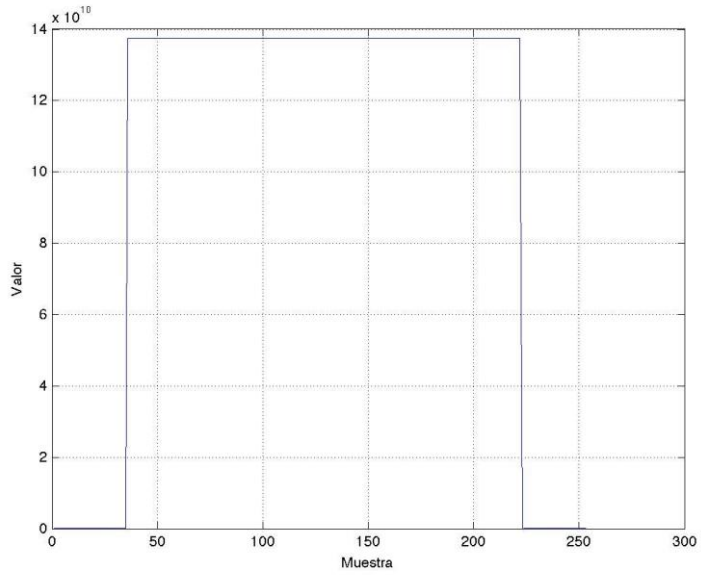

Figura 44 - Salida conformador trapezoidal tolerante a fallos.

3.3. Síntesis

La siguiente etapa en el flujo de diseño del *trapezoidal*_noc es realizar la síntesis del diseño. Para ello se creó el script que se comentará a continuación. Para realizar la síntesis se ha utilizado la herramienta Xilinx-ISE 14.1 (*Integrated Software Environment*).

3.3.1. Scripts de síntesis

Para sintetizar el sistema del conformador trapezoidal tolerante a fallos se creó el ejecutable *synth_trap_noc*. Para este proyecto se creó un directorio T: para evitar problemas con los *path* de trabajo en el que se guardaron todos los archivos relacionados con el proyecto, en este ejecutable lo primero que se hace es ir al directorio en el que se encuentra la implementación del proyecto del conformador trapezoidal (*T:/trapezoidal_noc/impl/trapezoidal_noc*), lo siguiente que se hace es que si no existe el directorio *xst*, se crea y se crea el directorio *projnav.tmp* en este directorio. A continuación se ejecuta el siguiente script:

```
xst -intstyle ise -filter
"T:/trapezoidal_noc/tools/ise/trapezoidal_noc/iseconfig/filter.filter" -ifn
"T:/trapezoidal_noc/impl/trapezoidal_noc/trapezoidal_noc_simple.xst" -ofn
"T:/trapezoidal_noc/impl/trapezoidal_noc/trapezoidal_noc_simple.syr"
```

Las opciones del archivo *xst* son las mismas que las explicadas en la sección 2.4.1. Scripts de síntesis. Las opciones más interesantes de este archivo son las siguientes [22] (Apéndice C):

1. **`set -tmpdir "T:\trapezoidal_noc\impl\trapezoidal_noc\xst_simple \projnav.tmp"`**
   Se crea el directorio donde se guardarán los archivos intermedios.

2. **`set -xsthdpdir "xst"`**
   Esta línea indica que se ha creado un directorio *xst* donde se guardarán los archivos de caché del *xst*.

3. **`-ifn trapezoidal_noc_simple.prj`**
   Se indica que la lista de fuentes que integran el proyecto se encuentran en el archive *trapezoidal.prj* (Apéndice D).

4. **`-ofn trapezoidal_noc_simple`**
   Indica que se use *trapezoidal* como nombre base para los archivos generados.

5. **`-ofmt NGC`**
   Indica que el formato de salida será *ngc* (archivo de *netlist* con información de restricciones).

6. **`-p xc6vlx240t-1-ff1156`**
   Se indica la placa utilizada.

7. **`-top trapezoidal_noc`**
   Se establece el nombre de la entidad top.

8. **`-keep_hierarchy Yes`**
   Si la jerarquía se mantiene durante la síntesis (como es en este caso), se preserva también la jerarquía en la implementación. Permite crear una netlist de simulación con la jerarquía deseada. .

9. **`-sd {"T:/trapezoidal_noc/impl/ip"}`**
   Con esta línea se indica el path en el que se encuentras los cores *edif* o *ngc*.

10. **-hierarchy_separator /**
Se indica que el carácter separador de las jerarquías es /.

11. **-bus_delimiter <>**
Indica que el delimitador de bus se realiza mediante los caracteres <>.

12. **-uc T:\trapezoidal_noc/impl/trapezoidal/trapezoidal_noc.xcf**
Indica el archive de restricciones de síntesis utilizado (el período del reloj del sistema es de 20 ns).

13. **-iobuf YES**
Se permiten *buffers* de entrada/salida en el diseño.

14. **-max_fanout 100**
Se limita el máximo fanout a 100.

15. **-bufg 32**
Se permiten un máximo de 32 buffers en la señal de reloj (valor máximo permitido por la placa).Esta opción sirve para evitar la degradación de la señal de reloj en aquellas entidades lejanas al cristal de cuarzo que la genera.

Por último se indica el archivo de salida *.syr* a generar en el que se volcarán los resultados de la síntesis con la opción *-ofn*.

3.3.2. Resultados de implementación

En esta sección se va a interpretar los datos devueltos por la síntesis en el archivo *trapezoidal_noc_simple.syr* generado al sintetizar. Lo primero que destaca de este archivo es que se utilizan 5667 registros, y una vez optimizada la lógica por la herramienta de *Xilinx* se utilizan 5224 registros.

También cabe destacar que se utilizan 100 buffers de entrada/salida (56 de entrada y 44 de salida). El período mínimo de reloj de este diseño es de 7.454ns, es decir, una frecuencia máxima de 134.156MHz.

3.3.3. Análisis y comparación

Si se comparan los resultados de la simulación de este diseño con respecto a la simulación del conformador trapezoidal sin tolerancia a fallos cabe destacar que antes se utilizaban 4810 registros (1866 optimizados) y ahora se utilizan 5667 (5224 optimizados). Este resultado es debido al aumento de lógica de la entidad *bus_fabric*, y de las entidades *valid2wr* dentro de cada *wrapper* (*ds1_wrapper*, *ds2_wrapper*, *hdp_wrapper* y *acc_wrapper*) que añaden 3358 registros (5224-1866).

También cabe destacar que antes había 93 buffers de entrada y salida y ahora hay 100. Ambos cuentan con el mismo número de buffers de entrada (56), pero lo que varía este número de buffers son los buffers de salida, teniendo el conformador trapezoidal un total de 37 y el conformador trapezoidal con tolerancia a fallos un total de 44. Esta variación se debe a las seis

salidas de error (una por esclavo) y a la salida que indica el dato de salida como válido (*salida_en*), que hacen en total los siete buffers de salida restantes.

También cabe destacar que el período mínimo de reloj es menor en el diseño del conformador trapezoidal con tolerancia a fallos. Cabría esperar que este tiempo fuese mayor que en el conformador trapezoidal sin tolerancia a fallos ya que, en el fondo, el diseño del conformador trapezoidal tolerante a fallos está segmentado y hay que añadirle el retardo de la comunicación por el bus. Sin embargo, debido a que está segmentado, el tiempo que muestra la herramienta es menor, ya que únicamente se muestra el tiempo entre dos registros. El camino crítico se muestra en la siguiente figura:

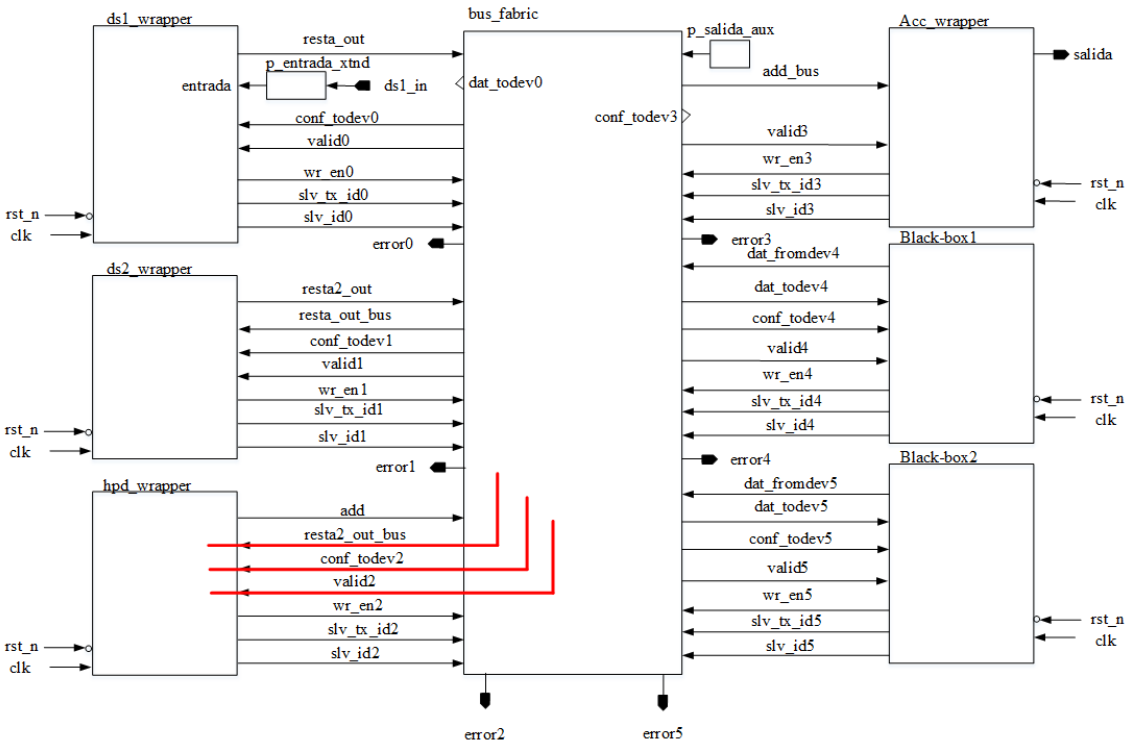

Figura 45 - Camino crítico del módulo trapezoidal_noc.

La salida del camino crítico del archivo trapezoidal_noc_simple.syr se muestra a continuación. En este archivo puede observarse que el camino crítico comienza en el multiplexor de configuración del esclavo 2 de la entidad *bus_fabric* (*i_bus_fabric/i_slv2/i_configuration_mux*). Y de éste pasa a la entidad *i_hpd_wapper*. En esta entidad atraviesa el camino inferior (*i_mult2*, *i_extender_m2*, *i_acc1* y *i _adder:add*).

```
Data Path: i_bus_fabric/i_slv2/i_configuration_mux/conf_27 (FF) to
i_hpd_wrapper/i_reg/q_36 (FF)
                                  Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     end scope: 'i_bus_fabric/i_slv2/i_configuration_mux:conf<27>'
     end scope: 'i_bus_fabric:conf_todev2<27>'
     begin scope: 'i_hpd_wrapper:conf_fromslv<27>'
     begin scope: 'i_hpd_wrapper/i_hpd:m2<13>'
```

```
begin scope: 'i_hpd_wrapper/i_hpd/i_mult2:op2<13>'
end scope: 'i_hpd_wrapper/i_hpd/i_mult2:mult<0>'
begin scope: 'i_hpd_wrapper/i_hpd/i_extender_m2:unextended<0>'
end scope: 'i_hpd_wrapper/i_hpd/i_extender_m2:extended<0>'
begin scope: 'i_hpd_wrapper/i_hpd/i_acc1:acc_in<0>'
begin scope: 'i_hpd_wrapper/i_hpd/i_acc1/i_adder_acc:op1<0>'
end scope: 'i_hpd_wrapper/i_hpd/i_acc1/i_adder_acc:add<35>'
end scope: 'i_hpd_wrapper/i_hpd/i_acc1:acc<35>'
begin scope: 'i_hpd_wrapper/i_hpd/i_adder:op2<35>'
end scope: 'i_hpd_wrapper/i_hpd/i_adder:add<36>'
end scope: 'i_hpd_wrapper/i_hpd:dout<36>'
begin scope: 'i_hpd_wrapper/i_reg:d<36>'
FDRE:D                    0.011          q_36
----------------------------------------
Total                     7.454ns (6.069ns logic, 1.385ns route)
                                  (81.4% logic, 18.6% route)
```

Pese a que el tiempo del camino crítico es menor, como ya se ha comentado, el tiempo que transcurre hasta que se consigue la salida es mucho mayor en el conformador trapezoidal tolerante a fallos, que en el que no consta de tolerancia. Cabe destacar que, como siempre, una vez el conformador esté trabajando de manera normal con una gran cantidad de datos, gracias a la segmentación el diseño será más rápido.

Gracias a las formas de onda generadas al ejecutar la simulación de ambas implementaciones (*trapezoidal* y *trapezoidal_noc*) se puede observar claramente la diferencia en el tiempo que tarda la entidad *trapezoidal* en dar el primer dato y lo que tarda la entidad *trapezoidal_noc*. En la Figura 46 se muestra la forma de onda generada por la simulación de la entidad *trapezoidal* y se observa que la primera salida se genera en el ciclo 11, a unos 125ns del comienzo. En la Figura 47, que muestra la forma generada por la simulación de la entidad *trapezoidal_noc*, se contempla que esta implementación genera el primer dato en el ciclo 59, es decir, a unos 580ns del comienzo. Esto se debe a que en el *trapezoidal_no* los datos tienen que atraveras todos los módulos (que ahora tienen un registro a su salida) además de los retardos asociados a la petición, a la concesión del bus y a la transmisión de los datos por el mismo. Cabe destacar que el dato de entrada entra en el mismo ciclo en ambas figuras.

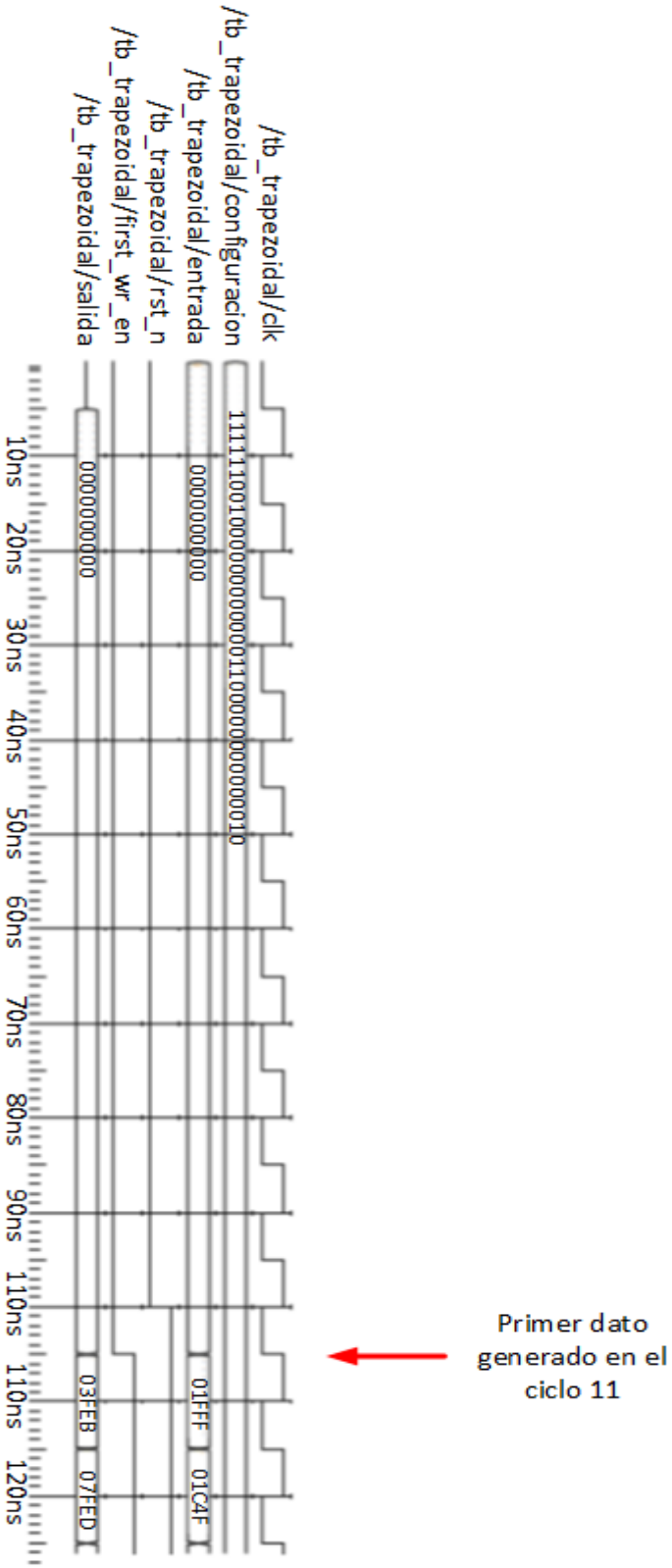

Figura 46 - Forma de onda generada por la simulación de trapezoidal.

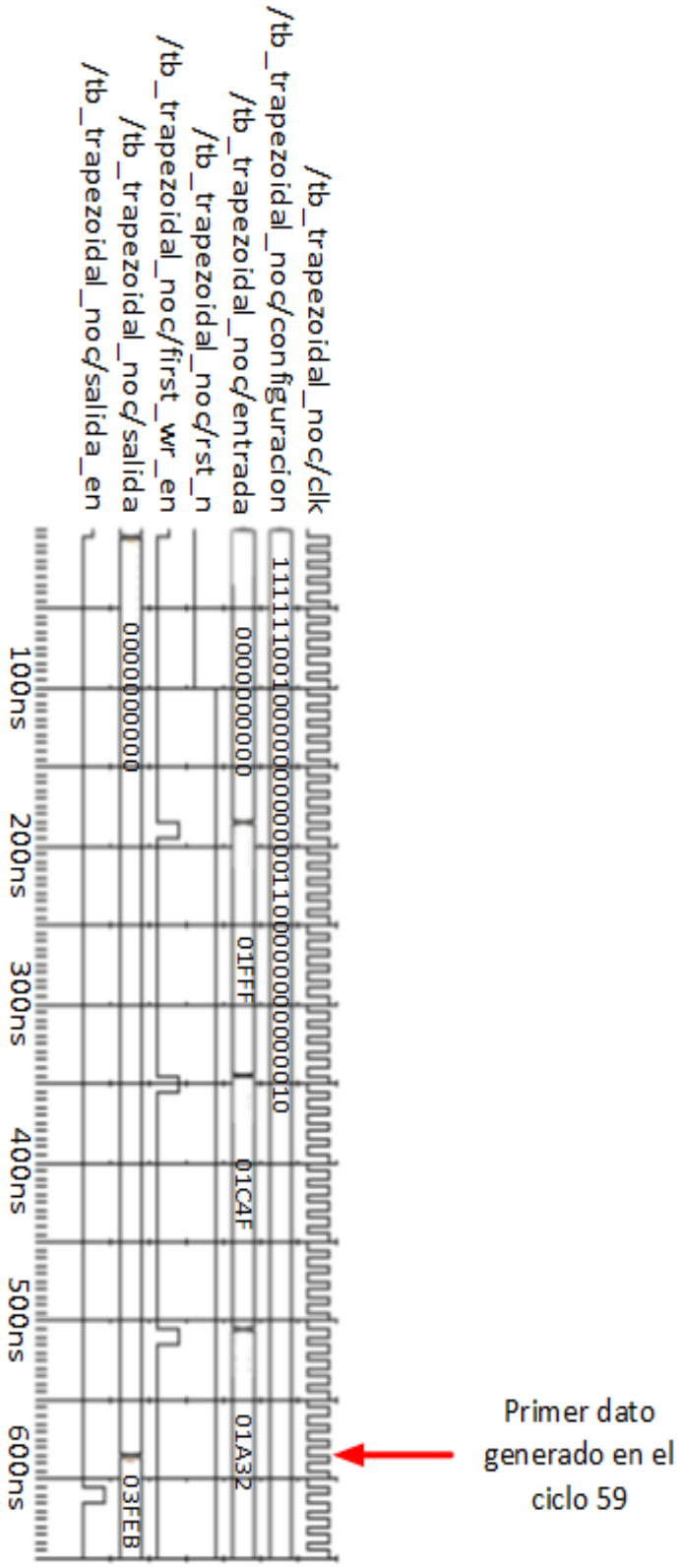

Figura 47 - Forma de onda generada por la simulación de trapezoidal_noc.

Capítulo 4

Reconfiguración Parcial Dinámica

Finalmente, una vez que se ha (1) completado el diseño, la verificación y la implementación del conformador trapezoidal, y (2) completado el rediseño, la verificación y la implementación del conformador con comunicaciones mediante bus compartido, en este capítulo se va a explicar cómo se han creado las distintas configuraciones (distintos ubicaciones de los módulos del conformador) del conformador haciendo uso de la reconfiguración dinámica parcial de la FPGA y dotando de esta forma al conformador trapezoidal de tolerancia a fallos en el sustrato físico.

Para comprobar el correcto funcionamiento de la reconfiguración se crearon dos configuraciones:

1. La configuración inicial (*init_config*) con los bloques *ds1_wrapper*, *ds2_wrapper*, *hpd_wrapper* y *acc_wrapper* conectados a los primeros cuatro esclavos, y las *black_box* conectadas en los dos últimos esclavos;

2. Una segunda configuración (*ds2_permuting*) en al que el módulo *ds2_wrapper* está conectado donde antes estaba una *black_box* y viceversa.

4.1. Definición de proyecto

La reconfiguración dinámica parcial ha sido implementada utilizando la herramienta *PlanAhead v14.1* de *Xilinx*. Esta herramienta, entre muchas otras funciones, permite asignar la reconfiguración dinámica parcial a un sistema gracias a la creación de particiones dinámicamente reconfigurables, en las que se puede inyectar cualquier archivo *ngc* con el fin de dotar de la funcionalidad del archivo inyectado a la partición.

Para crear un proyecto en *PlanAhead 14.1* es necesario un diseño previamente sintetizado. En este caso en el *top level* es necesario que las partes que van a ser estáticas, es decir, las que no van a cambiar de una configuración a otra, estén totalmente sintetizadas, y que las partes dinámicas del diseño estén declarados como cajas negras en las que no se indica ninguna funcionalidad. También es necesario sintetizar por separado las partes dinámicas para tener los archivos *ngc* generados por la síntesis con el fin de poder inyectarlos en las cajas negras del top *level*.

Con el fin de conseguir esta síntesis se creó el script *synth_trap_noc_reconfig* (Apéndice E) en el que se sintetizan los distintos módulos. Este script se explica a continuación.

Se empieza sintetizando los módulos dinámicos, estos módulos se sintetizan con la opción -*bufg* 0 en el correspondiente archivo *xst* con el fin de que no se permita insertar buffers en la señal de reloj. Además tampoco se permiten buffers de entrada o salida en los módulos reconfigurables. Los módulos que se sintetizan son *black_box*, *ds_wrapper*, *hpd_wrapper* y *acc_wrapper*. Cabe destacar que en las síntesis de estos módulos se indican los genéricos con los que contará cada módulo en su correspondiente archivo *xst*; es por esto que para el módulo *ds_wrapper*, que tiene dos instancias (*ds1_wrapper* y *ds2_wrapper*) que únicamente varían en el valor de los genéricos (los identificadores) es necesario generar dos scripts de síntesis indicándole a cada uno un archivo *xst* en los que se indican sus correspondientes genéricos (*ds1_wrapper.xst* y *ds2_wrapper.xst*).

Por último se sintetiza el top *level* (*trapezoidal_noc*) indicando en su archivo *xst* que se puede *bufferear* el reloj e indicando en su archivo *.prj* únicamente los módulos que necesita la parte estática y el propio módulo del *trapezoidal_noc*. Puesto que los módulos dinámicos no han sido incluidos en el archivo *prj*, en esta síntesis se toman estos módulos como cajas negras (tal y como se puede observar en el archivo creado *.syr*).

Debido a que esta última síntesis tiene totalmente sintetizada la lógica estática y tiene declarada la lógica dinámica como cajas negras, será la que se utilizará para el proyecto de *PlanAhead*. Se utiliza esta síntesis con el fin de inyectar los ficheros *ngc* conseguidos de los módulos dinámicos en las cajas negras indicadas en la simulación del *top level*.

4.2. Ejecución del proyecto

Para crear las dos configuraciones del diseño en *PlanAhead* se creó un script TCL (*planahead.tcl*). Este script se muestra en el Apéndice F.

En este script se crea el proyecto para la placa indicada en el directorio indicado. Se le añaden los ficheros *.ngc* de la *FIFO* del bloque estático *bus_fabric* (*slv_fifo.ngc*) y el archivo *.ngc* del diseño *top level* (*trapezoidal_noc.ngc*) generados en la síntesis.

A continuación se crea el archivo de restricciones (*trapezoidal.ucf*), en el que se fija dónde estarán situados los distintos módulos (*acc_wrapper*, *ds1_wrapper*, *ds2_wrapper*, *hpd_wrapper*, *black_box1* y *black_box2*) sobre el *layout* de la FPGA. También se pone el valor del período del reloj a 20 ns.

El siguiente paso es describir las propiedades del proyecto, indicando que es un proyecto con lógica parcialmente reconfigurable, que el simulador por defecto es *QuestaSim*, que el lenguaje por defecto es VHDL y el nombre de la primera configuración (*initial_config*).

A continuación se crean las particiones reconfigurables (los *wrappers* de *DS1*, *DS2*, *hpd* y *acc*). En esta sección se apunta en qué celda se ubica cada módulo reconfigurable. Después se crean las cajas negras, en este apartado se señala que en la caja negra 2 se puede insertar también el módulo *DS2* creando el módulo reconfigurable *ds2_wrapper* en la celda *i_bb_2*.

Finalmente se lanza la implementación de la configuración inicial. Una vez lanzada hay que promover la configuración, es decir, utilizar sus bloques estáticos y dinámicos como puntos de partida para otras configuraciones ya que en otras configuraciones únicamente se modificarán los bloques que se indiquen. Cabe destacar que antes de promover la implementación, ésta debe haber terminado, es por eso que se espera hasta que termine.

Una vez lanzada esta configuración, se crea una segunda configuración (que será explicada con más detalle en la siguiente sección) y se lanza.

Para terminar se comprueba que ambas configuraciones tienen la misma lógica estática y los mismos bits de partición. Una vez termina de hacer esta comprobación se puede comprobar en la consola tcl que la verificación se ha completado satisfactoriamente (Verification completed successfully!).

4.3. Configuraciones

Se han creado dos configuraciones distintas para comprobar el correcto funcionamiento del sistema cuando se reconfigura alguno de sus módulos. En las configuraciones que se crearon se cambió de lugar el módulo *ds2_wrapper* por el módulo *black_box* 2. En los siguientes apartados se explican estas configuraciones con más detalle.

Para estas configuraciones se creó una celda en el *layout* de la FPGA por cada dispositivo (indicadas en el archivo *.tcl*). Se tienen así dos celdas para las cajas negras (una para *i_bb1* y otra para *i_bb2*), dos celdas para el módulo *ds_wrapper* (una para *ds1_wrapper* y otra para *ds2_wrapper*), una para el módulo *acc_wrapper* y, por último, otra para *hpd_wrapper*.

4.3.1. Configuración inicial

La configuración inicial que se implementó se realizó de forma que cada módulo se situara en su respectiva celda del layour de la FPGA creados anteriormente. En la Figura 48 se observa cómo se han situado los diferentes módulos. En esta figura la lógica viene representada como “puntos” de un color claro diferente al fondo.

Se puede observar en la parte superior izquierda dos celdas delimitadas que no constan de lógica en su interior (ya que no hay ningún “punto” en su interior), estas celdas representan las cajas negras. Debajo de éstas se observan otras dos entidades, la superior representa el módulo *i_acc*_wrapper y la inferior el módulo *i_ds1*_wrapper. En la parte derecha de la figura se observan dos celdas más, la superior indica el módulo *i_ds2_wrapper*, y la inferior el módulo *i_hpd_wrapper*. Por último se observan partes de lógica fuera de las celdas comentadas, estas partes representan la entidad de comunicación (*bus_fabric*).

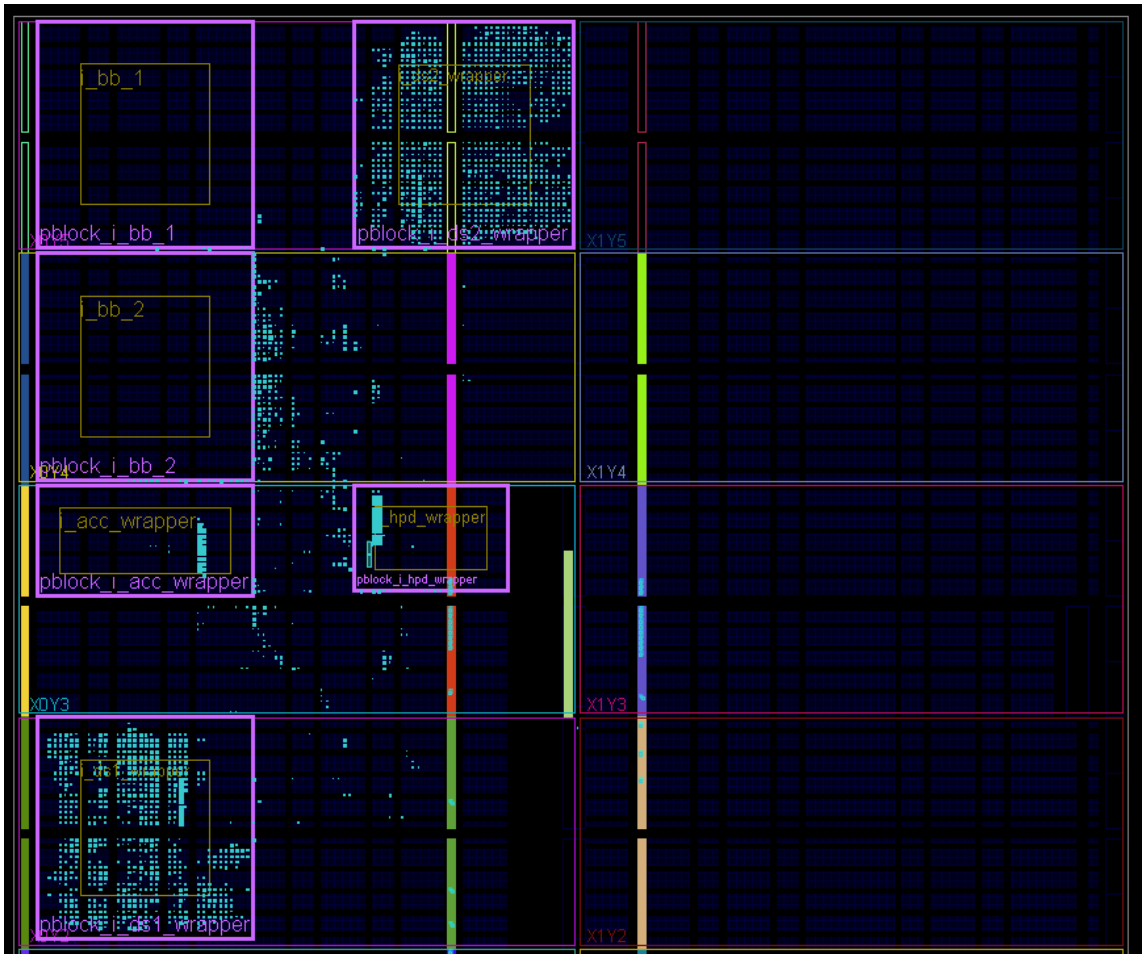

Figura 48 - Configuración inicial.

4.3.2. Configuración con el módulo DS2 permutado

Esta segunda configuración que se implementó se realizó de forma que cada módulo se situara en su respectiva celda de la placa física como la anterior configuración salvo dos módulos que se permutan (*i_ds2_wrapper* y *i_bb_2*: tomando *i_ds2_wrapper* la posición de *i_bb_2* y viceversa), en la Figura 49 se observa cómo se han situado los diferentes módulos. Se puede observar en la parte superior izquierda dos celdas delimitadas. Cabe destacar que en la anterior configuración es estas celdas estaban situadas las entidades de las cajas negras, sin embargo, en esta configuración se observa que la celda inferior consta de lógica, siendo esta lógica la del módulo *i_ds2_wrapper*.

Debajo de éstas se observan otras dos entidades, la superior representa el módulo *i_acc*_wrapper y la inferior el módulo *i_ds1*_wrapper. En la parte derecha de la figura se observan dos celdas más, cabe destacar que en la anterior configuración estas celdas representaban los módulos *i_ds2_wrapper* y *_i_hpd_wrapper*, pero tal y como se observa en la Figura 49, la primera celda está libre de lógica, encontrándose en esa celda la entidad *i_bb_2*. Por último se observan partes de lógica fuera de las celdas comentadas, estas partes representan la entidad de comunicación (*bus_fabric*).

En definitiva, en la Figura 49 se observa que los módulos *i_ds2_wrapper* y *i_bb_2* se han intercambiado de lugar.

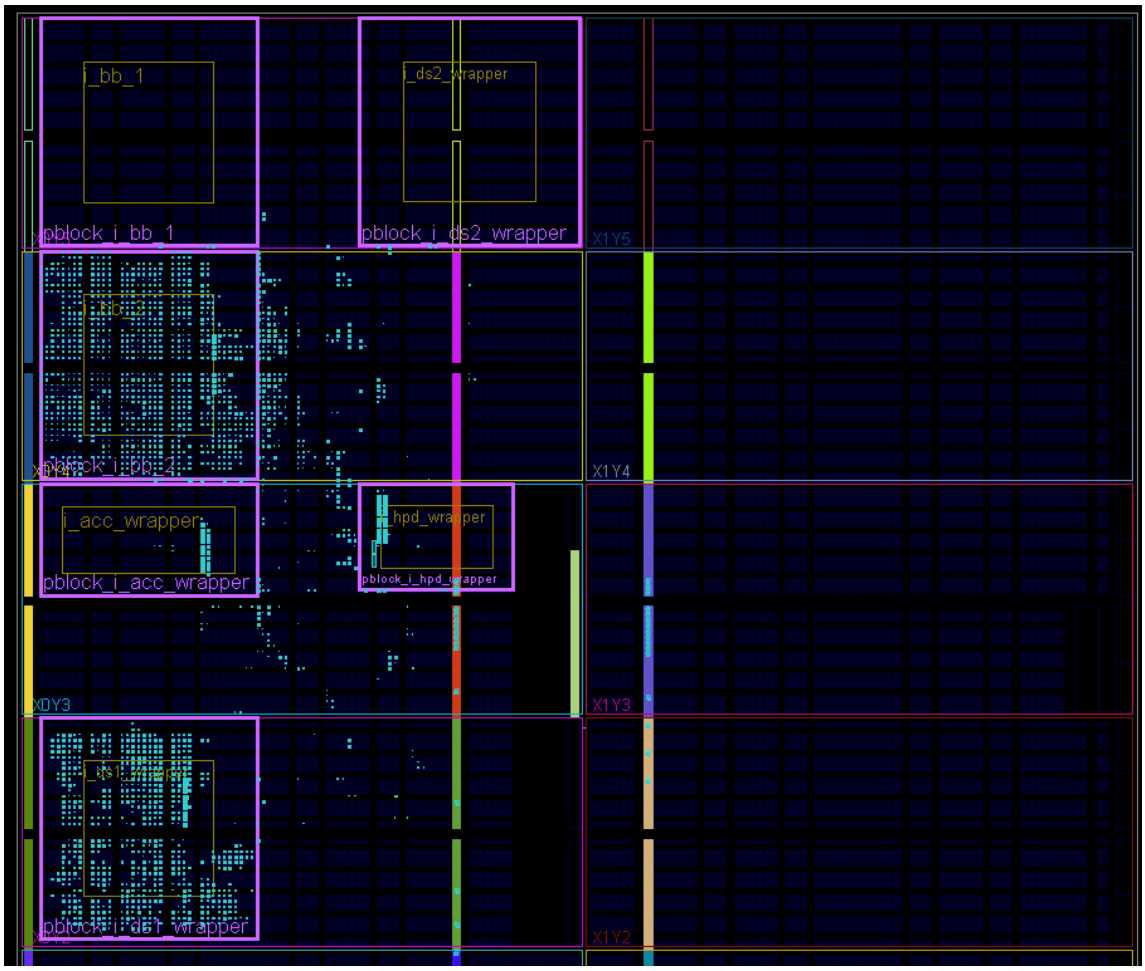

Figura 49 - Configuración con el módulo DS2 permutado.

4.4. Simulación de las configuraciones

Por último, con el fin de comprobar que ambas configuraciones dan el mismo resultado, se realizó una simulación de cada una de ellas por separado para finalmente compararlas. Si se obtuviese el mismo resultado en ambas simulaciones, significaría que el diseño del conformador trapezoidal tolerante a fallos funciona correctamente utilizando una reconfiguración parcial.

"Antes de realizar una simulación de temporización, la información sobre el diseño físico debe ser traducida y distribuida de nuevo al diseño lógico. Para las *FPGAs*, este proceso de back-anotación se hace con un programa llamado *NetGen*. Estos programas crean una base de datos, que traduce la información back-anotada en un formato de lista de conexiones que se puede utilizar para medir el tiempo de simulación [21]."

Antes de ejecutar esta simulación se creó el script *postPR_netlist* para crear la *netlist back*-anotada en el que se crean varios archivos para cada configuración (los más destacables son un *vhd* y un *sdf* en el que se indican los retardos) con el siguiente script (para la configuración inicial):

```
netgen -sim -ofmt vhdl
T:\trapezoidal_noc\planahead\planahead.runs\initial_config\initial_config.ncd
```

La opción *-sim* indica que se va a generar una *netlist* de simulación indicando con la opción *-ofmt* el *path* de salida de esta simulación y el lenguaje utilizado (*vhdl*) [24].

Este script genera los archivos *initial_config.nfl* en el que se indican los comandos ejecutados en el script y la información, los errores o los *warnings* que genera. Se genera también el archivo *initial_config.sdf* en el que se guardan todos los retardos correspondientes a las instancias del diseño. Por último se genera un archivo *initial_config.vhd* en el que se indica la conexión de las instancias indicadas en el archivo *sdf*.

4.4.1. Simulación de la configuración inicial

Para lanzar la simulación de la primera configuración explicada en la Sección 4.3.1. Configuración inicial se creó un script (*postPR_initial_config*) en el que en primera instancia se crean dos directorios, uno para la entidad *trapezoidal_noc* y otro para la entidad tb_*trapezoidal_noc* dentro del directorio *work/compilation*. A continuación se mapean las librerías *trapezoidal_noc* y *tb_trapezoidal_noc* en sus respectivos directorios con el comando *vmap*.

Después de esto se compilan lo archivos *vhd*, primero se compila el archivo *initial_config.vhd* generado anteriormente sobre el directorio *trapezoidal_noc* para indicar que la DUT (*Design Under Test*) será este archivo. A continuación se compilan los archivos *vhd* necesarios para el test en el directorio *tb_trapezoidal_noc* (*read_file.vhd* y *tb_trapezoidal_noc.vhd*).

Para terminar se lanza la simulación con el siguiente comando:

```
vsim -sdftyp
/tb_trapezoidal_noc/dut=T:/trapezoidal_noc/sim/initial_config/initial_config.
sdf tb_trapezoidal_noc.tb_trapezoidal_noc -do wave.do
```

Con la opción –sdftyp se indica que se utilizan los retardos típicos. Es decir, que se aplican los retardos del archivo *sdf*.

En este comando se indica que la DUT utilizada en esta simulación cumple con los retardos del archivo *initial_config.sdf*.

El test que se lanza en esta sección es el mismo que se ha utilizado durante todo el proyecto, salvo que la DUT es la configuración inicial (*initial_config*) tal y como se observa en la siguiente figura:

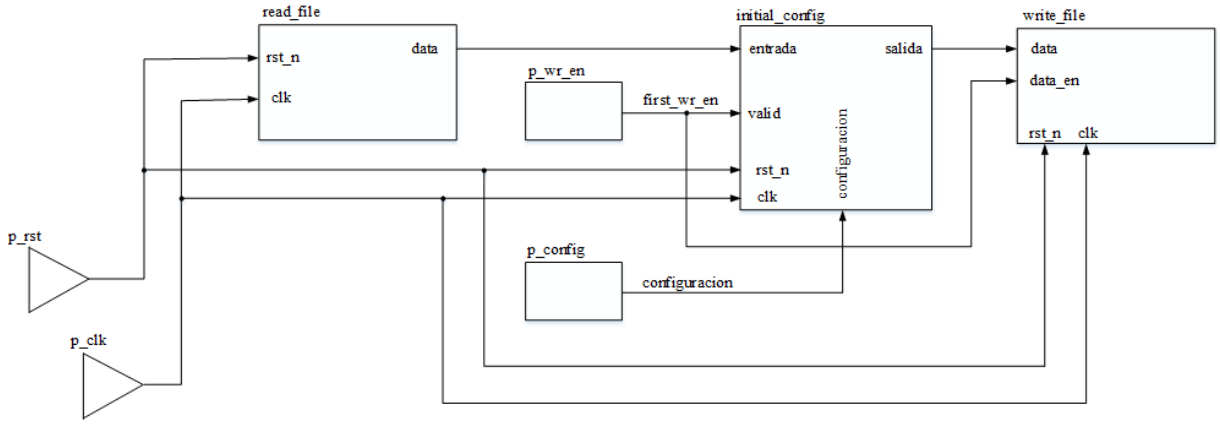

Figura 50 - Implementación tb_trapezoidal_noc initial_config

Tras lanzar este comando, se arranca la herramienta de simulación Questa mostrando la forma de onda del comportamiento del diseño al que se le ha realizado el test. Gracias al módulo *write_file* del testbench explicado en la Sección 2.3. Simulación se genera un fichero con las salidas generadas por esta entidad.

Esta simulación dio errores de retardos en señales (skew). En particular la señal de capacitación de los registros llegaba más tarde a algunos componentes que la señal de reloj. En la Figura 51 se observa cómo la línea de capacitación del chip (*CE*), que debería llegar al chip antes que la señal de reloj en verdad llega después. Este problema causa que el chip no cargue los datos en el ciclo correcto en aquellos registros a los que les ocurre este problema. Al haber también registros que no sufren este problema, los datos no viajarán en el ciclo que deberían.

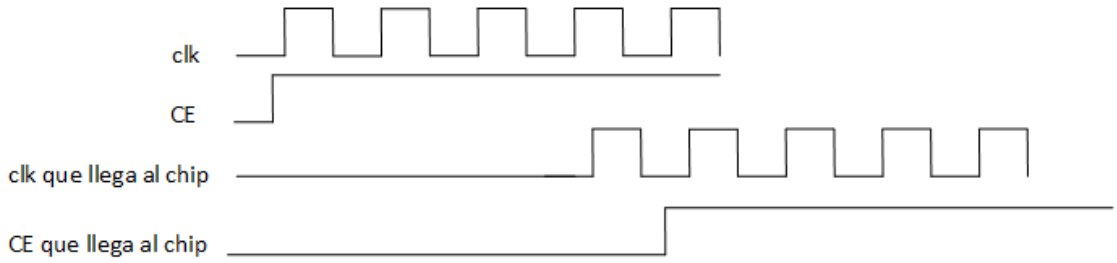

Figura 51 – Cronograma problemas con skew.

Para arreglar este problema se deberían haber añadido restricciones de temporización para no permitir este tipo de comportamiento. Por falta de tiempo no se realizó esta solución. La forma en que se arregló este problema fue añadiendo estas restricciones a la propia simulación, es decir, sobre la simulación no existen estos problemas, pero sí existen sobre la placa.

4.4.2. Simulación de la configuración con el módulo DS2 permutado

Para lanzar la simulación de la segunda configuración explicada en la Sección 4.3.2. Configuración con el módulo DS2 permutado se creó un script (*postPR_permuting_ds2*) en el que en primera instancia se crean dos directorios, uno para la entidad *trapezoidal_noc* y otro

para la entidad tb_*trapezoidal_noc* dentro del directorio *work/compilation*. A continuación se mapean las librerías *trapezoidal_noc* y *tb_trapezoidal_noc* en sus respectivos directorios con el comando *vmap*.

Después de esto se compilan lo archivos *vhd*, primero se compila el archivo *permuting_ds2.vhd* generado anteriormente sobre el directorio *trapezoidal_noc* para indicar que la DUT (*Design Under Test*) será este archivo. A continuación se compilan los archivos *vhd* necesarios para el test en el directorio *tb_trapezoidal_noc* (*read_file.vhd* y *tb_trapezoidal_noc.vhd*).

Para terminar se lanza la simulación con el siguiente comando:

```
vsim -sdftyp
/tb_trapezoidal_noc/dut=T:/trapezoidal_noc/sim/permuting_ds2/permuting_ds2.sd
f tb_trapezoidal_noc.tb_trapezoidal_noc -do wave.do
```

En este comando se indica que la DUT utilizada en esta simulación es el archivo *permuting.sdf*, testeando así el funcionamiento de esta entidad generada con el script explicado en la sección 4.4. Simulación de las configuraciones.

El test que se lanza en esta sección es el mismo que se ha utilizado durante todo el proyecto, salvo que la DUT es la configuración inicial (*permuting_ds2*) tal y como se observa en la siguiente figura:

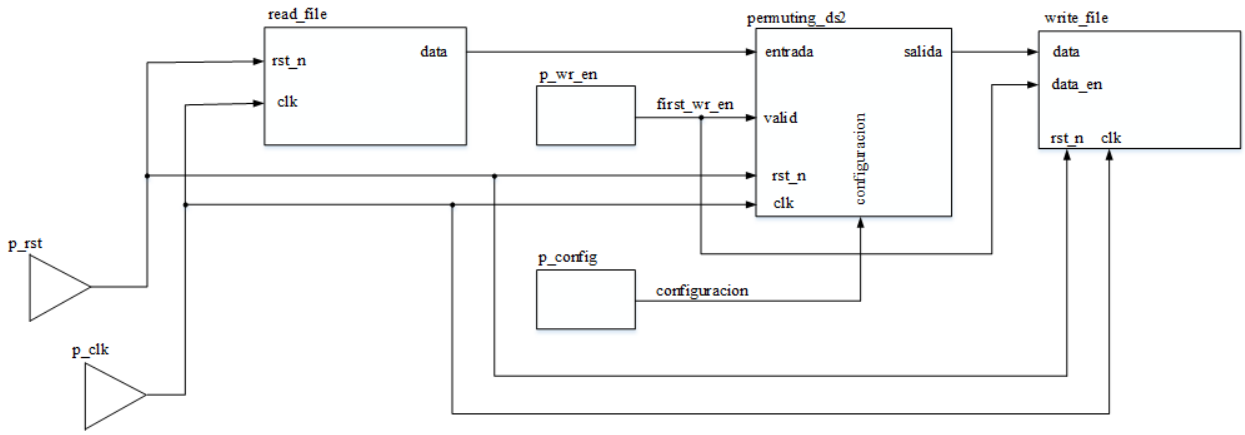

Figura 52 - Implementación tb_trapezoidal_noc con el módulo permuting_ds2.

Tras lanzar este comando, se arranca la herramienta de simulación *Questa* mostrando la forma de onda del comportamiento del diseño al que se le ha realizado el test. Gracias al módulo *write_file* del testbench explicado en la sección 2.3. Simulación se genera un fichero con las salidas generadas por esta entidad.

4.4.3. Comparación de las simulaciones

Por último para comprobar que ambas configuraciones funcionaban de la misma manera se creó un script de *Matlab* (*print_stimuli.m*) para representar los datos de entrada y de salida al sistema como una gráfica, tal y como ya se hizo en secciones anteriores. En esta sección lo importante es comprobar que las salidas de ambas configuraciones son iguales, ya que las

entradas inyectadas en ellas son exactamente las mismas. Para esto se generaron las gráficas de ambas señales (véanse la Figura 53 y la Figura 54). Y tal y como se observa en las figuras la altura de ambas toma un valor cercano a $14x10^{14}$ y el crecimiento de la función empieza en un valor cercano a 40 y decrece cerca del valor 225.

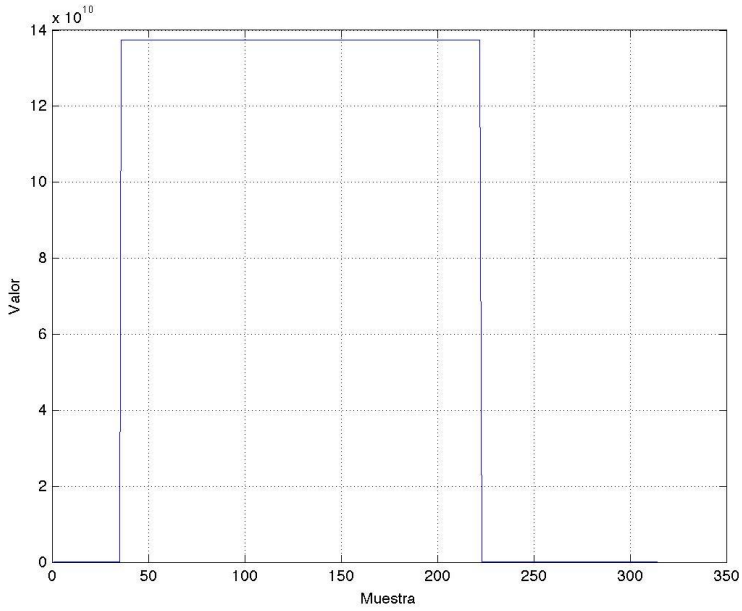

Figura 53 - Salida initial_config

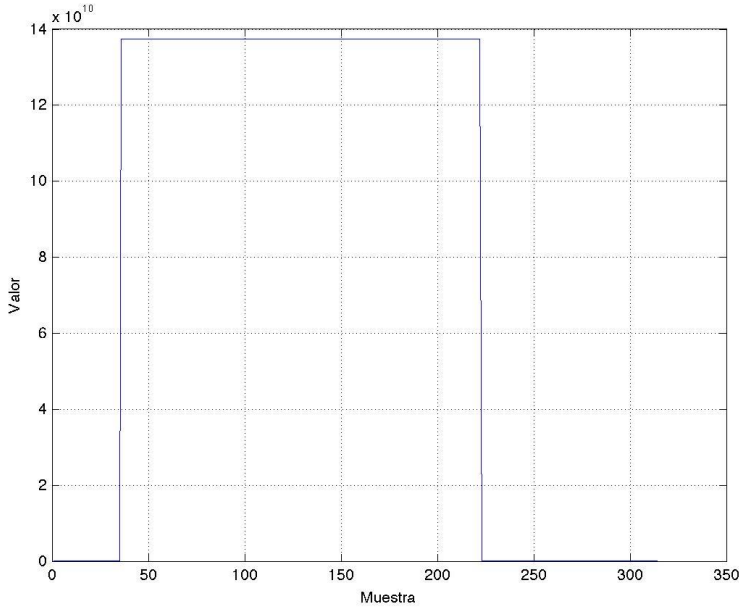

Figura 54 - Salida permuting_ds2

Por último, para comprobar que estas salidas son las correctas, y que por tanto, ambas configuraciones, además de funcionar igual, funcionan correctamente, se comprobaron estas

gráficas con la gráfica de la salida del conformador trapezoidal (Figura 55), observándose que los valores que toman las gráficas son los mismos.

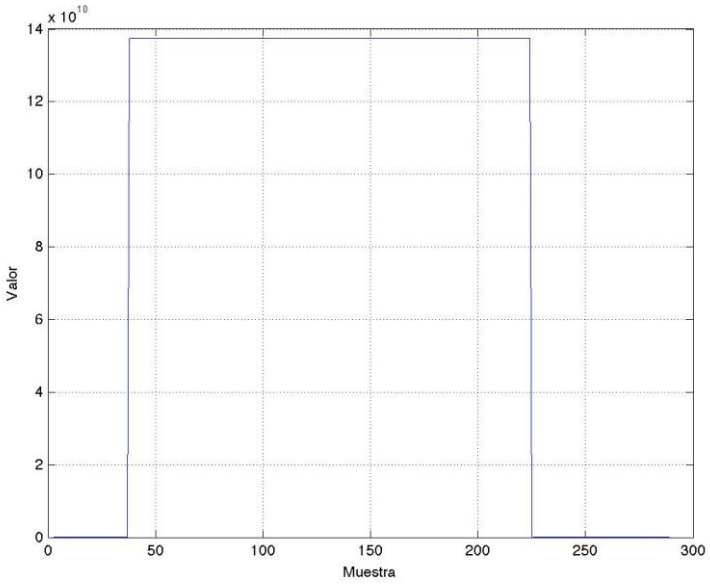

Figura 55 - Salida conformador trapezoidal

Capítulo 5

Conclusiones y futuras líneas de trabajo

5.1. Conclusiones

Con este proyecto se pretendía dotar de tolerancia a fallos a un conformador trapezoidal. Este tipo de sistemas suelen ser utilizados para aplicaciones como satélites o aceleradores de partículas. En este ámbito los fallos son muy probables, ya que la cantidad de partículas que pueden causar problemas físicos en la placa es más abundante. Debido a esta alta probabilidad de fallo se decidió dotar de tolerancia a fallos a este sistema.

Se pretendía dotar de la propiedad de reconfiguración dinámica parcial, es decir, poder reubicar los componentes que se encontrasen en zonas de la placa defectuosas a zonas que no tuviesen ningún fallo. Consiguiendo que el diseño funcionase a pesar de tener errores físicos.

Para conseguir esta reubicación hubo que cambiar las conexiones punto a punto mediante las que se comunicaban los componentes del conformador trapezoidal por una comunicación compartida. Hubo que cambiar estas conexiones ya que al reubicar un componente que estuviese conectado punto a punto con otro, esta conexión se perdía, y por tanto el sistema dejaba de funcionar.

Es por esto que se creó una entidad de comunicación a la que se conectarían todos los componentes del conformador trapezoidal. Esta entidad se creó con dos buses, uno para transmitir los datos y otro para indicar hacia quién iba dirigidos estos datos. Para conseguir conectar los componentes del conformador trapezoidal a estos buses se crearon unas entidades (esclavos). Con el fin de evitar las colisiones se implementaron dos soluciones: por un lado se le dio un identificador a cada componente (nunca habrá dos componentes con el mismo identificador), y por otro lado se añadió una entidad que arbitraba el uso al bus (*master*).

Esta entidad que arbitraba el uso del bus seguía una topología *token-ring*. Se comprobaban uno a uno los componentes que han solicitado el bus mediante una topología en anillo empezando por el identificador por valor 0. Cuando tocaba analizar la petición de un componente y éste había solicitado el bus, se le concedía.

Con la creación de esta entidad de comunicación se eliminaron las conexiones punto a punto entre los componentes del conformador trapezoidal. De esta forma se hizo posible la

reubicación de los componentes del conformador trapezoidal para dotarle de tolerancia a fallos.

Una vez se consiguió el conformador trapezoidal conectado por una entidad de comunicación se pasó a probar si era posible la reubicación de los distintos componentes. Para comprobarlo se utilizó la herramienta PlanAhead de Xilinx que permite, entre otras cosas, colocar componentes sintetizados (archivos *ngc*) en las zonas de la placa que se desee.

Para continuar, se crearon dos configuraciones sobre el diseño del conformador trapezoidal conectado por la entidad de comunicación. Estas dos configuraciones diferían en la colocación de un componente del conformador trapezoidal (DS2).

Para terminar, se realizaron simulaciones funcionales sobre estas dos configuraciones y se comprobó que:

1. Los resultados de ambas eran idénticos. Esto significa que la reubicación de los componentes no altera el comportamiento del sistema.
2. Los resultados de estas configuraciones eran idénticos a los resultados del conformador trapezoidal inicial. Esto quiere decir que los datos generados por ambas configuraciones eran correctos.

En definitiva, gracias a estas simulaciones funcionales, se pudo demostrar que el funcionamiento del conformador trapezoidal al que se le atribuyó la propiedad de la reconfiguración dinámica parcial funcionaba correctamente.

5.2. Futuras líneas de trabajo

En este capítulo se van a abordar posibles futuras líneas de trabajo tal y como son desenlazar los puertos de entrada/salida de los módulos reconfigurables, añadir un módulo de ethernet para transmitir las entradas, añadir una scan chain para detectar fallos y, por último, añadir una entidad que cuando se produzca una degeneración del sensor encontrase los mejores parámetros para regenerar el sistema.

Debido a que a los componentes que estén atados a puertos de entrada/salida no se les puede atribuir la propiedad de ser reconfigurables la primera línea de trabajo a realizar sería eliminar estos puertos de los componentes del conformador. Las conexiones de estos puertos eliminados pasarían a estar en la entidad de comunicación, siendo esta la que comunicase estos puertos con sus respectivos componentes. Así todos los componentes del conformador trapezoidal podrían ser reubicados en el *layout* de la FPGA.

Con el fin de dar inyectar los datos desde un equipo se optó por implementar un módulo de Ethernet. Consiguiendo así una mayor facilidad para inyectar nuevos datos al conformador trapezoidal.

También se estudió la posibilidad de añadir una cadena de *scan* (*scan-chain*) al diseño del conformador trapezoidal tolerante a fallos. Esta técnica se utiliza para detectar errores en el

funcionamiento de un determinado diseño utilizando los registros en sus caminos. En este caso se utilizaría para detectar los posibles errores en las etapas del conformador en tiempo de ejecución y así poder ejecutar el cambio de configuración en tiempo de ejecución.

La última modificación que se pensó fue añadir una entidad que cuando se produzca una degeneración del sensor encontrase los mejores parámetros para regenerar el sistema. La idea que surgió fue incluir un algoritmo genético. Un algoritmo genético es un modelo de una máquina de aprendizaje que deriva su comportamiento a partir de una metáfora de los procesos de la evolución en la naturaleza. Se realiza mediante la creación de una población de individuos representados por los cromosomas, en esencia, un conjunto de cadenas de caracteres que son análogos a los cromosomas de base-4 que se ven en nuestro propio ADN. Los individuos de la población pasan a continuación por un proceso de evolución [25].

Capítulo 6

Conclusions

This project aimed to provide fault tolerance to a trapezoidal shaper. Such systems are widely used for applications such as satellites and particle accelerators. Failures in this area are very likely, since the number of particles that can cause physical problems in the chip is heavier. Because of this high failure probability was decided to provide fault tolerance to this system.

It was intended to provide the property of partial dynamic reconfiguration, ie, to relocate the components which were at the defective chip areas to areas that did not have any fault. Getting the design would work in spite of physical errors.

To achieve this relocation had to change the point-to-point by which the components of the trapezoidal shaper communicated by a shared communication. It had to be changed because when a component that was connected with another component point to point was relocated, this connection is lost, and therefore the system stopped working.

This is why communication entity to which all components of the trapezoidal shaper would connect was created. This entity was created with two buses, one for transmit data and one to indicate who was directed to this data. To get connected components forming trapezoidal these buses some entities (slaves) were created. To avoid collisions two solutions were implemented: one was given an identifier for each component (never be two components with the same identifier), and secondly an entity arbitrating the use bus was added (master).

This entity arbitrated bus usage following a token-ring topology. Was checked one by one the components that have requested the bus using a ring topology by starting with the identifier 0. When played analyze the request of a component and it requested the bus, was granted.

With the creation of this entity communication the point to point connections between components of the trapezoidal shaper is removed. This was made possible relocation of components to give it trapezoidal shaper fault tolerance.

Once the trapezoidal shaper connected by a communication entity got passed test whether the relocation of the various components was possible. To check it the Xilinx PlanAhead tool that allows, among other things, placing synthesized components (ngc files) in areas of the plate that you want used.

To continue, two configurations on the design of trapezoidal shaper connected by communication entity was created. These two configurations differ in the placement of a component forming trapezoidal (DS2).

Finally, functional simulations on these two configurations were performed and it was found that:

1. Both results were identical. This means that the relocation of the components does not alter the behavior of the system.
2. Results of these configurations were identical to the results of the initial trapezoidal shaping. This means that the data generated by both configurations were correct.

In short, thanks to these functional simulations, it could be demonstrated that the operation of forming trapezoidal that is attributed the property of partial dynamic reconfiguration work properly.

Apéndices

Apéndice A - trapezoidal.xst

```
set -tmpdir "T:\trapezoidal_noc\impl\trapezoidal_noc\xst\projnav.tmp"
set -xsthdpdir "xst"
run
-ifn trapezoidal.prj
-ofn trapezoidal
-ofmt NGC
-p xc6vlx240t-1-ff1156
-top trapezoidal
-opt_mode Speed
-opt_level 1
-power NO
-iuc NO
-keep_hierarchy Yes
-netlist_hierarchy As_Optimized
-rtlview Yes
-glob_opt AllClockNets
-read_cores YES
-sd {"T:/trapezoidal_noc/impl/ip"}
-write_timing_constraints NO
-cross_clock_analysis NO
-hierarchy_separator /
-bus_delimiter <>
-case Maintain
-slice_utilization_ratio 100
-bram_utilization_ratio 100
-dsp_utilization_ratio 100
-lc Auto
-uc T:\trapezoidal_noc/impl/trapezoidal/trapezoidal.xcf
-reduce_control_sets Auto
-fsm_extract YES -fsm_encoding Auto
-safe_implementation No
-fsm_style LUT
-ram_extract Yes
-ram_style Auto
-rom_extract Yes
-shreg_extract YES
-rom_style Auto
-auto_bram_packing NO
-resource_sharing YES
-async_to_sync NO
-shreg_min_size 2
-use_dsp48 Auto
-iobuf YES
# Limitar el maximo fanout
-max_fanout 100
-bufg 32
-register_duplication YES
-register_balancing No
-optimize_primitives NO
```

```
-use_clock_enable Auto
-use_sync_set Auto
-use_sync_reset Auto
-iob Auto
-equivalent_register_removal YES
-slice_utilization_ratio_maxmargin
```

Apéndice B - trapezoidal.prj

```
# IP CORE FIFO MEM
vhdl work        "T:/trapezoidal_noc/impl/ip/slv_fifo.vhd"

vhdl trapezoidal
"T:/trapezoidal_noc/src/design/trapezoidal/hdl/constants.vhd"
vhdl trapezoidal "T:/trapezoidal_noc/src/design/trapezoidal/hdl/reg.vhd"
vhdl trapezoidal "T:/trapezoidal_noc/src/design/trapezoidal/hdl/dffe.vhd"
vhdl trapezoidal
"T:/trapezoidal_noc/src/design/trapezoidal/hdl/parameters.vhd"
vhdl trapezoidal "T:/trapezoidal_noc/src/design/trapezoidal/hdl/dff.vhd"
vhdl trapezoidal "T:/trapezoidal_noc/src/design/trapezoidal/hdl/adder.vhd"
vhdl trapezoidal
"T:/trapezoidal_noc/src/design/trapezoidal/hdl/accumulator.vhd"
vhdl trapezoidal "T:/trapezoidal_noc/src/design/trapezoidal/hdl/reg.vhd"
vhdl trapezoidal
"T:/trapezoidal_noc/src/design/trapezoidal/hdl/subtraction.vhd"
vhdl trapezoidal "T:/trapezoidal_noc/src/design/trapezoidal/hdl/delay.vhd"
vhdl trapezoidal "T:/trapezoidal_noc/src/design/trapezoidal/hdl/ds.vhd"
vhdl trapezoidal "T:/trapezoidal_noc/src/design/trapezoidal/hdl/reg.vhd"
vhdl trapezoidal "T:/trapezoidal_noc/src/design/trapezoidal/hdl/adder.vhd"
vhdl trapezoidal
"T:/trapezoidal_noc/src/design/trapezoidal/hdl/sign_extender.vhd"
vhdl trapezoidal
"T:/trapezoidal_noc/src/design/trapezoidal/hdl/multiplier.vhd"
vhdl trapezoidal
"T:/trapezoidal_noc/src/design/trapezoidal/hdl/accumulator.vhd"
vhdl trapezoidal "T:/trapezoidal_noc/src/design/trapezoidal/hdl/hpd.vhd"

# Top level modules
vhdl trapezoidal
"T:/trapezoidal_noc/src/design/trapezoidal/hdl/trapezoidal.vhd"
```

Apéndice C - trapezoidal_noc_simple.xst

```
set -tmpdir "T:\trapezoidal_noc\impl\trapezoidal_noc\xst_simple\projnav.tmp"
set -xsthdpdir "xst"
run
-ifn trapezoidal_noc_simple.prj
-ofn trapezoidal_noc
-ofmt NGC
-p xc6vlx240t-1-ff1156
-top trapezoidal
-opt_mode Speed
```

```
-opt_level 1
-power NO
-iuc NO
-keep_hierarchy Yes
-netlist_hierarchy As_Optimized
-rtlview Yes
-glob_opt AllClockNets
-read_cores YES
-sd {"C:/Documentos/Devel/trapezoidal_noc/impl/ip"  }
-write_timing_constraints NO
-cross_clock_analysis NO
-hierarchy_separator /
-bus_delimiter <>
-case Maintain
-slice_utilization_ratio 100
-bram_utilization_ratio 100
-dsp_utilization_ratio 100
-lc Auto
-uc T:\trapezoidal_noc/impl/trapezoidal_noc/trapezoidal_noc_simple.xcf
-reduce_control_sets Auto
-fsm_extract YES -fsm_encoding Auto
-safe_implementation No
-fsm_style LUT
-ram_extract Yes
-ram_style Auto
-rom_extract Yes
-shreg_extract YES
-rom_style Auto
-auto_bram_packing NO
-resource_sharing YES
-async_to_sync NO
-shreg_min_size 2
-use_dsp48 Auto
-iobuf YES
# Limitar el maximo fanout
-max_fanout 100
-bufg 32
-register_duplication YES
-register_balancing No
-optimize_primitives NO
-use_clock_enable Auto
-use_sync_set Auto
-use_sync_reset Auto
-iob Auto
-equivalent_register_removal YES
-slice_utilization_ratio_maxmargin 5
```

Apéndice D - trapezoidal_noc_simple.prj

```
# IP CORE FIFO MEM
vhdl work        "T:/trapezoidal_noc/impl/ip/slv_fifo.vhd"

# STATIC LOGIC
# |-- bus_fabric modules
```

```
vhdl trapezoidal
"T:/trapezoidal_noc/src/design/trapezoidal/hdl/constants.vhd"
vhdl trapezoidal "T:/trapezoidal_noc/src/design/trapezoidal/hdl/reg.vhd"
vhdl trapezoidal "T:/trapezoidal_noc/src/design/trapezoidal/hdl/dffe.vhd"
vhdl bus_fabric  "T:/trapezoidal_noc/src/design/bus_fabric/hdl/tbuf.vhd"
vhdl bus_fabric
"T:/trapezoidal_noc/src/design/bus_fabric/hdl/bus_reg_high.vhd"
vhdl bus_fabric  "T:/trapezoidal_noc/src/design/bus_fabric/hdl/bus_reg.vhd"
vhdl bus_fabric  "T:/trapezoidal_noc/src/design/bus_fabric/hdl/slv_tx.vhd"
vhdl bus_fabric  "T:/trapezoidal_noc/src/design/bus_fabric/hdl/slv_rx.vhd"
vhdl bus_fabric
"T:/trapezoidal_noc/src/design/bus_fabric/hdl/configuration_mux.vhd"
vhdl bus_fabric  "T:/trapezoidal_noc/src/design/bus_fabric/hdl/slv.vhd"
vhdl bus_fabric
"T:/trapezoidal_noc/src/design/bus_fabric/hdl/parameters.vhd"
vhdl bus_fabric  "T:/trapezoidal_noc/src/design/bus_fabric/hdl/master.vhd"
vhdl trapezoidal
"T:/trapezoidal_noc/src/design/bus_fabric/hdl/bus_fabric.vhd"
vhdl trapezoidal
"T:/trapezoidal_noc/src/design/trapezoidal/hdl/ds_wrapper.vhd"
vhdl trapezoidal "T:/trapezoidal_noc/src/design/trapezoidal/hdl/ds.vhd"
vhdl trapezoidal "T:/trapezoidal_noc/src/design/trapezoidal/hdl/valid2wr.vhd"
vhdl trapezoidal "T:/trapezoidal_noc/src/design/trapezoidal/hdl/delay.vhd"
vhdl trapezoidal
"T:/trapezoidal_noc/src/design/trapezoidal/hdl/subtraction.vhd"
vhdl trapezoidal "T:/trapezoidal_noc/src/design/trapezoidal/hdl/dff.vhd"
vhdl trapezoidal
"T:/trapezoidal_noc/src/design/trapezoidal/hdl/hpd_wrapper.vhd"
vhdl trapezoidal "T:/trapezoidal_noc/src/design/trapezoidal/hdl/hpd.vhd"
vhdl trapezoidal
"T:/trapezoidal_noc/src/design/trapezoidal/hdl/multiplier.vhd"
vhdl trapezoidal
"T:/trapezoidal_noc/src/design/trapezoidal/hdl/sign_extender.vhd"
vhdl trapezoidal
"T:/trapezoidal_noc/src/design/trapezoidal/hdl/accumulator.vhd"
vhdl trapezoidal "T:/trapezoidal_noc/src/design/trapezoidal/hdl/adder.vhd"
vhdl trapezoidal
"T:/trapezoidal_noc/src/design/trapezoidal/hdl/acc_wrapper.vhd"
vhdl trapezoidal
"T:/trapezoidal_noc/src/design/trapezoidal_noc/hdl/black_box.vhd"

# STATIC LOGIC
# |-- top_level modules
vhdl trapezoidal_noc
"T:/trapezoidal_noc/src/design/trapezoidal_noc/hdl/trapezoidal.vhd"

# DYNAMIC LOGIC NOT DECLARED (BLACK-BOXES)
```

Apéndice E - synth_trap_noc_reconfig

```
REM ------------------------------------------------------------------------
---
REM -- black_box synthesis
REM ------------------------------------------------------------------------
---
```

```
REM -- Creacion de los directorios temporales
REM -------------------------------------------------------------------------
---
cd "T:/trapezoidal_noc/impl/black_box"
if exist xst goto nodir
   mkdir "xst"
   mkdir "xst/projnav.tmp"
:nodir

xst -intstyle ise -filter
"T:/trapezoidal_noc/tools/ise/black_box/iseconfig/filter.filter" -ifn
"T:/trapezoidal_noc/impl/black_box/black_box.xst" -ofn
"T:/trapezoidal_noc/impl/black_box/black_box.syr"

cd "T:/trapezoidal_noc/tools/ise"

REM -------------------------------------------------------------------------
---
REM -- ds_wrapper synthesis
REM -- Son necesarias dos sintesis porque cada instancia de la entidad
REM    ds_wrapper tiene unos genericos distintos
REM -------------------------------------------------------------------------
---
REM -- Creacion de los directorios temporales
REM -------------------------------------------------------------------------
---
cd "T:/trapezoidal_noc/impl/ds_wrapper"
if exist xst goto nodir
   mkdir "xst"
   mkdir "xst/projnav.tmp"
:nodir

REM -------------------------------------------------------------------------
---
REM -- Sintesis del DS1
REM -------------------------------------------------------------------------
---
xst -intstyle ise -filter
"T:/trapezoidal_noc/tools/ise/ds_wrapper/iseconfig/filter.filter" -ifn
"T:/trapezoidal_noc/impl/ds_wrapper/ds1_wrapper.xst" -ofn
"T:/trapezoidal_noc/impl/ds_wrapper/ds1_wrapper.syr"

REM -------------------------------------------------------------------------
---
REM -- Sintesis del DS2
REM -------------------------------------------------------------------------
---
xst -intstyle ise -filter
"T:/trapezoidal_noc/tools/ise/ds_wrapper/iseconfig/filter.filter" -ifn
"T:/trapezoidal_noc/impl/ds_wrapper/ds2_wrapper.xst" -ofn
"T:/trapezoidal_noc/impl/ds_wrapper/ds2_wrapper.syr"


cd "T:/trapezoidal_noc/tools/ise"

REM -------------------------------------------------------------------------
---
REM -- hpd_wrapper synthesis
```

```
REM ------------------------------------------------------------------------
---
REM -- Creacion de los directorios temporales
REM ------------------------------------------------------------------------
---
cd "T:/trapezoidal_noc/impl/hpd_wrapper"
if exist xst goto nodir
   mkdir "xst"
   mkdir "xst/projnav.tmp"
:nodir

xst -intstyle ise -filter
"T:/trapezoidal_noc/tools/ise/hpd_wrapper/iseconfig/filter.filter" -ifn
"T:/trapezoidal_noc/impl/hpd_wrapper/hpd_wrapper.xst" -ofn
"T:/trapezoidal_noc/impl/hpd_wrapper/hpd_wrapper.syr"

cd "T:/trapezoidal_noc/tools/ise"

REM ------------------------------------------------------------------------
---
REM -- acc_wrapper synthesis
REM ------------------------------------------------------------------------
---
REM -- Creacion de los directorios temporales
REM ------------------------------------------------------------------------
---
cd "T:/trapezoidal_noc/impl/acc_wrapper"
if exist xst goto nodir
   mkdir "xst"
   mkdir "xst/projnav.tmp"
:nodir

xst -intstyle ise -filter
"T:/trapezoidal_noc/tools/ise/acc_wrapper/iseconfig/filter.filter" -ifn
"T:/trapezoidal_noc/impl/acc_wrapper/acc_wrapper.xst" -ofn
"T:/trapezoidal_noc/impl/acc_wrapper/acc_wrapper.syr"

cd "T:/trapezoidal_noc/tools/ise"

REM ------------------------------------------------------------------------
---
REM -- bus_fabric synthesis
REM ------------------------------------------------------------------------
---
REM -- Creacion de los directorios temporales
REM ------------------------------------------------------------------------
---
cd "T:/trapezoidal_noc/impl/bus_fabric"
if exist xst goto nodir
   mkdir "xst"
   mkdir "xst/projnav.tmp"
:nodir

xst -intstyle ise -filter
"T:/trapezoidal_noc/tools/ise/bus_fabric/iseconfig/filter.filter" -ifn
"T:/trapezoidal_noc/impl/bus_fabric/bus_fabric.xst" -ofn
"T:/trapezoidal_noc/impl/bus_fabric/bus_fabric.syr"
```

```
cd "T:/trapezoidal_noc/tools/ise"

REM ------------------------------------------------------------------------
---
REM -- top_level synthesis
REM ------------------------------------------------------------------------
---
REM -- Creacion de los directorios temporales
REM ------------------------------------------------------------------------
---
cd "T:/trapezoidal_noc/impl/trapezoidal_noc"
if exist xst goto nodir
   mkdir "xst"
   mkdir "xst/projnav.tmp"
:nodir

xst -intstyle ise -filter
"T:/trapezoidal_noc/tools/ise/trapezoidal_noc/iseconfig/filter.filter" -ifn
"T:/trapezoidal_noc/impl/trapezoidal_noc/trapezoidal_noc.xst" -ofn
"T:/trapezoidal_noc/impl/trapezoidal_noc/trapezoidal_noc.syr"

cd "T:/trapezoidal_noc/tools/ise"
```

Apéndice F - planahead.tcl

```
#------------------------------------------------------------------------
--
# PLANAHEAD Script
# ----------------
# Este script crea el proyecto y genera los archivos para las dos
configuraciones
#------------------------------------------------------------------------
--

#------------------------------------------------------------------------
--
# Creacion del proyecto
#------------------------------------------------------------------------
--
create_project planahead T:/trapezoidal_noc/planahead -part xc6vlx240tff1156-
1
set_property   board ml605 [current_project]
set_property design_mode GateLvl [current_fileset]
add_files -norecurse
{T:/trapezoidal_noc/impl/trapezoidal_noc/trapezoidal_noc.ngc
                      T:/trapezoidal_noc/impl/ip/slv_fifo.ngc}

#------------------------------------------------------------------------
--
# Creacion del archivo de constraints
#------------------------------------------------------------------------
--
create_fileset -constrset init_constraints
add_files -fileset init_constraints -norecurse
{T:/trapezoidal_noc/tools/planahead/trapezoidal.ucf}
```

```
set_property target_constrs_file
{T:/trapezoidal_noc/tools/planahead/trapezoidal.ucf} [get_filesets
init_constraints]
set_property constrset init_constraints [get_runs impl_1]
delete_fileset constrs_1

#-----------------------------------------------------------------------
--
# Definicion de las propiedades del proyecto
#-----------------------------------------------------------------------
--
set_property name                initial_config  [current_run]
set_property is_partial_reconfig true            [current_project]
set_property target_simulator    ModelSim        [current_project]
set_property target_language     VHDL            [current_project]
link_design -name netlist_initial

#-----------------------------------------------------------------------
--
# Creando modulos reconfigurables
#-----------------------------------------------------------------------
--
create_reconfig_module -name acc_wrapper -cell i_acc_wrapper
set_property edif_top_file
T:/trapezoidal_noc/impl/acc_wrapper/acc_wrapper.ngc [get_filesets
i_acc_wrapper#acc_wrapper]
save_design
load_reconfig_modules -reconfig_modules i_acc_wrapper:acc_wrapper

create_reconfig_module -name ds1_wrapper -cell i_ds1_wrapper
set_property edif_top_file T:/trapezoidal_noc/impl/ds_wrapper/ds1_wrapper.ngc
[get_filesets i_ds1_wrapper#ds1_wrapper]
save_design
load_reconfig_modules -reconfig_modules i_ds1_wrapper:ds1_wrapper


create_reconfig_module -name ds2_wrapper -cell i_ds2_wrapper
set_property edif_top_file T:/trapezoidal_noc/impl/ds_wrapper/ds2_wrapper.ngc
[get_filesets i_ds2_wrapper#ds2_wrapper]
save_design
load_reconfig_modules -reconfig_modules i_ds2_wrapper:ds2_wrapper


create_reconfig_module -name hpd_wrapper -cell i_hpd_wrapper
set_property edif_top_file
T:/trapezoidal_noc/impl/hpd_wrapper/hpd_wrapper.ngc [get_filesets
i_hpd_wrapper#hpd_wrapper]
save_design
load_reconfig_modules -reconfig_modules i_hpd_wrapper:hpd_wrapper

#-----------------------------------------------------------------------
--
# Creando black-boxes
#-----------------------------------------------------------------------
--
create_reconfig_module -name bb -cell i_bb_1
set_property edif_top_file T:/trapezoidal_noc/impl/black_box/black_box.ngc
[get_filesets i_bb_1#bb]
```

```
save_design
load_reconfig_modules -reconfig_modules i_bb_1:bb


create_reconfig_module -name bb -cell i_bb_2
set_property edif_top_file T:/trapezoidal_noc/impl/black_box/black_box.ngc
[get_filesets i_bb_2#bb]
save_design
load_reconfig_modules -reconfig_modules i_bb_2:bb

create_reconfig_module -name ds2_wrapper -cell i_bb_2
set_property edif_top_file T:/trapezoidal_noc/impl/ds_wrapper/ds2_wrapper.ngc
[get_filesets i_bb_2#ds2_wrapper]


#-----------------------------------------------------------------------------
--
# Con el siguiente comando se define el modulo reconfigurable ds2_wrapper
Comoro
# el modulo activo
#-----------------------------------------------------------------------------
--
# load_reconfig_modules -reconfig_modules i_bb_2:ds2_wrapper
# save_design

#-----------------------------------------------------------------------------
--
# DRC
#-----------------------------------------------------------------------------
--
report_drc -name drc_init_config

#-----------------------------------------------------------------------------
--
# Run implementation
#-----------------------------------------------------------------------------
--
launch_runs initial_config

#-----------------------------------------------------------------------------
--
# Promote implementation
#-----------------------------------------------------------------------------
--
promote_run -run initial_config -partition_names {trapezoidal i_acc_wrapper
i_ds1_wrapper i_ds2_wrapper i_hpd_wrapper i_bb_1 i_bb_2}

#-----------------------------------------------------------------------------
--
# Creamos una nueva implementacion en la que el bloque bb_2 contendra DS_2 y
else
# bloque ds_2 se convierte en una black-box.
#-----------------------------------------------------------------------------
--
config_partition -run permuting_ds2 -import -import_dir
T:/trapezoidal_noc/planahead/planahead.promote/Xinitial_config -preservation
routing
```

```
config_partition -run permuting_ds2 -cell i_acc_wrapper -reconfig_module
acc_wrapper -import -import_dir
T:/trapezoidal_noc/planahead/planahead.promote/Xinitial_config
config_partition -run permuting_ds2 -cell i_ds1_wrapper -reconfig_module
ds1_wrapper -import -import_dir
T:/trapezoidal_noc/planahead/planahead.promote/Xinitial_config
config_partition -run permuting_ds2 -cell i_ds2_wrapper -reconfig_module bb -
implement
config_partition -run permuting_ds2 -cell i_hpd_wrapper -reconfig_module
hpd_wrapper -import -import_dir
T:/trapezoidal_noc/planahead/planahead.promote/Xinitial_config
config_partition -run permuting_ds2 -cell i_bb_1 -reconfig_module bb -import
-import_dir T:/trapezoidal_noc/planahead/planahead.promote/Xinitial_config
config_partition -run permuting_ds2 -cell i_bb_2 -reconfig_module ds2_wrapper
-implement
create_run permuting_ds2 -flow {ISE 14} -strategy {ISE Defaults}
current_run [get_runs permuting_ds2]

launch_runs permuting_ds2

#-------------------------------------------------------------------------
--
# Por ultimo, se verifica que ambas configuraciones tienen la misma logica
estatica
# y los mismos bits de particion
#-------------------------------------------------------------------------
--
verify_config -runs {  initial_config permuting_ds2 } -file
T:/trapezoidal_noc/planahead/pr_verify.log -verbose
```

Bibliografía

[1] G. F. Knoll, Radiation Detecton and Measurement, Fourth Edition ed., J. W. &. Sons, Ed., 2010, p. 830.

[2] Ó. Garnica, J. Lanchares, J. L. Risco Martín, J. Hignacio Hidalgo, J. M. Colmenar y A. Cuesta, «Shapers, Real Time Evolvable Hardware for Optimal Reconfiguration of Cusp-Like Pulse».

[3] H. Kirrmann, «Fault Tolerant Computing,» 2005.

[4] Xilinx, *Virtex-6 FPGA - User Guide,* 2013.

[5] Xilinx, *Partial Reconfiguration User Guide,* 2010.

[6] Xilinx, *Partial Reconfiguration of Xilinx FPGAs Using ISE Design Suite,* 2012.

[7] J. Follows, Token Ring Solutions, Reed Books IBM.

[8] R. Moreno Vozmediano, R. Santiago Montero y J. C. Fabero Jiménez, «https://www.fdi.ucm.es/profesor/rubensm/Redes/Trasparencias/Tema%204.pdf,» 2012. [En línea].

[9] Xilinx, *XPS IIC Bus Interface,* 2011.

[10] M. B. Stensgaard y J. Sparsø, «ReNoC: A Network-on-Chip Architecture with Reconfigurable Topology,» 2008.

[11] J. H. Bahn, «http://gram.eng.uci.edu/comp.arch/lab/NoCOverview.htm,» [En línea].

[12] K. Asanovic y H. Seongmoo, «http://publications.csail.mit.edu/abstracts/abstracts05/heomoo2/heomoo2.html,» [En línea].

[13] Xilinx, «http://www.xilinx.com/products/boards-and-kits/EK-V6-ML605-G.htm,» [En línea].

[14] Xilinx, «http://forums.xilinx.com/t5/CPLDs/Speed-Grade/td-p/3052,» [En línea].

[15] Xilinx, «http://forums.xilinx.com/t5/7-Series-FPGAs/7-series-Package-FFG-vs-FBG/td-p/227449,» [En línea].

[16] Xilinx, «http://www.xilinx.com/support/quality/pb-free-rohs-compliant.html,» [En línea].

[17] Xilinx, Virtex-6 FPGA Packaging, 2011.

[18] Xilinx, Virtex-6 Family Overview, 2012.

[19] V. T. Jordanov, G. F. Knoll, A. C. Huber y J. A. Pantazis, «Digital techniques for real-time pulse shaping,» 1994.

[20] Xilinx, XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices, 2012.

[21] Xilinx, Command Line Tools User Guide, 2009.

[22] Xilinx, XST User Guide.

[23] R. K. Decker, D. Puperi y R. Leach, «Impact to Space Shuttle Vehicle Trajectory on Day of Launch from change in Low Frequency Winds».

[24] Xilinx, «Command Line Tools User,» 2009.

[25] C. M. University, «http://www.cs.cmu.edu/Groups/AI/html/faqs/ai/genetic/part2/faq-doc-2.html,» [En línea].