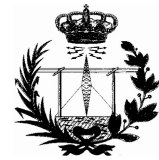
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Diseño y desarrollo del interleaver para el decodificador Turbo Código WiMax



AUTOR: Juan Valera Requena
DIRECTOR: Fco. Javier Garrigós Guerrero

Julio / 2007

Autor	Juan Valera Requena
E-mail del Autor	drdolor@gmail.com
Director	Fco. Javier Garrigós Guerrero
E-mail del Director	javier.garrigos@upct.es
Título del PFC	Diseño y desarrollo del interleaver para el decodificador Turbo Código WiMax
Descriptores	Codificación de canal, turbo código, códigos duo-binarios, interleaver.

Resúmen

El futuro de las redes inalámbricas de banda ancha pasa por la nueva tecnología WiMax, capaz de ofrecer grandes velocidades de acceso en un amplio radio de acción. En comunicaciones radio de altas prestaciones la codificación de canal y las técnicas de corrección de errores que se aplican son cruciales para un alto rendimiento de la red.

Los Turbo Códigos descubiertos en los años 90 se aproximan al límite teórico de Shannon de máxima capacidad de canal, y la modificación *no-binaria* hace aumentar las prestaciones de estos códigos. El mayor inconveniente de los Turbo Códigos es la complejidad que presenta el decodificador, debiéndose realizar en hardware para que éste no represente el cuello de botella del sistema.

La tecnología WiMax (IEEE 802.16) recoge como codificación opcional los Turbo Códigos, con lo que las prestaciones aumentarán.

Tal vez, la mayor dificultad de un decodificador Turbo Código sea la implementación de un interleaver de alta velocidad, pues siempre representan el factor limitante de los decodificadores por ser el elemento principal de paso intermedio de datos.

El presente proyecto versará sobre el desarrollo de un interleaver de alta velocidad, para lo cual se recurrirá a una arquitectura de decodificación paralelizada.

Titulación	Ingeniería de Telecomunicación
Intensificación	Sistemas y Redes de Telecomunicación
Departamento	Departamento de Electrónica, Tecnología de Computadores y Proyectos
Fecha de Presentación	Julio - 2007

“No pidas una carga ligera,
pide unas espaldas fuertes.”
Theodore Roosevelt

Agradecimientos

Aún arriesgando a caer en un tópico, lo he de hacer:

A mis padres, porque sí y porque no existe ningún no; y a mi hermana, porque ella lo vale y de verdad.

A los que han pasado estos años a mi lado, sufriendo a veces y disfrutando otras, pero siempre riendo: Raul, Mik (que no Kitt ni Tik), Fustel, Vicen y Ana (estos al final se casan, o si no, al tiempo), Lydia, Sandra, etc… La gente del Alberto Colao, pues en 5 años hay cabida para todo, buenos y malos momentos, pero siempre se recordaran los buenos: Lucas, Adri, Manolo "el calvo", Flaut, Charly, Ramallo, etc… ¡imposible nombrar a todos y todas! Creo que acabo antes si digo "*al Atletico Pico-Pala*".

Y como en los malos tiempos es cuando la gente se une… a la gente de clase: Manolo, Patxi (Alfonso), Sebas, Fran, Gonzalo, Chules, Ana, Mónica, Cristina, etc…

A ese *peasso* cuarteto que eramos *Delirium Tremens*: Vicen, Mik (anda mira, ¡si éstos dos repiten!) y Luis. Gracias, pues junto a vosotros y haciendo eso que llamamos música me he sentido más realizado que nunca… *ta-ca-ta-ca-plasssshhh...*

A Erasmo de Róterdam, que gracias a él he podido conocer a gente majísima en Torino: Tito Rober, Rafa, Imanol, Alex, Andrés, Michael, Carmen, Astrid, Andrea, Helena, *La Paca*, Alice, Giada, etc… Pero sobre todo, gracias Erasmo de Róterdam porque sín ti nunca podría haber conocido a *Mary*, gracias por aguantarme tanto y por lo que nos queda por aguantarnos ;)

A todo el personal de la UPCT, desde reprografía hasta el mismisimo Joan, y sin olvidarnos de *Casa Paco*, que todos sabemos que algún día será centro adscrito de la universidad. A Javier Garrigos, por llevar parte de este proyecto y aguantar mis prisas por querer hacerme mayor; a Tati y Virginia (*RELINT*) por lo eficiente de su trabajo con nosotros. Al Politecnico di Torino y al laboratorio VLSI del departamento de Electronica, al profesor Masera, Mario Nicola y Maurizio Martina.

A los amiguetes de La Manga, como no, Felix, Raul, Tony Javy, Alfredo… por estar ahí todos estos veranos de estudio. De aquí en adelante prometo salir más.

También se ha de agradecer a aquellos cientificos que paso a paso acabas tratandolos como si fueran de la familia: Fourier, Gauss, Moore, Euler, Riemann, Shannon, Laplace, Ampere, Faraday, Maxwell, Morse, Bell, Hertz, Marconi, Nyquist, Weber, Carson, Armstrong, Mainman, Kirchhoff, etc..

¡Y qué carajo! ¡A mi mismo! ¡Olé yo!

NOTA: Papá, buscate otra muletilla que decirme en mis ratos ociosos. Por cierto, Fourier y Shannon viene a cenar.

CAPÍTULO *PÁGINA*

1. INTRODUCCIÓN 1
1.1. CODIFICACIÓN DE CANAL. CONCEPTOS.................................. 1
1.1.1. ARQ (AUTOMATIC REPEAT REQUEST) 1
1.1.2. FEC (FORWARD ERROR CORRECTION).................................. 2
1.2. TURBO CÓDIGOS 2
1.3. TURBO CÓDIGOS DUO-BINARIOS...................................... 4
1.4. WIMAX 5
1.4.1. MODO OFDMA. CODIFICACIONES DE CANAL 6
1.5. OBJETIVO DEL PROYECTO.. 6
2. INTERLEAVER 7
2.1. INTRODUCCIÓN 7
2.1.1. ¿QUÉ ES UN *INTERLEAVER*? 7
2.1.2. INTERLEAVER TURBO CÓDIGO [IEEE 802.16E-2005] 7
2.1.3. IMPLEMENTACIÓN ELECTRÓNICA. PRIMERA APROXIMACIÓN. 9
2.2. EL INTERLEAVER PARALELO 10
2.2.1. PARALELIZANDO EL ALGORITMO 10
2.2.2. CASOS ESPECIALES NO PARALELIZABLES. DISCUSIÓN. 13
2.2.3. ACCESO A MEMORIA. COLISIONES. 16
2.3. COLISIONES: DISCUSIÓN Y SOLUCIÓN 19
2.3.1. CONSIDERACIONES PREVIAS 19
2.3.2. REVISIÓN BIBLIOGRÁFICA.. 20
2.3.3. SOLUCIONES POSIBLES 21
2.3.4. SOLUCIONES EN LECTURA. ESTUDIO Y VIABILIDAD 21
2.3.4.1. ARQUITECTURA TIBB.. 22
2.3.4.2. MECANISMO DE PARADA TOTAL 24
2.3.4.3. COMPARATIVA TIBB – PARADA TOTAL. SOLUCIÓN ADOPTADA......... 26
2.3.5. SOLUCIÓN EN ESCRITURA, TIBB. ESTUDIO Y VIABILIDAD: 27
3. IMPLEMENTACIÓN 29
3.1. INTERFAZ DEL INTERLEAVER... 29
3.1.1. COMPORTAMIENTO.. 29

3.1.2. INTERFAZ ENTRADA/SALIDA 30

3.2. FLUJO DE DISEÑO Y VALIDACIÓN. HERRAMIENTAS. 33

3.3. DISEÑO TOP/DOWN 34

3.4. DISEÑO DOWN/TOP 36

3.4.1. ALGORITMO INTERLAVER Y MEMORIA DE PARÁMETROS 36

3.4.1.1. MANIPULACIÓN DEL ALGORITMO 36

3.4.1.2. DISEÑO DE LA MEMORIA ROM. ENTIDAD '*VALUESN*' 43

3.4.1.3. ENTIDAD '*INTERLEAVERALGORITHM*' 44

3.4.2. GENERADOR DE ÍNDICES 45

3.4.2.1. ENTIDAD '*INDEXGENERATOR*' 45

3.4.2.2. TEST Y VALIDACIÓN 46

3.4.3. TRADUCTOR DE ÍNDICES 46

3.4.3.1. ALGORITMO DE TRADUCCIÓN 46

3.4.3.2. ENTIDAD '*ADDRESSTRANSLATE*' 48

3.4.4. GENERADOR DE DIRECCIONES 48

3.4.4.1. ENTIDAD '*ADDRESSGENERATOR*' 48

3.4.4.2. TEST Y VALIDACIÓN 49

3.4.5. MEMORIA LIFO DOBLE 49

3.4.5.1. ARQUITECTURA 50

3.4.5.2. CONDICIONES DE CONMUTACIÓN 51

3.4.5.3. ENTIDAD 'LIFOMULTIPLE' 51

3.4.6. CONMUTADOR DE LECTURA 52

3.4.6.1. ARQUITECTURA 52

3.4.6.2. ENTIDAD '*SWITCHMATRIX*' 53

3.4.6.3. ENTIDAD '*SWITCHREADMEMORY*' 54

3.4.6.4. TEST Y VALIDACIÓN 54

3.4.7. CONMUTADOR DE ESCRITURA 55

3.4.7.1. ARQUITECTURA 55

3.4.7.2. ENTIDAD '*SWITCHWRITEMEMORY*' 57

3.4.7.3. TEST Y VALIDACIÓN 57

3.4.8. UNIDAD DE CONTROL CENTRAL 58

3.4.8.1. CRONOGRAMA DE LECTURA 59

3.4.8.2. MAQUINA DE ESTADOS 59
3.4.8.3. ARQUITECTURTA 60
3.4.9. INTERLEAVER. CONEXIONADO DE ENTIDADES. 60
3.5. INTERLEAVER. TEST Y VALIDACIÓN. 62
4. RESULTADOS 63
4.1. RESULTADOS DE SÍNTESIS 63
4.2. RENDIMIENTO 63
4.2.1. LECTURA 63
4.2.2. ESCRITURA 64
5. CONCLUSIONES 65
6. REFERENCIAS 66

ANEXO I: FUNCIONES Y SCRIPTS MATLAB®

I. INTERLEAVERFUNCTION.M I
II. OPTIMAL_PARAM_REDUCED.M II
III. OPTIMAL_PARAM_CTC.M III
IV. RECURRENCEINTERLEAVERFUNCION.M IV
V. RECURRENCEINTERLEAVERFUNCION2.M V
VI. FILLBLANK VI
VII. MATRIXINVEST.M VII
VIII. COLLISIONS.M VIII
IX. BUFFERREADING.M XI
X. BUFFERWRITING.M XIV
XI. TOTALSTALLING.M XVII
XII. RECURRENCE.M XX
XIII. RECURRENCE2.M XXI
XIV. PARAMETERSTOLOAD.M XXII
XV. VERIFORDER_INDEXGENERATOR.M XXVIII
XVI. VERIFSCRAMB_INDEXGENERATOR.M XXX

XVII. VERIFORDER_ADXGEN.M XXXII

XVIII. VERIFSCRAMB_ADXGEN.M XXXIV

XIX. VERIFORDER_SRM.M XXXVI

XX. VERIFSCRAMB_SRM.M XXXVIII

XXI. VERIFORDER_SWM.M XL

XXII. VERIFSCRAMB_SWM.M XLII

XXIII. VERIF_INTERLEAVER.M XLIV

ANEXO II: COMPONENTES Y TESTBENCHS VHDL

I. INTERLEAVERALGORITHM.VHD I

II. INDEXGENERATOR.VHD VI

III. ADDRESSTRANSLATE.VHD IX

IV. ADDRESSGENERATOR.VHD XI

V. LIFOMULTIPLE.VHD XIII

VI. SWITCHMATRIX.VHD XVI

VII. SWITCHREADMEMORY.VHD XXXVIII

VIII. SWITCHWRITEMEMORY.VHD XLV

IX. INTERLEAVER.VHD L

X. TB_INDEXGENERATOR_REPORT_ORDER.VHD LIX

XI. TB_INDEXGENERATOR_REPORT_SCRAMB.VHD LXIV

XII. TB_ADXGEN_ORDER.VHD LXIX

XIII. TB_ADXGEN_SCRAMB.VHD LXXV

XIV. TB_SRM_ORDER.VHD LXXXI

XV. TB_SRM_SCRAMB.VHD LXXXIX

XVI. TB_SWM_ORDER.VHD XCVII

XVII. TB_SWM_SCRAMB.VHD CVII

XVIII. TB_INTERLEAVER.VHD CXVII

1. INTRODUCCIÓN

1.1. CODIFICACIÓN DE CANAL. CONCEPTOS.

Todo sistema de comunicación tiene un mismo objetivo: la transmisión de información desde un emisor hasta un receptor a través de un canal. Debido a la naturaleza ruidosa de todo canal el mensaje es distorsionado hasta llegar al receptor, por lo que en muchos casos se producen diferencias entre las secuencias de datos enviadas y las recibidas, estas diferencias son llamadas errores. Para solventar los errores en recepción se realiza la codificación de canal, cuyo objetivo es que el receptor sea capaz de detectar y corregir los errores producidos durante la transmisión.

La codificación de canal consiste en introducir redundancia, de forma que sea posible reconstruir la secuencia de datos original de la forma más fiable posible. Existen dos técnicas fundamentales en la corrección de datos:

- ***ARQ*** (*Automatic Repeat reQuest*). Detección de errores o corrección hacia atrás. Cuando el receptor detecta un error solicita al emisor la repetición del bloque de datos transmitido. El emisor retransmitirá los datos tantas veces como sea necesario hasta que los datos se reciban sin errores.
- ***FEC*** (*Forward Error Correction*). Corrección de errores o corrección hacia delante. Se basa en el uso de códigos autocorrectores que permiten la corrección de errores en el receptor.

Es fácilmente observable que las técnicas de corrección FEC son más eficaces que las ARQ, pues no se debe retransmitir el mensaje original, por lo que son indicadas para sistemas sin retorno o aplicaciones en tiempo real donde no se puede esperar a una retransmisión. En contra y desde un punto de vista del computacional, las técnicas FEC son más costosas que las ARQ, por lo que generalmente suelen ser aplicadas en sistemas donde un error sea fatídico o en sistemas de comunicación en los que sea altamente costosa una retransmisión o bien, ésta, sea imposible.

1.1.1. ARQ (AUTOMATIC REPEAT REQUEST)

El ARQ es una de las técnicas comúnmente más utilizadas para el control de errores en la transmisión de datos, garantizando la integridad de los mismos. Esta técnica de control de errores se basa en el reenvío de los paquetes de información que se detecten como erróneos.

Para controlar la correcta recepción de un paquete se utilizan ACK's (*acknowledge*) y NACK's de forma que cuando el receptor recibe un paquete correctamente el receptor asiente con un ACK y si no es correcto responde con un NACK. Durante el protocolo que controla recepción de paquetes pueden surgir múltiples problemas (pérdida de ACK, recibir un ACK incorrecto, etc.) complicándose así el contenido del ACK y surgiendo nuevos conceptos como el de *timeout*. Si el emisor no recibe información sobre la recepción del paquete durante un tiempo fijado (*timeout*) éste se reenvía automáticamente.

Suele aplicarse en sistemas que no actúan en tiempo real ya que el tiempo que se pierde en el reenvío puede ser considerable y ser más útil emitir mal en el momento que

correctamente un tiempo después. Esto se puede ver muy claro con una aplicación de videoconferencia donde no resulta de utilidad emitir el *pixel* correcto de la imagen 2 segundos después de haber visto la imagen.

1.1.2. FEC (FORWARD ERROR CORRECTION)

FEC es un tipo de mecanismo de corrección de errores que permite su corrección en el receptor sin retransmisión de la información original. La posibilidad de corregir errores se consigue añadiendo al mensaje original unos bits de redundancia. La fuente digital envía la secuencia de datos al codificador, encargado de añadir dichos bits de redundancia. A la salida del codificador obtenemos la denominada palabra código. Esta palabra código es enviada al receptor y éste, mediante el descodificador adecuado y aplicando los algoritmos de corrección de errores, obtendrá la secuencia de datos original. Los dos principales tipos de codificación usados son:

- *Códigos bloque*: la paridad en el codificador se introduce mediante un algoritmo algebraico aplicado a un bloque de bits. El descodificador aplica el algoritmo inverso para poder identificar y, posteriormente corregir los errores introducidos en la transmisión.
- *Códigos convolucionales*: los bits se van codificando tal y como van llegando al codificador. Cabe destacar que la codificación de uno de los bits está enormemente influenciada por la de sus predecesores. La decodificación para este tipo de código es compleja ya que en principio, es necesaria una gran cantidad de memoria para estimar la secuencia de datos más probable para los bits recibidos.

FEC reduce el número de transmisiones de errores, así como los requisitos de potencia de los sistemas de comunicación e incrementa la efectividad de los mismos evitando la necesidad del reenvío de los mensajes dañados durante la transmisión. En general incluir un número mayor de bits de redundancia supone una mayor capacidad para corregir errores. Sin embargo este hecho incrementa notablemente tanto el ancho de banda de transmisión, como el retardo en la recepción del mensaje.

1.2. TURBO CÓDIGOS

Corría el año 1993 cuando en una conferencia del IEEE sobre Comunicaciones en Ginebra (Suiza), dos ingenieros electrónicos franceses Claude Berrou y Alain Glavieux del departamento de electrónica del *Ecole Nationale Supérieure des Télécommunications* de *Bretagne* en Brest (Francia), hicieron una fabulosa aclamación: habían inventado un esquema de codificación digital que podría llevar a una eventual transmisión sin errores y potencias de transmisión eficaces, más allá de lo que la mayoría de los expertos podían llegar a pensar. Defendían que con ese esquema de codificación y haciendo uso de la misma potencia de transmisión se podía llegar a conseguir el doble de velocidad que con otros esquemas de codificación, o viceversa, con la mitad de potencia en transmisión se conseguían los mismos ratios de transmisión.

Este nuevo esquema de codificación era llamado *Turbo Código* y pertenece a la familia de códigos correctores de errores convolucionales. Ésta codificación posee un rendimiento en

términos de tasa de error de bit (BER) muy próximos al conocido límite de Shannon [1], limite matemático teórico de la máxima tasa de transferencia en entornos ruidosos.

El codificador turbo implementa dos codificadores convolucionales (*ilustración 1*), el primero codifica la secuencia de entrada (de longitud *m* bits) mientras que el segundo codifica la secuencia de entrada permutada por el *interleaver*. De esta forma, cada codificador presenta a su salida unos bits de paridad, de longitud *n/2*. Finalmente la palabra código a enviar al canal es conformada con los bits originales sin codificar (*m* bits), y las dos secuencias de paridad (*n/2* bits cada una) producidas por cada codificador convolucional, por lo que la palabra código presenta una longitud de *m+n* bits, con lo que el ratio del código es:

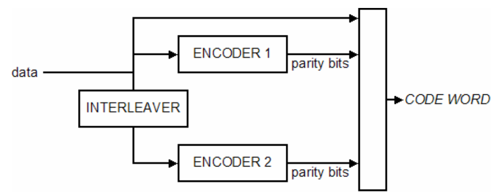
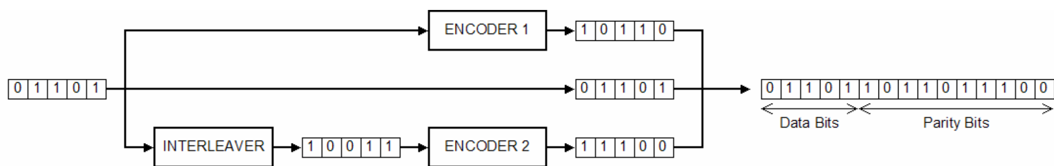


Ilustración 1. Codificador Turbo.

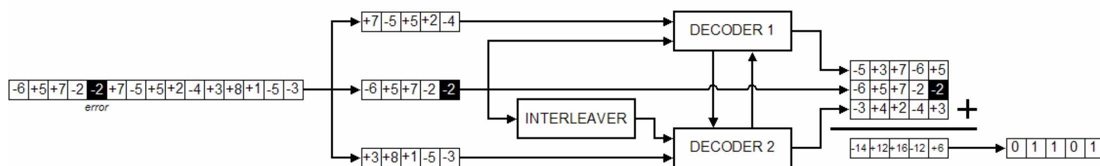
$$R = \frac{m}{m+n}$$

Este cociente representará el tanto por uno de información que existe en la palabra código transmitida, dando así una visión de la proporción de bits de paridad frente a bits de información.

Una vez que son transmitidos los bits por el canal, éstos son expuestos al ruido pudiendo llegar erróneos o no. La señal analógica en recepción es muestreada y cuantificada en un determinado rango, de forma que el valor más bajo de ese rango indicará que lo recibido será un *0*, y el valor más alto del rango que lo recibido es un *1*; por lo que el proceso de cuantifiación es modelable mediante una función densidad de probabilidad. En el ejemplo mostrado en la *ilustración 2* el rango escogido para la cuantificación es [-8 +8], por lo que los números negativos representarán un *0* y los positivos un *1*.



ENCODER



DECODER

Ilustración 2. Ejemplo de codificación y decodificación turbo.

En la decodificación se produce el proceso que se observa en la *ilustración 2*: el primer decodificador recoge la cuantificación llevada a cabo para la secuencias de paridad del primer codificador y la cuantificación de los bits de datos; mientras que el segundo decoder hace lo

propio con la secuencia de los bits de datos permutada y la secuencia de paridad del segundo codificador. Los decodificadores intercambian información entre si repetidamente, y al cabo de un cierto número de iteraciones (normalmente de 4 a 10) cada uno entrega una secuencia de bits decodificada. Finalmente se suman las secuencias proporcionadas por cada decodificador más la correspondiente con la de los datos originales, proporcionando una salida no binaria, sino cuantificada, conociéndose como *decisión suave* de decodificación. Seguidamente, una *decisión dura* decide entre 0 y 1.

Existen múltiples algoritmos que optimizan estas decodificaciones, los más comúnmente usados son los algoritmos Viterbi y los algoritmos recursivos MAP, que son basados en decisiones suaves llamadas *SISO* (*Soft-Input, Soft-Output*).

Las aplicaciones en las que se pueden encontrar esta codificación son múltiples, como en la tecnología UMTS, el estándar de televisión por satélite DVB-S, comunicaciones espaciales, etc…

1.3. TURBO CÓDIGOS DUO-BINARIOS

Los turbo códigos hasta ahora presentados son códigos de dos dimensiones, pues cada dato es codificado al menos dos veces (*ilustración 1*) en codificadores convolucionales [2]. Los códigos no-binarios son construidos a partir de un único codificador convolucional con más de una entrada. Esta familia de códigos ofrecen un mejor rendimiento que los turbo códigos clásicos, tasas de error de bit muy bajas, mejor convergencia en la decodificación iterativa y una menor latencia [3].

El la *ilustración 3* se observa un codificador duo-binario turbo de dos entradas. Los datos son procesados por parejas y son alimentados al codificador en dos pasos, primero en orden normal, y luego en orden permutado, por lo que para cada par de datos *A* y *B* se obtendrán dos bits de paridad pertenecientes al orden normal, Y_1 y W_1, y otros dos pertenecientes al orden permutado, Y_2 y W_2. Al alimentar al codificador en parejas de bits, el bloque total de datos es divido también a la mitad, ganando de este modo en velocidad de procesamiento. No obstante, debido a las propiedades no binarias de éstos códigos así como a sus propiedades circulares, la decodificación llevada a cabo mediante el método clásico resulta mucho más compleja.

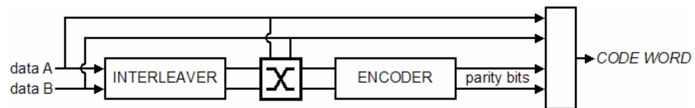


Ilustración 3. Codificador Duo-Binario Turbo.

La cuantificación producida en recepción es modelable mediante una función densidad de probabilidad que indique cuan probable es que la información recibida sea un 1, o bien un 0 lógico, por lo que haciendo usos de

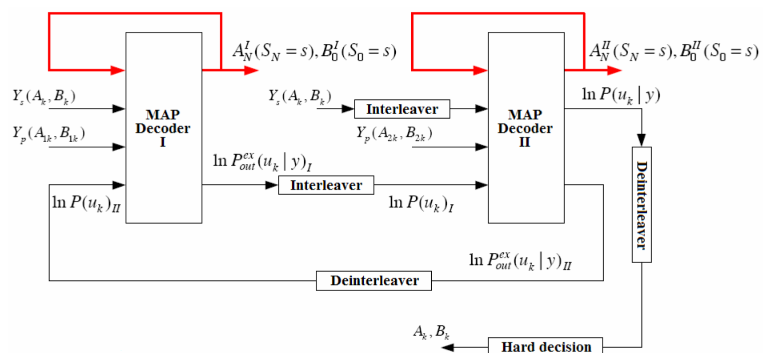


Ilustración 4. Decodificador Duo-Binario Turbo.

algoritmos MAP iterativos se puede llegar a una *decisión blanda*, para después concretar la *decisión dura* (*ilustración 4*). Estos algoritmos MAP son altamente costosos en su implementación para turbo códigos duo-binarios; sin embargo, para el algoritmo MAP modificado y presentado en [4], *Max-Log-MAP*, se obtienen resultados de implementaciones menos complejas, por lo que su adopción en la implementación del decodificador hace superar todos las desventajas iniciales que presentaba la decodificación en un principio.

Este tipo de codificación ha sido escogida por muchos estándares como el DVB de comunicaciones espaciales, WiMax, etc…

1.4. WIMAX

WiMax (*Worldwide Interoperability for Microwave Access*, o *Interoperabilidad Mundial para Acceso por Microondas*) es el estándar tecnológico que permite el acceso en la última milla de forma inalámbrica, siendo la alternativa inalámbrica de banda ancha de las conexiones cable y tecnologías xDSL. WiMax proporciona conectividad de banda ancha fija, itinerante, portable y móvil, sin necesidad de tener visión directa con una estación base. Dispone de radios de celdas de 3 a 10 kilómetros, con tasas de transmisión de hasta 40 Mbps en cada canal.

El ancho de banda proporcionado por esta tecnología es suficiente como para proporcionar múltiples servicios a unos cuantos usuarios a la velocidad típica de un enlace T-1, o bien, a centenares de usuarios con velocidades xDSL. Se espera que en los despliegues típicos llevados a cabo se puedan ofrecer velocidades de hasta 15 Mbps en una celda de 3 kilómetros de radio. En un futuro, las tecnologías WiMax serán incorporadas en dispositivos móviles como ordenadores portátiles, PDAs, teléfonos móviles de altas prestaciones, etc…

La concepción actual de WiMax, denominada *Mobile Wimax*, está basada en el estándar IEEE 802.16e-2005 [7], aprobada el 7 de diciembre del 2005, que permite utilizar este sistema con terminales en movimiento. Muchos fabricantes de hardware y operadores esperaban a esta decisión para empezar a desplegar redes de WiMax, ahora ya saben qué especificaciones técnicas debe tener el hardware del WiMax móvil, que es mucho más jugoso económicamente, con lo que es posible diseñar infraestructuras mixtas fijo-móvil. Este último estándar es una corrección del IEEE 802.16-2004 [6], y por lo tanto el estándar actual es el 802.16-2004 corregido por el 802.16e-2005, por lo que se deben de leer ambos al tiempo para su entendimiento.

El estándar IEEE 802.16-2004 remplaza a los anteriores estándares IEEE 802.16-2001, 802.16c-2002, y 802.16a-2003.

Existen otras tecnologías en el mercado que compiten directamente con WiMax en el sector del acceso inalámbrico de banda ancha, como los sistemas móviles de tercera generación (UMTS y CDMA2000), HIPERMAN, y WiBro. Dos de los mayores sistemas 3G, CDMA2000 y UMTS, compiten con WiMax pues ofrecen acceso a Internet a velocidades DSL además de servicio telefónico. Estándares como el europeo HIPERMAN o el coreano WiBro han sido armonizados como parte de WiMax y no son vistos como competencia, sino como complemento a éste.

1.4.1. MODO OFDMA. CODIFICACIONES DE CANAL

Para resolver los requerimientos de una solución de bajo coste en un entorno multitrayecto (no de visión directa) fue escogida como técnica de transmisión en la capa física la multiplexación por división en frecuencia ortogonal (OFDMA), que dispone de propiedades óptimas para accesos de banda ancha en escenarios radiados.

Todas las codificaciones de canal que presenta el estándar IEEE 802.16 para este modo pertenecen a la familia correctora de errores FEC, entre ellas una es obligatoria y las demás opcionales. La codificación obligatoria del estándar es la "*Codificación Convolucional*" (CC), y las codificaciones opcionales: "*Codificación Turbo en Bloque*" (BTC), "*Códigos de Baja Densidad de Paridad*" (LDPC) y "*Turbo Códigos Convolucionales*" (CTC).

Todas las codificaciones propuestas para el modo OFDMA disponen de corrección de errores en recepción, pero unas ofrecen mejores prestaciones que otras. Es en [5] donde se estudian todas éstas para el enlace de subida, mostrando las bondades de cada una de ellas y comparándolas entre sí. Se demuestra que los dos códigos más robustos a errores son los *turbo códigos convolucionales* (CTC) y los *códigos de baja densidad de paridad* (LDPC), siendo ligeramente superiores los turbo códigos, y quedando muy por detrás los *códigos convolucionales* (CC) usados como codificación base.

Se demuestra así que un decodificador de canal WiMax de altas prestaciones deberá implementar, además de la decodificación convolucional base, decodificadores turbo y decodificadores LDPC.

1.5. OBJETIVO DEL PROYECTO

Se ha observado que la codificación de canal que mejores resultados aporta a la tecnología WiMax es la turbo código. Se ha comprobado también que los decodificadores turbo son de alta complejidad y que requieren de varias iteraciones para recuperar el mensaje original, siendo pieza clave del proceso el interleaver, que normalmente representa el cuello de botella de los decodificadores turbo.

Así pues, el objetivo del proyecto será el diseño de un interleaver de alta velocidad para un decodificador turbo WiMax.

2. INTERLEAVER

2.1. INTRODUCCIÓN

2.1.1. ¿QUÉ ES UN *INTERLEAVER*?

La operación de interleaving es comúnmente usada en comunicaciones digitales para proteger los símbolos transmitidos contra los llamados errores a ráfagas (errores inducidos por el canal de forma continuada sobre los símbolos transmitidos) no pudiéndose llevar a cabo una decodificación correcta.

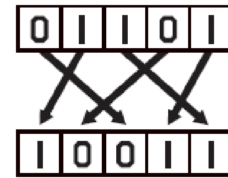


Ilustración 5. Interleaving.

Usualmente, los datos transmitidos son codificados usando bits de paridad para así, poder decodificar correctamente en recepción algunos bits alterados en la transmisión del mensaje. Sin embargo, si se produce un error a ráfagas, los bits alterados en la transmisión son consecutivos y se hace muy difícil, casi imposible, una correcta decodificación del mensaje original. Así pues, si los datos son mezclados antes de su transmisión y aparecen errores a ráfagas en la transmisión, cuando estos lleguen a destino y vuelvan a su estado original (antes de ser mezclados) se obtienen errores no consecutivos, y por lo tanto fáciles de corregir con los bits de paridad pudiendo decodificar correctamente el mensaje recibido. A la operación de mezclado de datos se le conoce como *INTERLEAVING* y a la operación inversa *DE-INTERLEAVING.*

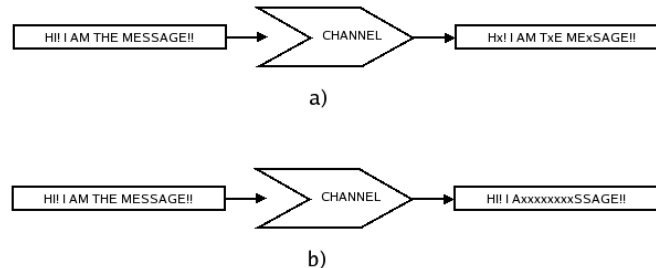


Ilustración 6. [a] errores puntuales. [b] error a ráfagas.

Este mezclado no se puede hacer de forma aleatoria, sino que tiene que seguir un cierto algoritmo para poder recomponer el orden original en recepción. Estos algoritmos se suelen basar en teorías matemáticas complejas y su estudio no es el objetivo de este proyecto, razón por la cual únicamente centraremos la atención sobre el algoritmo interleaver tratado en la siguiente sección.

2.1.2. INTERLEAVER TURBO CÓDIGO [IEEE 802.16e-2005]

El algoritmo interleaver que va a ser estudiado es el usado por la tecnología WiMax en su codificación Turbo Código. En la 16º parte del estándar del IEEE sobre redes de área local y metropolitana (IEEE 802.16) [6] se encuentra la información sobre éste, más exactamente en la sección 8.4.9.2 [*Physical Layer \ WirelessMAN-OFDMA \ Channel coding \ Encoding*].

La codificación que recogen el estándar como obligatoria corresponde con la "*Codificación Convolucional*" (CC), pero también recoge la posibilidad de implementar otras tres codificaciones opcionales: "*Codificación Turbo en Bloque*" (BTC), "*Códigos de Baja Densidad de Paridad*" (LDPC) y "*Turbo Códigos Convolucionales*" (CTC). Es esta última, como ya se menciono en la introducción, sobre la que versara todo el proyecto.

El algoritmo interleaver sobre el que recae nuestra atención se encuentra en la sección 8.4.9.2.3.2. "*CTC Interleaver*" del estándar [6]:

The two-steps interleaver shall be performed by the next algorithm:

Step 1: Switch the alternate couples

Let the sequence $u_0 = [(A_0,B_0), (A_1,B_1), (A_2,B_2), (A_3,B_3), \ldots (A_{N-1},B_{N-1})]$ be the input to first encoding *C1*.

for i = 0 ... *N*-1
  if (i mod 2 == 1) let $(A_i,B_i) \rightarrow (B_i,A_i)$ (i.e., switch the couple)

This step gives the sequence:

$u_1 = [(A_0,B_0), (B_1,A_1), (A_2,B_2), (B_3,A_3), \ldots (B_{N-1},A_{N-1})] = [u_1(0), u_1(1), u_1(2), u_1(3), \ldots , u_1(N-1)]$

Step 2: P(j)

The function *P(j)* provides the address of the couple of the sequence u_1 that shall be mapped onto address *j* on the interleaved sequence (i.e., $u_2(i) = u_1(P(j))$)

for *j* = 0 ... N-1
  switch *j* mod 4:
    case 0: $P(j) = (P_0 \cdot j + 1) \bmod N$
    case 1: $P(j) = (P_0 \cdot j + 1 + N/2 + P_1) \bmod N$
    case 2: $P(j) = (P_0 \cdot j + 1 + P_2) \bmod N$
    case 3: $P(j) = (P_0 \cdot j + 1 + N/2 + P_3) \bmod N$

This step gives a sequence:

$u_2 = [u_1(P(0)), u_1(P(1)), u_1(P(2)), u_1(P(3)), \ldots , u_1(P(N-1))] =$

$= [(B_{P(0)}, A_{P(0)}), (A_{P(1)}, B_{P(1)}), (B_{P(2)}, A_{P(2)}), (A_{P(3)}, B_{P(3)}), \ldots , (A_{P(N-1)}, B_{P(N-1)})]$.

Sequence u_2 is the input to the second encoding *C2*.

Texto 1: Algoritmo de interleaver CTC IEEE 802.16-2004

La primera parte del algoritmo (*step 1*) pertenece a la permutación de los dos bloques de información provenientes de la fuente, característica de los turbo códigos duo-binarios. En su segunda parte (*step 2*) se presenta el algoritmo interleaver, siendo *j* un índice que recorre todas las posiciones del bloque de datos, del primero al último, y *P(j)* la posición conmutada correspondiente a un índice *j*. Es decir, la posición del bloque inicial *j* es conmutada por la posición *P(j)* en el bloque de salida.

Este algoritmo hace uso de cinco parámetros:

- ***N***: tamaño del bloque. El codificador/decodificar es alimentado por *k* bits (*k* = 2·*N* bits)
- ***P0, P1, P2*** y ***P3***: parámetros del algoritmo interleaver.

Según sea el tamaño del bloque, *N*, se disponen de ciertos parámetros *Pi*. En la revisión del estándar del año 2005 [7] se introdujeron ciertos cambios en estos parámetros. En la siguiente tabla (*tabla 1*) se observan éstos tal como el IEEE los considera hoy en día:

N	P0	P1	P2	P3
24	5	0	0	0
36	11	18	0	18
48	13	24	0	24
72	11	6	0	6
96	7	48	24	72
108	11	54	56	2
120	13	60	0	60
144	17	74	72	2
180	11	90	0	90
192	11	96	48	144
216	13	108	0	108
240	13	120	60	180
480	13	240	120	360
960	13	480	240	720
1440	17	720	360	540
1920	17	960	480	1440
2400	17	1200	600	1800

Tabla 1. Valores P0, P1, P2 y P3 según N.

Ilustremos un ejemplo: imaginese un bloque de tamaño 24, por lo que para codificar/decodificar de manera secuencial este bloque se tomaría primero los bits del bloque '*0*', luego los de '*1*' y así hasta el ultimo, el '*23*', es decir:

'*0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23*'

Sin embargo si realizamos una codificación/decodificación basada en el algoritmo interleaver, se obtendría la secuencia:

'*1 18 11 4 21 14 7 0 17 10 3 20 13 6 23 16 9 2 19 12 5 22 15 8*'

De esta forma la secuencia permutada dicta que se tomen primeramente los bits del bloque '*1*', después los del bloque '*18*', etc. Si sobre esta secuencia realizamos un *DE-INTERLEAVING* se obtendría de nuevo la secuencia sin alterar, de '*0*' a '*23*'.

2.1.3. IMPLEMENTACIÓN ELECTRÓNICA. PRIMERA APROXIMACIÓN.

Recopilando la información hasta ahora expuesta se puede esbozar un sucinto croquis de como debería ser un interleaver implementado electrónicamente:

- Es capaz de procesar bloques de distinto tamaño (*N*).

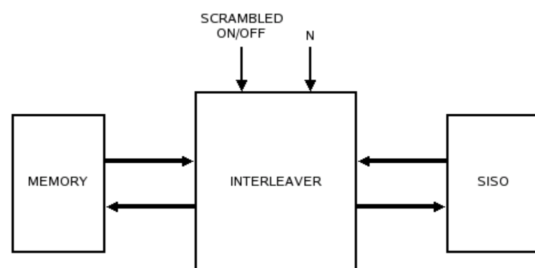


Ilustración 7. Interleaver. Primera aproximación.

- Lee datos de una memoria de forma ordenada o desordenada (según proceda) y los entrega a la SISO encargada del proceso de decodificación.
- Escribe los datos procesados por la SISO de forma ordenada o desordenada (según proceda).

Asumiendo éstos cuatro criterios de funcionalidad básica se puede esbozar un esquema sobre lo que podría ser el interleaver, *ilustración 7*.

Tal como ha sido ilustrado, se tendrá comunicación de datos con la SISO y la memoria, además de las señales de control externas que dictan el tamaño del bloque y si se realizar la lectura/escritura en orden o desorden.

2.2. EL INTERLEAVER PARALELO

"*Divide y vencerás*". Una forma clásica para reducir el tiempo de ejecución de una tarea es introducir el paralelismo, es decir, dividir la tarea en varias partes y realizar estas contemporáneamente, al mismo tiempo.

Si se realizara el algoritmo tal como viene descrito en el estándar (*texto 1*) se necesitarían *N* recursiones para finalizarlo, pero sin embargo y como se vera a continuación, si se paraleliza, se pasaría de *N* recursiones a *n* -minúscula- recursiones, siendo *N* > *n*.

2.2.1. PARALELIZANDO EL ALGORITMO

El caso del algoritmo a estudio (*texto 1*) esta basado en un bucle ***for*** el cual recorre el rango de valores ***[0 ... N-1]***, por lo que es fácilmente paralelizable si se separa éste en varios bucles *for*, haciendo que cada uno de éstos recorra un subrango de valores del rango original. Si se llama ***Pe*** al grado de paralelismo del proceso se puede reescribir el bucle de la siguiente forma:

$$\begin{array}{ccccccc} \text{for } j = \text{rango} & & \text{Proceso 1} & \text{Proceso 2} & & \text{Proceso Pe} \\ [0, \ldots, N-1] & \rightarrow & \left[0, \ldots, \frac{N}{Pe} - 1\right] & \left[\frac{N}{Pe}, \ldots, 2 \cdot \frac{N}{Pe} - 1\right] & \cdots & \left[(Pe-1) \cdot \frac{N}{Pe}, \ldots, N-1\right] \end{array}$$

$$\text{Proceso k} \quad \rightarrow \quad \left[(k-1) \cdot \frac{N}{Pe}, \ldots, k \cdot \frac{N}{Pe} - 1\right]$$

Se define en este punto otro parámetro importante, el ***subloque***, y es referenciado por la letra ***n*** atendiendo a la expresión:

$$n = \frac{N}{Pe} \quad \text{subloque}$$

Ahora, para completar el algoritmo se necesitarán sólo *n* recursiones en lugar de las *N* anteriores, pues hay *Pe* procesos paralelos con lo que se agiliza el proceso global:

$$\begin{array}{ccccccc} \text{for } j = \text{rango} & & \text{Proceso 1} & \text{Proceso 2} & & \text{Proceso Pe} \\ [0, \ldots, N-1] & \rightarrow & [0, \ldots, n-1] & [n, \ldots, 2 \cdot n - 1] & \cdots & [(Pe-1) \cdot n, \ldots, N-1] \end{array}$$

$$\text{Proceso k} \quad \rightarrow \quad [(k-1)\cdot n, ..., k\cdot n-1]$$

Notar que el grado de paralelismo *Pe* debe ser elegido tal que *n* sea un número entero. A continuación es ilustrado un ejemplo del paralelismo descrito para un bloque *N = 24* y con un grado de paralelismo *Pe = 6*, *ilustración 8*:

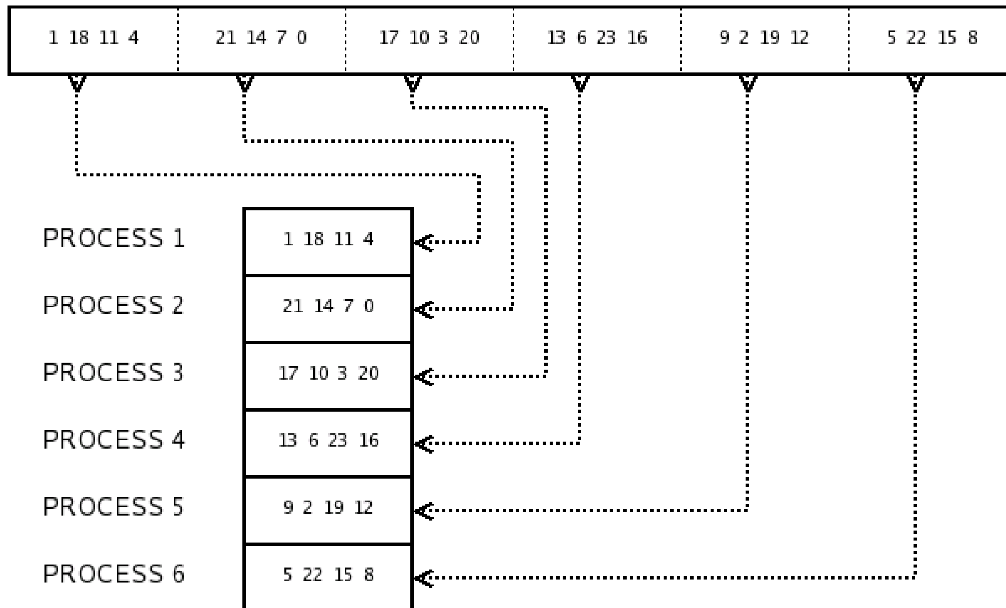


Ilustración 8. Ejemplo del algoritmo paralelizado para N=24 y Pe=6.

Se observa como la secuencia original correspondiente a *N = 24* es dividida en seis subloques, cada uno de ellos con el mismo número de muestras, siendo el primer subloque lo que procesará el proceso 1, el segundo subloque el proceso 2, etc., consiguiendo así procesar todo el bloque de forma paralela y en un menor número de recursiones.

De esta forma, el interleaver es dividido en *Pe* interleavers, realizando cada uno de ellos un bucle *for* de *n* iteraciones, teniendo que escribir/leer estos una memoria y un SISO. Se añade así pues otra entrada para seleccionar con que grado de paralelismo se desea trabajar. En la *ilustración 9* se observa la segunda aproximación de diseño, sólo varía la introducción del parámetro de paralelización *Pe*.

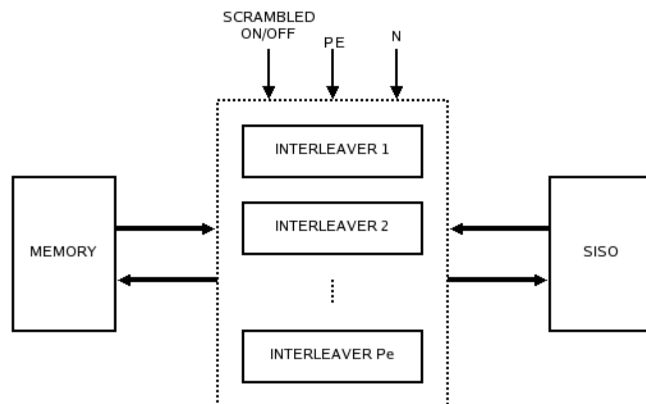


Ilustración 9. Interleaver. Segunda aproximación.

Pero la ilustración anterior no es una aproximación correcta. La SISO también debe de ser paralelizada en *Pe* SISOs, y, la memoria debe disponer de *Pe* puertas para soportar las *Pe* escrituras/lecturas simultaneas, o bien, optar por Pe memorias independientes. La paralelización de

la SISO es posible, y de hecho recomendable para ganar velocidad de procesado según nos relata S. Yoon [8]. Por su parte, en el articulo publicado por F. Speziali [9] se relata que la memoria de *Pe* puertas resulta compleja cuando el grado de paralelismo es superior a 2 y se sugiere el uso de *Pe* memorias independientes de puerta única. Considerando estos dos puntos se puede rediseñar el interleaver tal como se observa en la *ilustración 10*.

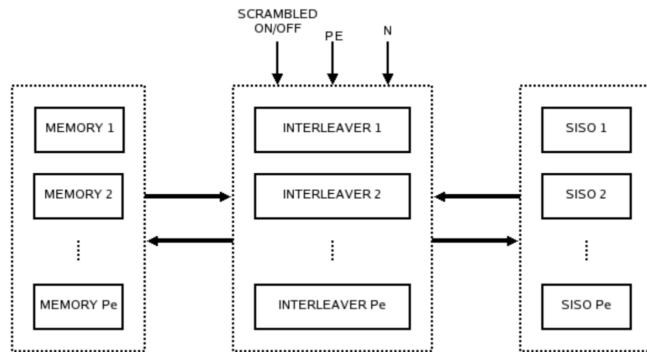


Ilustración 10. Interleaver. Tercera aproximación.

Normalmente, los valores usuales que toma el parámetro *Pe* suelen ser potencias de 2: 2, 4, 8, 16, etc... En el capitulo del estándar [6] que trata sobre el codificador en estudio, *sección 8.4.9.2.3.1*, nos dice que *N* es múltiplo de 4; por lo que no hay problema en escoger los valores $Pe = 2$ y $Pe = 4$.

Sin embargo, estos valores de paralelismo no son eficientes cuando el tamaño del bloque es elevado, pues dan una velocidad total de proceso baja comparada con otros bloques de menor número de elementos. Es por esta razón que también adoptaremos los valores $Pe = 6$ y $Pe = 8$, teniendo que realizar previamente un estudio sobre la viabilidad de estos valores, es decir que el tamaño de subloque para los valores de *N* y *Pe* sea un numero entero. En la siguiente tabla (*tabla 2*) se pueden observar los resultados de este sencillo estudio:

N	Pe			
	2	4	6	8
24	12	6	4	3
36	18	9	6	***4,5***
48	24	12	8	6
72	36	18	12	9
96	48	24	16	12
108	54	27	18	***13,5***
120	60	30	20	15
144	72	36	24	18
180	90	45	30	***22,5***
192	96	48	32	24
216	108	54	36	27
240	120	60	40	30
480	240	120	80	60
960	480	240	160	120
1440	720	360	240	180
1920	960	480	320	240
2400	1200	600	400	300

Tabla 2. Tamaño de subloque *n* según *N* y *Pe*

Se observa que para los valores previamente advertidos, *Pe = 2* y *Pe = 4*, no existe problema a la hora de llevar a cabo la paralelización, pues el tamaño del subloque resulta un número entero. Al igual ocurre con el valor *Pe = 6*, no existe ningún problema pues los subloques *n* resultan ser números enteros. Sin embargo, para *Pe = 8* se obtienen tres casos en los cuales *N* no es divisible por 8:

N = 36 N = 108 N = 180

2.2.2. CASOS ESPECIALES NO PARALELIZABLES. DISCUSIÓN.

A continuación se analiza matemáticamente los tres casos conflictivos. Como se mencionó anteriormente, el tamaño de subloque ha de ser un número entero, por lo que para estos tres casos se define el subloque como el número entero más próximo a cero (función conocida en el argot matemático como *fix*):

$$n = \mathrm{fix}\left(\frac{N}{Pe}\right)$$

$$Pe = 8 \quad \Rightarrow \quad \begin{cases} N = 36 & \Rightarrow \quad n = \mathrm{fix}\left(\frac{36}{8}\right) = 4 \\ N = 108 & \Rightarrow \quad n = \mathrm{fix}\left(\frac{108}{8}\right) = 13 \\ N = 180 & \Rightarrow \quad n = \mathrm{fix}\left(\frac{180}{8}\right) = 22 \end{cases}$$

$$N \neq 8 \cdot n \quad \Rightarrow \quad \begin{cases} N = 36 & \Rightarrow \quad N - n = N - (8 \cdot 4) = 36 - 32 = 4 \\ N = 108 & \Rightarrow \quad N - n = N - (8 \cdot 13) = 108 - 104 = 4 \\ N = 180 & \Rightarrow \quad N - n = N - (8 \cdot 22) = 180 - 176 = 4 \end{cases}$$

Definiendo así el subloque *n* se comprueba que en todos los casos se quedan fuera 4 muestras que no son procesadas. Teniendo esto en cuenta es posible implementar una paralización del algoritmo con un grado *Pe = 8* para estos valores conflictivos de diversas formas:

- Crear un noveno proceso. Los procesos del 1 al 8 elaboran una secuencia de tamaño $\mathrm{fix}\left(N/Pe\right)$, y se crea un noveno interleaver que procesa los 4 elementos restantes leídos/escritos en una novena memoria y un noveno SISO. (*Ilustración 11 - a*).
- Forzar a que cuatro procesos elaboren bloques de tamaño $\mathrm{fix}\left(N/Pe\right)$ y otros cuatro que elaboren bloques de tamaño $\mathrm{fix}\left(N/Pe\right)+1$. (*Ilustración 11 - b*). Se verifica en este caso que:

$$N = 4 \cdot \mathrm{fix}\left(N/Pe\right) + 4 \cdot \left(\mathrm{fix}\left(N/Pe\right) + 1\right) \qquad \text{siempre que} \qquad \frac{N}{Pe} \notin \aleph$$

Así pues, se observa que para el caso (*a*) se consumen $\mathrm{fix}\left(N/_{Pe}\right)$ recursiones para completar todo el bloque *N*; mientras que en el caso (*b*) son consumidas $\mathrm{fix}\left(N/_{Pe}\right)+1$ recursiones. Luego, se puede sentenciar que es más eficiente el caso (*a*) que el (*b*) en cuanto a menor número de recursiones consumidas. En la *ilustración 11* se ilustra cada una de las dos posibles soluciones para un caso concreto: *N = 36*.

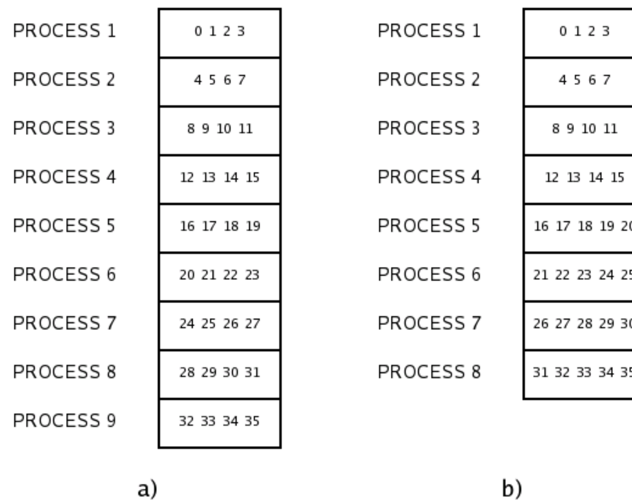


Ilustración 11. Soluciones cuando el tamaño del subloque no es entero.

Lo que se debe de plantear ahora es la siguiente cuestión: "***¿MERECE LA PENA ELABORAR UNA LÓGICA EXTRA EN EL DISEÑO FINAL PARA ESTOS TRES CASOS?***".

Ambas soluciones (*a*) y (*b*) requieren lógica extra en el diseño, como un sincronismo especial entre las SISOs por tener distintos números de muestras (*solución b*), o tener que añadir una novena SISO y memoria (*solución a*), además de la lógica extra para poder distinguir estos casos entre los demás. El sincronismo entre SISOs requerido por la solución (*b*) puede no ser tan complejo, pero necesitaría de biestables para crear la maquina de estados precisa, lo que conllevaría un mayor consumo de recursos en el diseño final. A si mismo, la solución (*a*) consumiría más recursos al tener que implementar una novena memoria y una novena SISO, con la correspondiente maquina de estados para controlar el proceso.

Otro punto a tener en cuenta es el enventanado con solapamiento realizado por el decodificador. Con un solapamiento suficiente de muestras la decodificación será realizada óptimamente y a un alto *bitrate* [10] [11]. Cada subloque *n* es dividido en ventanas, y cada ventana debe de estar compuesta por un número óptimo de muestras. Atendiendo solamente a este concepto se podría desechar directamente la posibilidad de paralelizar a grado 8, pues los subloques proporcionados por tal grado de paralelización para los tamaños que aquí se tratan (36, 108 y 180) disponen de pocas muestras como para realizar un enventanado óptimo sobre ellas.

A continuación se analiza el problema desde el punto de vista de la mejora de la eficacia, es decir, la mejora en cuanto a un menor número de recursiones utilizadas al paralelizar a grado 8 (en sus dos versiones). Se comparará con el grado de paralelismo

inmediatamente inferior, 6. En la siguiente tabla (*tabla 3*) se comprueba tal diferencia de recursiones.

N	Pe = 6	Pe = 8 (Sol. *a*)	Pe = 8 (Sol. *b*)	***DIFERENCIA RECURSIONES*** 6 vs. 8a	***DIFERENCIA RECURSIONES*** 6 vs. 8b
36	6	4	5	***2***	***1***
108	18	13	14	***5***	***4***
180	30	22	23	***8***	***7***

Tabla 3. Comparación de recursiones entre *Pe = 6* y *Pe = 8* (*a* y *b*)

En la tabla, se observa que la mayor diferencia en cuanto a recursiones se da para la solución *a* y *N = 180*, con un total 8 recursiones menos. Para el caso *N = 36* existe una diferencia de 1 y 2 recursiones para las soluciones (*b*) y (*a*) respectivamente; y por su parte, para *N* = *108* la diferencia de los casos (*a*) y (*b*) son de 5 y 4 recursiones respectivamente.

Para ambas soluciones es trivial que existe una mejora en cuanto al número de recursiones consumidas (mejor si se adopta la solución (*b*) que la (*a*). Si se realiza un análisis cuantitativo de las recursiones ahorradas frente a la costosa lógica extra añadida para poder soportar este modo de operación, y sin perder de vista el enventanado que debe soportar el decodificador, se puede declarar sin duda como opciones inválidas la paralelización a grado 8 de los bloques de tamaño 36, 108 y 180; y a la vez, recomendar su paralelización en menores grados de paralelismo.

Se diseña así la siguiente tabla, *tabla 4*, que recoge todos los posibles tamaños de bloque y los casos en que el interleaver acepta la paralelización.

N	Pe 2	Pe 4	Pe 6	Pe 8
24	√	√	√	√
36	√	√	√	
48	√	√	√	√
72	√	√	√	√
96	√	√	√	√
108	√	√	√	
120	√	√	√	√
144	√	√	√	√
180	√	√	√	
192	√	√	√	√
216	√	√	√	√
240	√	√	√	√
480	√	√	√	√
960	√	√	√	√
1440	√	√	√	√
1920	√	√	√	√
2400	√	√	√	√

Tabla 4. Paralelizaciones permitidas.

2.2.3. ACCESO A MEMORIA. COLISIONES.

Como ya se comento anteriormente, y en su momento F. Speziali en [9], la memoria de *Pe* puertas resulta compleja cuando el grado de paralelismo es superior a 2, por lo que recomienda el uso de *Pe* memorias independientes de puerta única. Luego, se escogerán para el diseño *Pe* memorias iguales, las cuales se comunicarán con el interleaver intercambiando datos con él. El interleaver generará *Pe* instrucciones simultáneas de lectura y/o escritura, lo que puede provocar conflicto de acceso a memoria cuando dos o más procesos quieran acceder a la misma memoria.

A estos conflictos de acceso a memoria se les conocerá como ***COLISIONES*** y se deberá estudiar en que situaciones se dan y de que forma, para así poder afrontarlas de una manera más eficaz.

Como ya se ha comentado se disponen de *Pe* memorias independientes, pero… ***¿cómo se distribuyen los datos en la memoria?*** Es decir, si se disponen de *N* bloques, ¿como se distribuyen éstos entre las *Pe* memorias? La situación ideal sería un escenario libre de colisiones, por lo que una distribución uniforme de los *N* bloques en las *Pe* memorias ayudaría a evitar tal situación ya que de este modo la posibilidad de que un bloque se encuentre en una memoria u otra es la misma. Luego sólo se dispondrán de *Pe* memorias activas (según el grado de paralelización del proceso), las cuales dispondrán de *n* bloques cada una de ellas (*ilustración 12*).

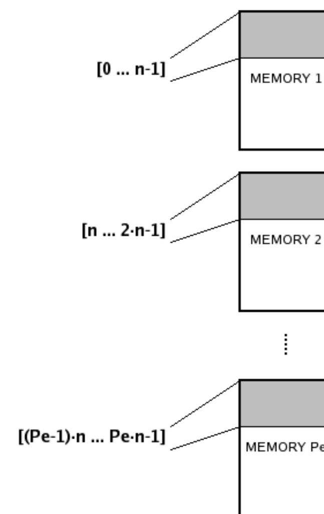


Ilustración 12. Datos en memoria.

En cuanto al direccionamiento, los datos serán almacenados/escritos en las primeras *n* posiciones de la memoria. Así, la posición 0 de la memoria 1 corresponderá con el índice 0 generado por la secuencia del interleaver; la posición *n-1* de la memoria *Pe* (la última memoria activa del proceso) corresponderá al índice *N-1* generado por el interleaver. A continuación se expresará de forma matemática las referencias a memoria y su direccionamiento de un cierto índice *k* de la secuencia *[0 … N-1]*:

Se denominó anteriormente el subloque como: $n = \frac{N}{Pe}$

Así, un cierto índice *k* de la secuencia [0, …, N-1] hace *referencia a la memoria*:

$$\text{fix}\left(\frac{k}{n}\right) + 1$$

y sobre esta memoria, se referencia a la *dirección*:

$$\text{mod}(k,n)$$

A priori, se debería realizar el estudio sobre colisiones para los dos posibles casos de generación de secuencia: secuencia en orden permutado (según interleaver) y secuencia ordenada.

Cuando el interleaver genera una secuencia de índices ordenados, cada uno de los procesos concurrentes *k* genera (en orden) los índices del intervalo:

$$[(k-1)\cdot n, ..., k\cdot n-1]$$

Si los *N* bloques se distribuyen en memoria como se mencionó anteriormente, es fácilmente observable que cada proceso concurrente *k* accederá únicamente a una memoria, evitando así las colisiones. Luego se puede sentenciar que ***LA GENERACIÓN DE LA SECUENCIA ORDENADA NO PRODUCE PROBLEMAS DE ACCESO A MEMORIA O COLISIÓN***. Por tanto, solamente se estudiarán las posibles colisiones cuando el interleaver se encuentre trabajando en modo *desordenado*.

El estudio de las colisiones que son producidas durante el funcionamiento del interleaver se realiza gracias a la programación de un script Matlab® (`collision.m`, presentado en el anexo I). El script está programado para testar todas las posibles combinaciones de *N* y *Pe*, generando a partir de estos parámetros la secuencia desordenada para cada proceso. Una vez obtenida la secuencia, se calcula la memoria a la cual hace referencia cada índice, verificando en cada caso si existe o no colisión. Además, se crea un informe de salida el cual contiene datos de interés, como:

- Valores de *N* y *Pe* para los cuales existe alguna colisión, así como la secuencia de los accesos a las distintas memorias por parte de cada proceso, cada fila representa una iteración y cada columna un proceso; además, cuando en una iteración existe colisión ésta se marca con una ***C*** a la derecha de la fila correspondiente.
- Estadísticas de colisiones. Si se produce alguna colisión en la realización de una secuencia se informa del tanto por ciento de colisiones ocurridas en la generación de dicha secuencia.
- Accesos concurrentes. Para cada valor de *Pe* (2, 4, 6, y 8) y entre todas las realizaciones del parámetro *N*, se reporta la probabilidad del número de accesos concurrentes a memoria.

Se ejecuta el script (*anexo I*) y se obtienen los siguientes escenarios para los cuales existe colisión:

Parámetros *N* y *Pe*		Probabilidad de Colisión
N = 108	Pe = 2	96,2963 %
N = 108	Pe = 4	100 %
N = 108	Pe = 6	88,8889 %
N = 72	Pe = 8	66,6667 %

Tabla 5. Colisiones y probabilidad según valores de *N* y *Pe*.

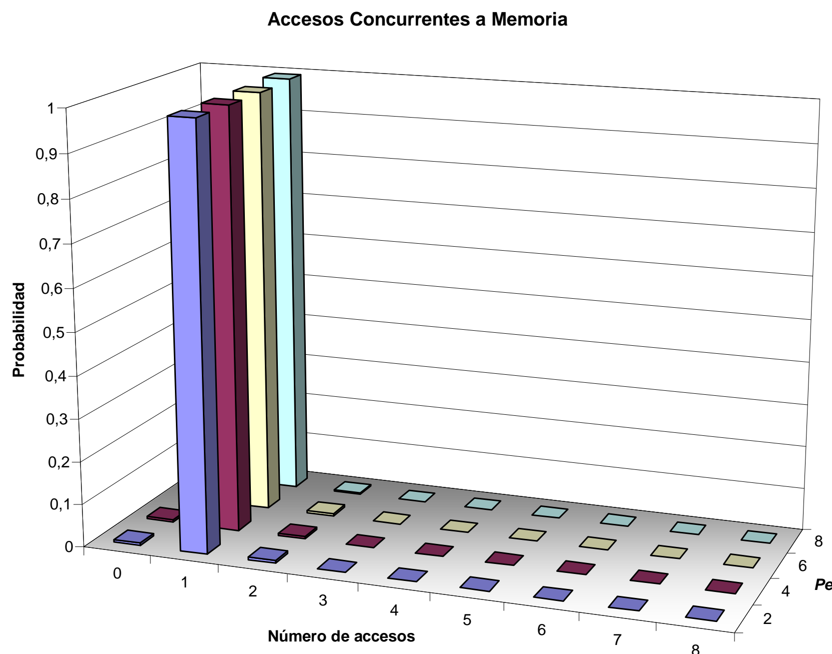
Se observa que son pocos los casos para los cuales hay colisiones (sólo 4 de los 65 casos posibles), pero cuando éstas suceden, tienen lugar en la mayoría de las recursiones del algoritmo, por ejemplo el caso extremo de *N = 108* y *Pe = 4* se podría considerar de *colisión*

total. Solamente el caso *N = 72* y *Pe = 8* parece salirse de la norma, teniendo en su haber 6 iteraciones con colisiones de las 9 posibles, a pesar de todo, siguen siendo demasiadas.

Atendiendo al número de accesos concurrentes a memoria se puede estudiar la naturaleza de estas numerosas colisiones (*tabla 6, gráfica 1*):

Número de accesos concurrentes	Pe = 2	Pe = 4	Pe = 6	Pe = 8
0	0,0059935	0,0059935	0,0055325	0,0028736
1	0,98801	0,98801	0,98893	0,99425
2	0,0059935	0,0059935	0,0055325	0,0028736
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0

Tabla 6. Accesos concurrentes a memoria.



Gráfica 1. Accesos concurrentes a memoria.

Se observa que en la mayoría de los casos existe un único acceso a memoria (acceso libre de colisión), y que cuando existe colisión (2 accesos concurrentes o más) sólo se dan dos accesos concurrentes por ciclo, y además, en la misma proporción que las recursiones sin acceso a memoria (0 accesos). Esto da una idea de cómo se producen las colisiones: "*a pares*", es decir, toda colisión será producida únicamente por dos

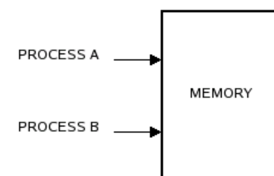


Ilustración 13. Dos accesos concurrentes. Colisión "*a pares*".

procesos, tal como se observa en la *ilustración 13*.

Solventar estas colisiones de lectura/escritura no es trivial, por lo que se dedicara un apartado exclusivo a su discusión y posterior solución.

2.3. COLISIONES: DISCUSIÓN Y SOLUCIÓN

2.3.1. CONSIDERACIONES PREVIAS

Es crucial el tratamiento y solución que se le den a las colisiones, pues el comportamiento final del interleaver estará influenciado por las decisiones aquí tomadas.

Una característica a tener muy en cuenta es que las SISOs deben de recibir los datos de cada proceso de forma contemporánea, es decir, los datos leídos de la memoria por el interleaver se deben de entregar a la vez a las distintas SISOs, aunque existan colisiones que retrasen la lectura de ciertos bloques. Si esto no fuera así las SISOs deberían de tener un sincronismo especial entre ellas [8] difícil de implementar. Para facilitar la comprensión, ilustremos un ejemplo (*ilustración 14*):

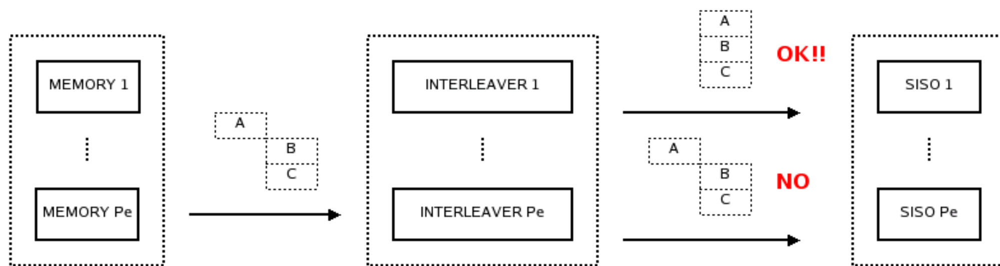


Ilustración 14. Ejemplo de entrega contemporánea de datos a la SISO.

En un instante del proceso, se requiere la lectura de los bloques A, B y C, pero A y B se encuentran en la misma memoria (provocando una colisión) teniendo que retardar la lectura del bloque A (o B). Por su parte, el bloque C no provoca colisión y puede ser leído en el instante requerido. Puesto que estos tres bloques han sido requeridos en el mismo instante temporal, también han de ser entregados a las SISOs al mismo tiempo, tal como se observa en la *ilustración 14*.

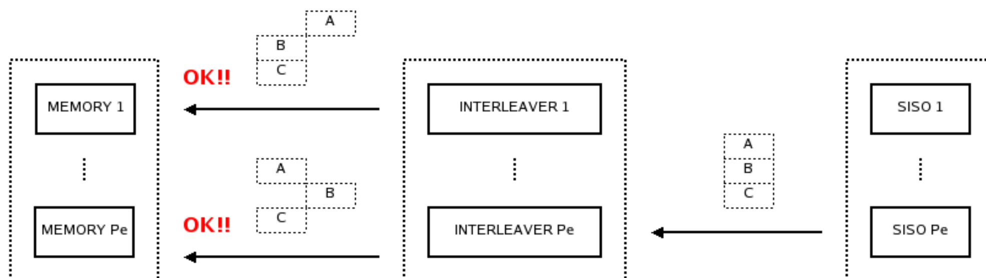


Ilustración 15. Ejemplo de escritura en memoria con colisión.

Sin embargo, cuando los bloques ya han sido procesados y vuelven a pasar por el interleaver para ser escritos en memoria, se vuelve a ocasionar una colisión pues los bloques A

y B han de ser escritos en la misma memoria. Esta vez no importa el orden en que se escriban los bloques, por lo que las dos soluciones presentadas en *ilustración 15* (anteponer el bloque A al B o viceversa) son igualmente válidas.

Las memorias sobre las que se basarán los estudios de esta sección serán síncronas sensibles a los flancos de subida de reloj, de puerta única y con entrada de habilitación. Introducirían un retardo típico de un ciclo para las operaciones de escritura y lectura.

Es pues, función del interleaver, solventar estos conflictos de acceso a memoria (ya sea en lectura o escritura), respetando la restricción impuesta de entrega contemporánea de datos a las SISOs.

2.3.2. REVISIÓN BIBLIOGRÁFICA

F. Speziali realiza una revisión en [9] de distintas filosofías existentes para solventar el problema de las colisiones en los interleavers de alta velocidad. Arquitecturas como la TIBB (*Tree Interleaver Bottleneck Breaker*) [13] o la RIBB [14] (basada en buffers antepuestos en las memorias e interconectados mediante redes en anillo), ambas de M. Thul, son estudiadas y comparadas en este documento [9], y es en éste donde se introducen mejoras a la arquitectura TIBB, que por su sencillez y eficacia es la más comúnmente usada.

La arquitectura TIBB propuesta en [13] consiste en conectar un buffer a la entrada/salida de datos de la memoria, y este buffer a su vez es conectado a una ventana que contiene tantos registros como accesos simultáneos en el peor de los casos posibles. De esta forma, los datos que colisionan se pueden tener al mismo tiempo en la ventana, y de ahí pasan al buffer, el cual, y ciclo a ciclo, se comunica con la memoria. Esta arquitectura reporta una sencilla solución a este problema, pero tiene el inconveniente que el área consumida crece exponencialmente según aumentan los procesos paralelos del interleaver, así como los accesos simultáneos posibles.

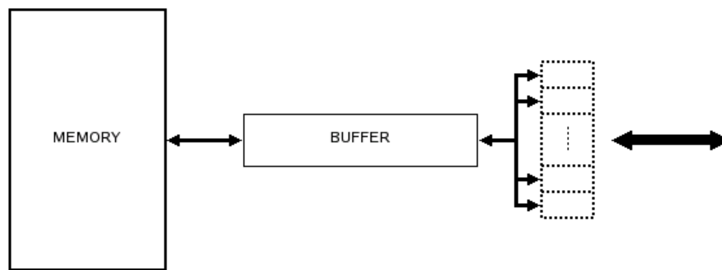


Ilustración 16. Arquitectura TIBB.

Sin embargo, en [9] se propone una modificación a la arquitectura TIBB, la *TIBB con mecanismo de parada*. Esta nueva arquitectura se basa en una TIBB con un menor tamaño de buffers y con lógica extra que permite la parada del sistema para no provocar un desbordamiento de los buffers. Al disponer de unos buffers de menor tamaño el área consumida es mucho menor aún aumenten los *productores* (procesos paralelos). Sin embargo,

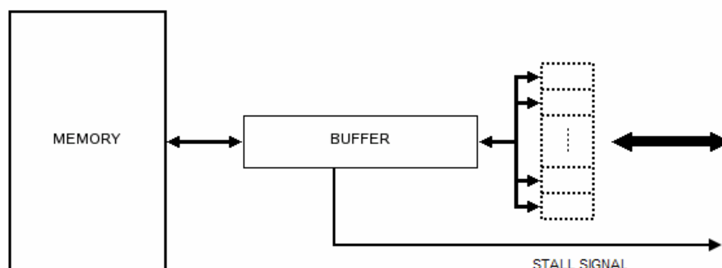


Ilustración 17. Arquitectura TIBB con mecanismo de parada.

las paradas del sistema producidas por el mecanismo comentado provocarán una penalización en el tiempo de proceso total, aunque se demuestra que estas penalizaciones son aceptables en casi todos los casos.

2.3.3. SOLUCIONES POSIBLES

Como se ha visto anteriormente, los dos casos de acceso a memoria -lectura y escritura- atienden a distintos comportamientos, por lo que las soluciones adoptadas para cada escenario no han de ser necesariamente iguales.

Tras la revisión bibliográfica se puede estudiar la viabilidad de las siguientes soluciones para cada caso:

- **LECTURA**: se ha de tener muy presente la necesidad impuesta de que los datos requeridos en cierto instante han de ser entregados simultáneamente a las SISOs. Basado en esta restricción de diseño, se discutirá la viabilidad de las dos soluciones propuestas, pero completamente radicalizadas, una arquitectura puramente TIBB y una arquitectura, que llamaremos, de *parada total*.
- **ESCRITURA**: en la interfaz del interleaver, como se verá en el próximo capítulo, no se especificará la posibilidad de parada de las SISOs, por lo que no se puede implementar un mecanismo como el presentado en [9]. Se debe pues acudir a una arquitectura TIBB y estudiar los tamaños de los buffers, para posteriormente discutir su viabilidad. Notar que la escritura de los datos no han de ser necesariamente contemporáneos a su producción por parte de las SISOs.

Para ambos casos, es indispensable el uso de algún tipo de buffer en memoria. Como se comprobó en el estudio sobre colisiones realizado anteriormente, siempre que exista alguna colisión coexistirán dos accesos simultáneos a memoria, por lo que los buffers implementados han de poder soportar la lectura/escritura de dos procesos simultáneos. La implementación de este tipo de buffers es completamente factible, pues no es igualmente complicado implementar una pequeña memoria de dos puertas (buffer) que una gran memoria de dos o más puertas. Estos buffers tendrán un comportamiento de pila FIFO (*First-In First-Out*).

En ambas situaciones, lectura y escritura de memoria, se considera que el interleaver genera constantemente las órdenes de lectura, o escritura, hasta el término de los *N* bloques.

2.3.4. SOLUCIONES EN LECTURA. ESTUDIO Y VIABILIDAD

En lectura se deberán disponer de varios buffers antepuestos a las memorias para así secuenciar las lecturas conflictivas, sea cual sea la solución adoptada. Se podrán distinguir los siguientes:

- Buffer de entrada, ***BUFFER_en***, los cuales secuencian las lecturas. Deben almacenar un bit de habilitación, así como la dirección de lectura; por lo que el ancho de palabra del buffer será de ADDRESS_SIZE + 1 bits.

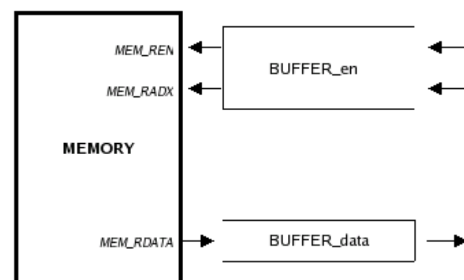


Ilustración 18. Buffers en lectura.

- Buffer de salida de datos, ***BUFFER_data***, donde poder almacenar los datos ya leídos a la espera de ser requeridos por el interleaver para su posterior entrega a las SISOs. El ancho de palabra de este buffer será el mismo que el de los datos, DATA_SIZE.

Los parámetros a estudiar para ambas soluciones propuestas serán la ocupación máxima de los buffers, así como los ciclos totales consumidos para la finalización de la operación de lectura.

En un escenario sin colisiones, la máxima ocupación de ambos buffers será de un elemento, pudiéndose considerar de esta forma a los buffers como simples registros intermedios de los cuales se podría llegar a prescindir. A su vez, en un escenario sin colisiones, los ciclos consumidos para la finalización de la lectura de todos los bloques será igual al tamaño del subloque -*n*- correspondiente más uno; correspondiendo este ciclo extra al tomado por la memoria para leer el dato requerido.

2.3.4.1. ARQUITECTURA TIBB

En esta arquitectura, los buffers se dispondrán en cada una de las ocho memorias tal como se observa en la *ilustración 18*. Según el interleaver genera las órdenes de lectura (señales MEM_REN y MEM_RADX), éstas se introducen en los buffers de las memorias correspondientes, y ciclo a ciclo van siendo atendidas. Los datos de salida de memoria son cargados en su buffer, *BUFFER_data*, y es desde éste donde el interleaver se encarga de recogerlos de forma pertinente para, así, entregarlos correctamente a las SISOs.

El mecanismo de escritura/lectura de estos buffers se puede dividir en dos sencillos algoritmos:

1) Escritura de buffers de entrada. En cada ciclo, se escanea en orden todas las distintas ordenes de lectura producidas por cada proceso, desde el proceso 1 hasta el 8. Si un proceso cualquiera quiere acceder a cierta memoria, se carga en el *BUFFER_en* el correspondiente bit de habilitación de lectura y la dirección a la cual se desea acceder en esa memoria.

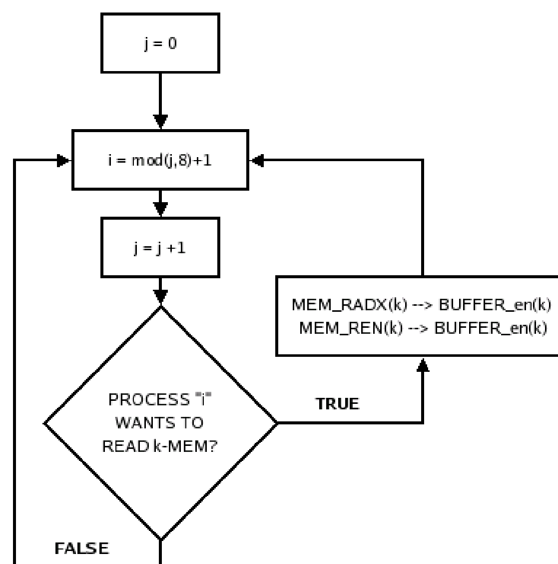


Ilustración 19. Algoritmo para escritura de buffers.

2) Lectura de buffers de salida. El interleaver memoriza el orden y memorias de las cuales se requirieron datos en cierto instante. A partir de la lectura de esta pequeña memoria se va completando la ventana de datos que se ha de entregar a la SISO. Las memorias en las cuales se almacenan los órdenes de lectura de las memorias no se tendrán en cuenta en el estudio.

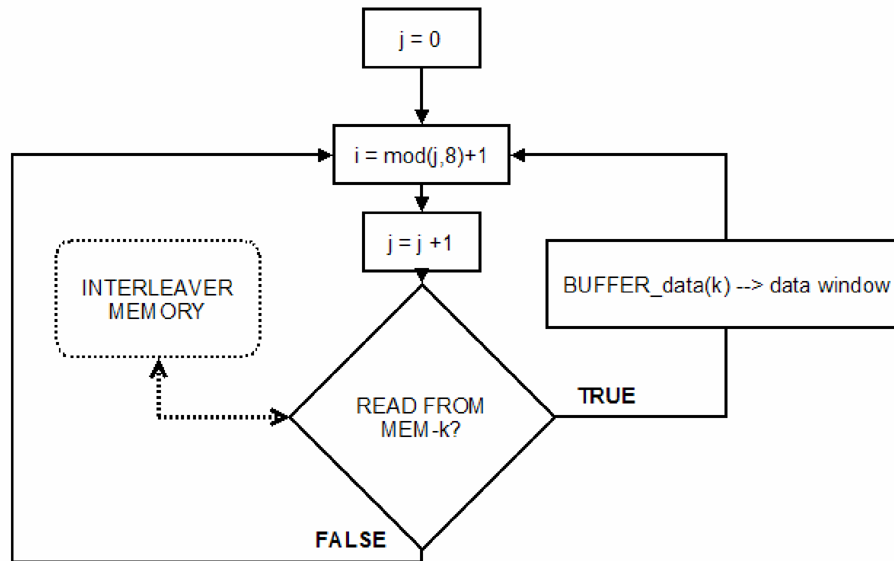


Ilustración 20. Algoritmo para lectura de buffers.

De nuevo, gracias a la programación de script Matlab® (`bufferReading.m`, presentado en el anexo I), se simula el comportamiento de las memorias, buffers e interleaver obteniéndose un informe de salida con todos los datos de interés. Los resultados de máxima ocupación de ambos buffers y ciclos de reloj consumidos son presentados en la siguiente tabla, se desprecian las combinaciones de *N* y *Pe* que solamente requieren de una única posición en cada buffer:

	BUFFER_en	BUFFER_data	CICLOS
N = 108 Pe = 2	5	28	107
N = 108 Pe = 4	5	14	55
N = 108 Pe = 6	6	7	35
N = 72 Pe = 8	4	4	16

Tabla 7. Resultados de simulación de la arquitectura TIBB en lectura.

Efectivamente, se observa que los únicos casos en los cuales existe una ocupación en los buffers de más de un elemento son los casos de colisión. En cuanto a los resultados extraídos, se puede ver que el peor de los caso para el buffer de entrada (*BUFFER_en*) es de 6 elementos, el cual es un tamaño aceptable; pero sin embargo, para el peor de los casos de los buffers de salida (*BUFFER_data*) se da cuando este llega a tener una ocupación total de 28 elementos, un tamaño bastante alto que hace predecir la invalidez, a priori, de esta arquitectura para la lectura en memoria.

2.3.4.2. MECANISMO DE PARADA TOTAL

El mecanismo añadido por F. Speziali a la arquitectura TIBB [9] se basa en la parada de los *productores* de instrucciones de lectura/escritura para no saturar a los buffers. Para su estudio y desarrollo se basaron en unos algoritmos interleaver pseudos-aleatorios, de los cuales no se conocen a priori cuando van a provocar colisión. En el caso que nos ocupa si que se conocen cuando se producirán las colisiones así como la forma de éstas, por lo que es posible desarrollar un algoritmo más preciso a partir de esta información.

En la sección 2.2.3 fueron estudiadas las colisiones provocadas por el algoritmo interleaver que nos ocupa y se llegó a la conclusión que sólo aparecían en unos casos concretos, y que cuando aparecían solamente eran dos los procesos colisionantes. Teniendo todo este conocimiento previo se diseña un mecanismo de parada basado en los estudios realizados en el documento [9] para resolver las colisiones provocadas de una forma más óptima.

La información a priori de la cual se dispone es la siguiente:

1) Las colisiones solamente aparecen cuando:
   N = 108 y Pe = 2 ó *N = 108 y Pe = 4* ó *N = 108 y Pe = 6* ó *N = 72 y Pe = 8*

2) La proporción de recursiones colisionantes frente a las no colisionantes (cuando se dan valores de *N* y *Pe* que producen una) es muy alta, casi 1, en todos los casos.

3) Cuando se da colisión, son ***ÚNICAMENTE DOS*** los *productores* que colisionan, es decir, una petición puede ser procesada al instante mientras la otra ha de esperar al siguiente ciclo para ser procesada.

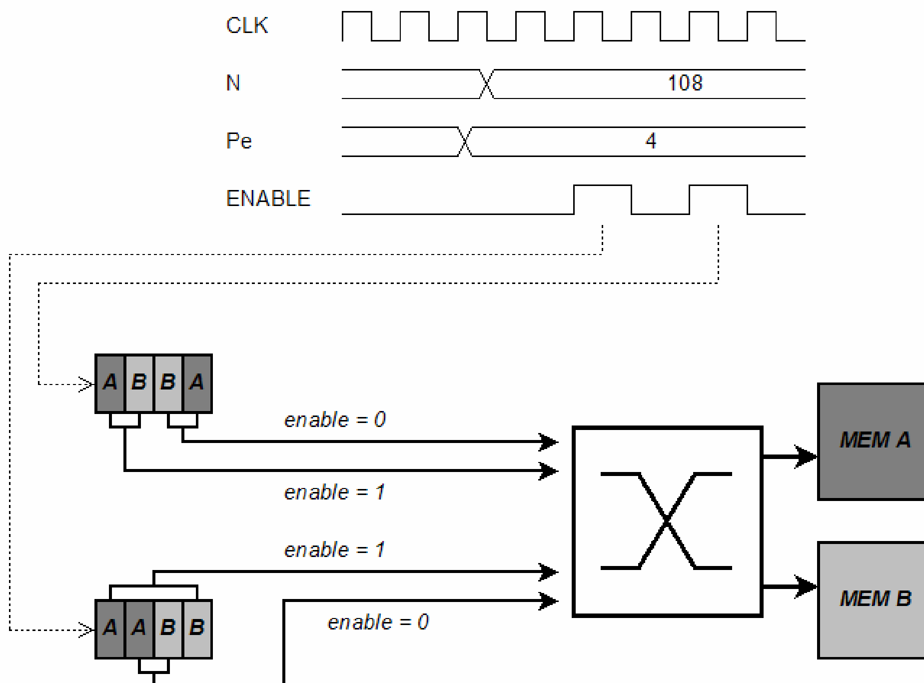


Ilustración 21. Explicación gráfica del mecanismo de *parada total*.

Puesto que cuando se da una colisión, ésta se resuelve en dos ciclos (tercera consideración previa), se puede detener el interleaver un ciclo después de que se haya producido para que de este modo sea “*absorbida*” en el ciclo *ocioso*. En el ciclo de generación de las direcciones colisionantes se atienden aquellas peticiones de lectura que no colisionen entre ellas, y en el siguiente ciclo, las demás que no han podido ser atendidas. De esta forma, en dos ciclos se atienden todas las peticiones de lectura producidas en un ciclo, teniendo solamente que almacenar en memoria los requerimientos de lectura del segundo ciclo y los datos obtenidos de memoria en el primer ciclo; por lo que la memoria buffer consumida será mínima con esta solución.

También se hace necesario la implementación en el interleaver de una máquina de estados interna capaz de parar la generación de instrucciones de lectura en los casos en los cuales se produzca colisión (primera consideración previa).

En la *ilustración 21* se puede observar una explicación gráfica de este mecanismo. La señal *ENABLE* activa (o desactiva) la generación de direcciones por parte del interleaver, por lo que cuando se detecta un caso en el cual se producen colisiones la señal adopta la forma vista: un ciclo en estado lógico activado, y el siguiente ciclo un estado lógico desactivado. En el ciclo en el cual se mantiene activa la señal *ENABLE* se procesan todos aquellos datos posibles que no producen colisión entre ellos, para que en el siguiente ciclo (en el que la señal *ENABLE* se desactiva) sean procesados los datos que no han podido ser procesados. Se llamará a este mecanismo de *PARADA TOTAL* por la particularidad de parar al interleaver ciclo a ciclo cuando se producen colisiones.

En la siguiente ilustración -*22*- se observa un diagrama de flujo y otro de estados, los cuales describen el funcionamiento que debería implementar la máquina de estados del interleaver para implementar el mecanismo de *parada total*.

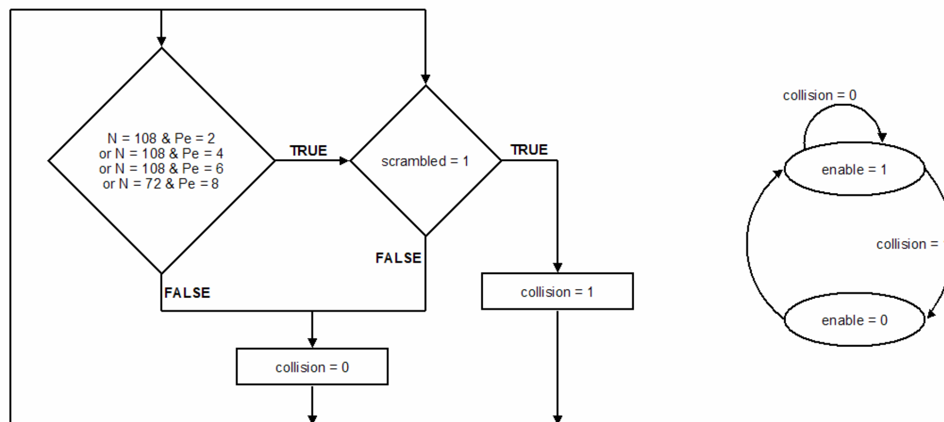


Ilustración 22. Maquina de estados del interleaver para implementar el mecanismo de *parada total*.

Se programa el comportamiento de este mecanismo en el interleaver a través de un script Matlab® (`totalStalling.m`, presentado en el anexo I) con el objetivo de estudiar los ciclos extra de reloj consumidos debidos a las paradas, así como el tamaño máximo que toman los buffers de la memoria, como se hizo anteriormente con la arquitectura TIBB. A

continuación, en la *tabla 8* se presentan los resultados obtenidos de la simulación del mecanismo de *parada total* en lectura de memoria:

	BUFFER_en	BUFFER_data	CICLOS
N = 108 Pe = 2	2	1	109
N = 108 Pe = 4	2	1	55
N = 108 Pe = 6	2	1	37
N = 72 Pe = 8	2	1	19

Tabla 8. Resultados de simulación del mecanismo de *parada total* en lectura.

Se comprueba que el tamaño de los buffers son los mínimos posibles, tamaño dos para los buffers de entrada (dos accesos concurrentes como máximo) y tamaño uno para el buffer de salida (registro para el almacenamiento momentáneo de todos los datos requeridos en el ciclo *activo* del interleaver).

2.3.4.3. COMPARATIVA TIBB – PARADA TOTAL. SOLUCIÓN ADOPTADA

Ambas opciones requieren lógica extra en el diseño: una máquina de estados para el mecanismo de *parada total;* y una pequeña memoria interna en el interleaver (no tenida en cuenta en el estudio) y su correspondiente lógica, para la arquitectura TIBB. Estos elementos extra de fácil implementación no se considerarán decisivos a la hora de elegir una opción sobre otra. Se ha podido comprobar que ambas soluciones satisfacen la necesidad de resolver las colisiones, pero consumiendo diversos recursos cada una de ellas. Para la selección de una solución sobre otra se atenderá a la ocupación de los buffers y a los ciclos extra tomados por cada solución.

	TIBB	PARADA TOTAL
MÁXIMA OCUPACIÓN EN *BUFFER_en*	6	2
MÁXIMA OCUPACIÓN EN *BUFFER_data*	28	1

Tabla 9. Comparativa de ocupación de buffers de memoria en lectura para cada solución.

	TIBB	PARADA TOTAL	*DIFERENCIA CICLOS*
N = 108 Pe = 2	107	109	***2***
N = 108 Pe = 4	55	55	***0***
N = 108 Pe = 6	35	37	***2***
N = 72 Pe = 8	16	19	***3***

Tabla 10. Comparativa de ciclos consumidos por cada solución.

Atendiendo a la ocupación de los buffers (*tabla 9*), la arquitectura TIBB hace un uso intensivo de los buffers de salida y entrada, mientras que el mecanismo de *parada total* minimiza este efecto consumiendo el mínimo posible: dos accesos concurrentes (dos posiciones en *BUFFER_en*) y el registro de salida de memoria (una posición en *BUFFER_data*).

Prestando atención ahora a los ciclos consumidos por cada solución se observa que no existe casi diferencia entre ambas técnicas, siendo la mayor diferencia de 3 ciclos a favor de la arquitectura TIBB, despreciables a las altas frecuencias de los dispositivos microelectrónicos de hoy día.

Parece pues, evidente, que es mejor la solución aportada por el mecanismo de *parada total* frente a la arquitectura TIBB, pues el ahorro de buffers es más que considerable frente a los insignificantes 3 ciclos de desventaja del mecanismo de *parada total*.

2.3.5. SOLUCIÓN EN ESCRITURA, TIBB. ESTUDIO Y VIABILIDAD:

Como se verá próximamente en el capítulo 3, en la interfaz del interleaver no se especifica la posibilidad de parar los datos a escribir en la memoria por parte de las SISOs, por lo que se debe acudir a una arquitectura puramente TIBB y estudiar los tamaños de los buffers. La filosofía de funcionamiento es la misma tanto para lectura como para escritura, se anteponen unos buffers a la memoria, en los cuales, y ciclo a ciclo, se van leyendo (o escribiendo) datos a memoria (*ilustración 16*).

En el caso de la escritura en memoria se dispondrá de un único buffer (*ilustración 23*) ya que todos los datos son de entrada: habilitación y dirección de escritura, así como los datos a escribir, por lo que la longitud de palabra del buffer es de:

DATA_SIZE + ADDRESS_SIZE + 1

Al igual que en lectura, este buffer ha de ser capaz de soportar un máximo de dos escrituras simultaneas.

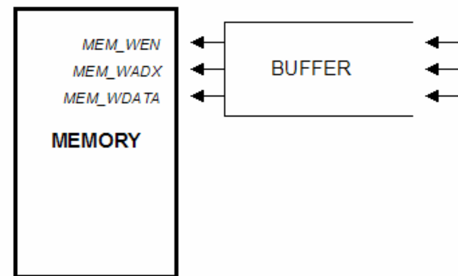


Ilustración 23. Buffer en escritura.

Por las características de la escritura en memoria no es necesario que los datos se escriban contemporáneamente a su producción en las SISOs, lo que simplifica el proceso al algoritmo de escritura en buffers presentado en la anterior *ilustración 19*. Al tener que implementar solamente este algoritmo no se ha de memorizar el orden en el cual se escribe en los buffers, por lo que se libera al interleaver de dicha memoria.

A partir del informe de salida del script Matlab® `bufferWriting.m` (sección X del anexo I), el cual, simula el comportamiento del interleaver en escritura implementando una arquitectura TIBB, se da a conocer la ocupación de los buffers, así como los ciclos extra tomados para la finalización de la escritura de los *N* bloques. La *tabla 11* muestra estos resultados:

	TAMAÑO BUFFER	CICLOS EXTRA
N = 108 Pe = 2	5	2
N = 108 Pe = 4	5	3
N = 108 Pe = 6	6	4
N = 72 Pe = 8	4	3

Tabla 11. Resultados de simulación de arquitectura TIBB en escritura de memoria

Los ciclos extra, como se explicó en el caso de lectura de memoria, es la diferencia de ciclos entre una escritura sin colisiones (*n*+1 ciclos) y la escritura con colisión.

Se verifica que la ocupación máxima de los buffers es de 6 elementos (bajo si se revisan documentos como [9] y [13]), con un máximo de 4 ciclos extra (insignificante a altas frecuencias); por consiguiente, se debería de implementar un total de 8 buffers, uno por cada memoria, de tamaño:

6·(DATA_SIZE + ADDRESS_SIZE + 1)

Luego se necesitarían de un total de: 48·(DATA_SIZE + ADDRESS_SIZE + 1) bits destinados a buffer de doble puerta.

Con unos valores típicos de:

- DATA_SIZE: 32 bits
- ADDRESS_SIZE: 12 bits (representación del bloque máximo *N*)

por lo que se requerirían de un total de 270 bytes destinados a buffer de doble puerta.

No es una solución óptima pues el consumo de memoria no es el mínimo, pero tampoco es una solución desorbitada comparada con estudios similares realizados en [9] y [13], por lo que se declara a ésta solución como ***VIABLE***.

3. IMPLEMENTACIÓN

El fin de este proyecto es la implementación en hardware del interleaver descrito en el capítulo 2. Es en este capítulo donde se describirá y validará la arquitectura de todos los componentes que forman el interleaver, así como el propio interleaver.

3.1. INTERFAZ DEL INTERLEAVER

En esta sección se describe finalmente el comportamiento y la interfaz de entrada/salida que debe implementar el interleaver a diseñar para un correcto funcionamiento.

3.1.1. COMPORTAMIENTO

Como ya se ha comentado, el interleaver estará sumido en el corazón de un decodificador de canal WiMax (IEEE 802.16) para turbo códigos. Como es habitual en las arquitecturas de decodificadores turbo [12], el interleaver interconecta la memoria (donde se almacenan los datos) con la(s) SISO(s) (lógica encargada de la decodificación de datos), intercambiando información entre éstos con la máxima brevedad posible. Los datos que intercambian entre si las distintas entidades son los siguientes:

- INTERLEAVER - MEMORIA: direccionamiento de memoria y datos.
- INTERLEAVER - SISO: datos.

En la implementación, las señales que componen el interleaver serán las típicas señales de control, más buses de direccionamiento de memoria y buses de datos.

De esta forma, el interleaver lee los datos de la memoria, los conmuta según su algoritmo –si procede-, y los pasa a la SISO para su procesamiento. En cuanto los datos hayan sido procesados serán inmediatamente devueltos al interleaver que los volverá a conmutar según proceda y los escribirá, finalmente, en memoria. De esta forma, la secuencia de operaciones llevadas a cabo por el interleaver serán las siguientes:

- Inicio de lectura de datos.
- Conmutación de datos según algoritmo interleaver.
- Entrega de datos a SISO.
- Espera al procesamiento de datos.
- Lectura de datos procesados.
- Conmutación de datos procesados según algoritmo interleaver.
- Escritura en memoria de los datos procesados.

Las lecturas/escrituras en orden o en desorden estarán condicionadas por la ventana en proceso, leyendo ésta en *orden* o en *desorden*, por lo que durante este proceso la señal ***SCRAMBLED*** (añadida en la primera aproximación) se mantendrá estable.

El procesado de bloques por parte de la(s) SISO(s) se realizará a través de algunos de los algoritmos recursivos MAP existentes. El algoritmo presentado en [11] para códigos convolucionales se basa en dos recursiones principales para una ventana de datos, una hacia adelante (*alfa*) y hacia atrás (*beta*). En la primera recursión -*alfa*-, los bloques son leídos e

introducidos del primero al último en las SISOs para ser procesados. En la recursión *beta* los bloques ya procesados son escritos en memoria en orden inverso, del último al primero.

Una vez procesados los bloques por las SISOs, éstos son entregados en orden inverso (hacia atrás, recursión *beta*) al que fueron leídos en la recursión *alfa* (hacia adelante). El interleaver debe escribir estos bloques en la misma dirección y memoria de los cuales fueron leídos, por lo que ha de guardar temporalmente las direcciones y referencias a memoria creadas en la recursión *alfa* para así usarlas posteriormente en orden inverso en la escritura. Esta memoria, en la cual se leen los datos en orden inverso a como fueron introducidos se modela a través de una pila de datos LIFO (*Last-Input, First-Output*).

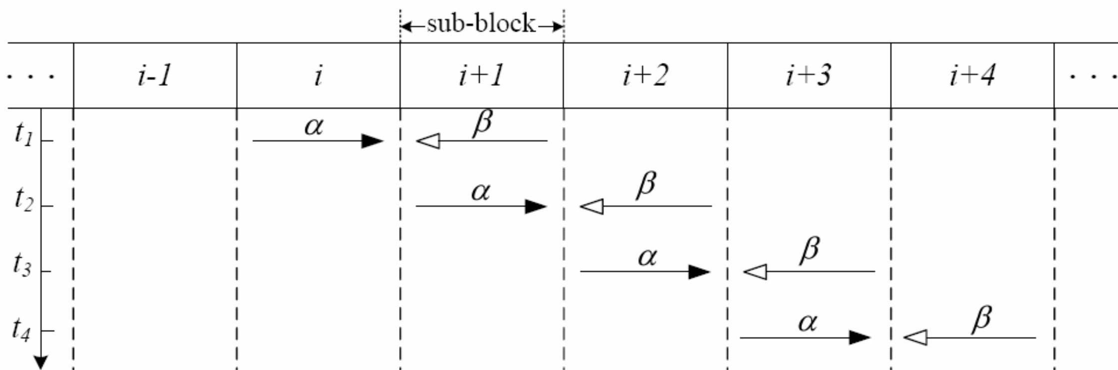


Ilustración 24. Temporalización del algoritmo MAP enventanado.

En la *ilustración 24* se observa la temporalización del algoritmo con las dos recursiones mencionadas, *alfa* y *beta*. Notar que mientras se produce una recursión *alfa*, al mismo tiempo se está produciendo la recursión *beta* de la venta anterior, por lo que coexiste lectura y escritura de la memoria LIFO. Esta situación inducirá a la creación de una pila LIFO doble para evitar conflictos de acceso.

3.1.2. INTERFAZ ENTRADA/SALIDA

A continuación, sin más dilación, se pasa a presentar la interfaz de entrada/salida del interleaver.

Se usaran dos *genéricos* para representar el ancho de los dos tipos de buses que se implementarán, de datos y de direcciones, los de datos se modelarán a través del genérico '***DATA_SIZE***' y los de direcciones con el correspondiente '***ADDRESS_SIZE***'. La existencia de 8 procesos paralelos -como máximo- insta a la creación de 8 buses distintos para cada fin (bus de datos, de direcciones, etc), lo que conlleva un desmedido número de señales en el diseño. Para solventar este problema se unificaran los buses destinados al mismo fin en uno sólo. En la *ilustración 25* se observan 8 buses de datos unificados en uno sólo, por lo que su manejo es realizado más cómodamente, siendo los '*DATA_SIZE*' bits mas significativos (MSB) los correspondientes al octavo proceso y los '*DATA_SIZE*' menos significativos (LSB) los pertenecientes al primero proceso. De esta forma, en un sólo bus se concentra toda la información de, por ejemplo, el bus

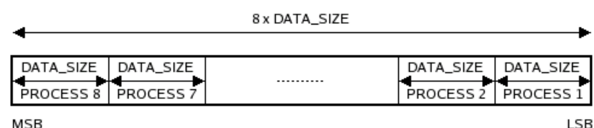


Ilustración 25. Unificación de buses, 8 en 1.

que une la salida de datos del interleaver con la entrada de datos de las SISOs.

Los distintos tipos de buses y señales que componen la interfaz del interleaver los podemos distinguir en cuatro clases:

- señales de reset y reloj.
- señales de control.
- señales de SISO.
- señales de memoria.

En el primer grupo (señales de reset y reloj) se tienen las siguientes tres señales típicas de cualquier diseño electrónico:

- **clk**: reloj, sensible a los flancos de subida.
- **clear**: reset síncrono, activo alto.
- **rst_n**: reset asíncrono de lógica negativa, activo bajo.

La segunda clase de señales, corresponde a las señales de control, se pueden encontrar las señales que controlan el proceso, informado al interleaver de los parámetros básicos del algoritmo. Estas señales son:

- **N**: tamaño del bloque a procesar. El bloque de mayor tamaño es de 2400. Representado en binario puro: 12 bits.
- **Pe**: grado de paralelismo adoptado por el interleaver. Puede tomar los valores: 1, 2, 4, 6 y 8. Representado en binario puro, 4 bits.
- **scrambled**: dicta si la secuencia producida es en orden o en orden permutado -según algoritmo interleaver-. Cuando la señal se encuentre en un nivel lógico bajo se generan índices en orden; por el contrario, si se encuentra en un nivel lógico alto se aplica el algoritmo de interleaver que proporciona un direccionamiento en orden permutado.
- **burst_length**: tamaño de la ventana a procesar. Representado en binario puro. El interleaver sólo debe de leer tantos bloques como se le indique en esta entrada, como máximo *N*. Representado por 12 bits.
- **clear_addGen**: indica el comienzo de un nuevo bloque. Esta señal resetea el generador de direcciones del interleaver para así "comenzar de cero". Activa en alto.
- **start_read**: dispara el proceso. El interleaver comienza a leer tantos bloques como se le indique en *burst_length* y se pausará hasta recibir otro *start_read*, o bien un *reset/clear* que lo resetee. Activa en alto durante un ciclo.

En el tercer bloque se describen las señales del interleaver que son conectadas con la(s) SISO(s):

- **RDATA**: contiene los datos para las distintas SISOs. Bus: 8·DATA_SIZE.
- **RDATA_VALID**: valida los datos de *RDATA* cuando todos ellos son validos en el mismo instante temporal.

- **WDATA**: contiene los datos ya procesados por las SISOs. Bus: 8·DATA_SIZE.
- **WDATA_VALID**: valida los datos de *WDATA* cuando todos ellos son validos en el mismo instante temporal.

Finalmente, el cuarto bloque hace referencia a las señales que interconectan la memoria con el interleaver. Puesto que las memorias disponen de entrada de habilitación se dispondrán de buses de ancho 8 bits (tanto como memorias presentes) para interconectar éstas con el interleaver.

- **MEM_REN**: habilitación de lectura de memoria. Bus de 8 bits, siendo el bit más significativo (MSB) el correspondiente con la habilitación de la octava memoria, y el menos significativo (LSB) el correspondiente a la primera memoria.
- **MEM_RADX**: bus que contiene las direcciones de lectura de las 8 memorias. Bus: 8·ADDRESS_SIZE.
- **MEM_RDATA**: contiene los datos leídos de las 8 memorias (según entrada de habilitación). Bus: 8·DATA_SIZE.
- **MEM_WEN**: habilitación de escritura en memoria. Bus de 8 bits, siendo el bit más significativo (MSB) el correspondiente con la habilitación de la octava memoria, y el menos significativo (LSB) el correspondiente a la primera memoria.
- **MEM_WADX**: bus que contiene las direcciones de escritura de las 8 memorias. Bus: 8·ADDRESS_SIZE.
- **MEM_WDATA**: contiene los datos a escribir en las 8 memorias (según entrada de habilitación). Bus: 8·DATA_SIZE.

A continuación, en la *ilustración 26* se observa el interleaver como bloque, con todos los buses y señales descritos. Tras la ilustración, una tabla (*tabla 12*) recoge un pequeño resumen de todas las señales, su dirección (entrada o salida) y su longitud en número de bits.

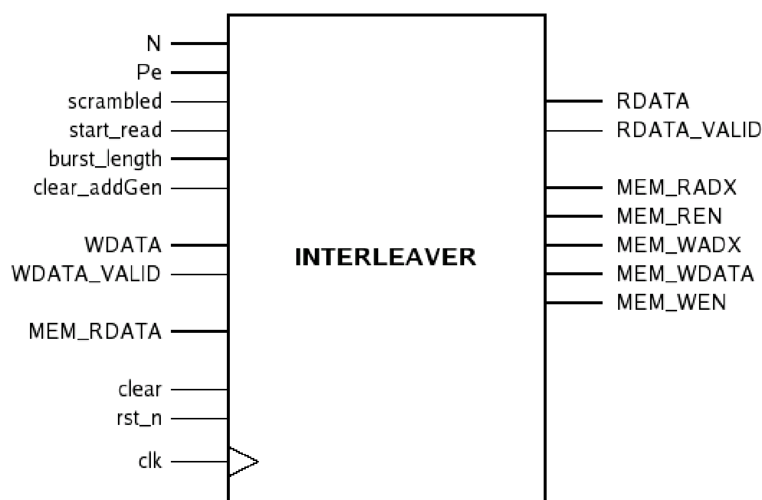


Ilustración 26. Interfaz del interleaver.

	PUERTO	DIRECCIÓN	NÚMERO DE BITS
RESET Y RELOJ	clk	in	1
	clear	in	1
	rst_n	in	1
CONTROL	N	in	12
	Pe	in	4
	scrambled	in	1
	burst_length	in	12
	clear_addGen	in	1
	start_read	in	1
SISO	RDATA	out	8·DATA_SIZE
	RDATA_VALID	out	1
	WDATA	in	8· DATA_SIZE
	WDATA_VALID	in	1
MEMORIA	MEM_REN	out	8
	MEM_RADX	out	8·ADDRESS_SIZE
	MEM_RDATA	in	8·DATA_SIZE
	MEM_WEN	out	8
	MEM_WADX	out	8·ADDRESS_SIZE
	MEM_WDATA	out	8·DATA_SIZE

Tabla 12. Puertos del interleaver.

3.2. FLUJO DE DISEÑO Y VALIDACIÓN. HERRAMIENTAS.

Se hará uso del lenguaje de descripción hardware VHDL para inferir todas las entidades necesarias, así como los correspondientes *testbenchs* de simulación y verificación del diseño.

Por cada componente que presente un comportamiento definido será programado un *testbench*, el cual testeará todas las posibles combinaciones de valores de entrada y reportará los datos de salida a un informe. Por la ingente cantidad de datos se hace imposible la validación "manual", por lo que serán programados scripts Matlab® que lleven a cabo las correspondientes validaciones. La *ilustración 27* muestra el flujo de diseño y validación a seguir:

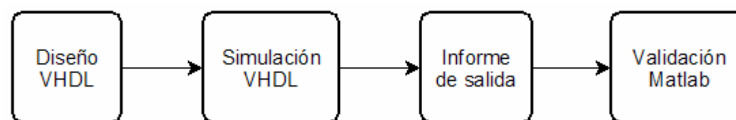


Ilustración 27. Flujo de diseño y validación.

Todos los diseños VHDL, así como sus correspondientes *testbenchs*, serán plasmados en el anexo II, "*componentes y testbenchs VHDL*". Por su parte, los scripts Matlab® de validación podrán ser encontrados en el anexo I.

El siguiente listado refleja el software y hardware del que se hará uso:

- *Escritura código VHDL*: Emacs, provisto del módulo VHDL, licencia GNU.

- *Simulación*: ModelSim® XE III de Mentor Graphics Corporation.
- *Validación de simulación:* Matlab® de MathWorks.
- *Síntesis*: Synplify Pro® de Synplicity.

Y como dispositivo lógico programable:

- *FPGA*: Virtex-II Pro, XC2VP100, de la casa comercial Xilinx®.

3.3. DISEÑO TOP/DOWN

Se ha descrito el comportamiento e interfaz que debe implementar el interleaver, por lo que para atender a las características impuestas desde el inicio del diseño se ha de realizar un diseño desde el mayor nivel de abstracción hacia el más abajo: diseño *top/down*. Es pues en esta sección donde se diseccionará el diseño del interleaver en sus distintas partes funcionales, pero sin caer en el *exceso de detalle*, que podría conllevar un diseño final caótico e ininteligible.

De la descripción del comportamiento del interleaver realizada en la sección 3.1, así como todo lo mencionado en el capítulo 2, es posible distinguir los siguientes componentes dentro del interleaver:

- *Generador de direcciones*. Genera las direcciones y referencias a memoria que han de ser leídas/escritas en orden o en orden conmutado según el algoritmo interleaver. La señal de control externa que dicta *orden* o *desorden* es la señal *SCRAMBLED*.
- *Conmutador de lectura*. Conmuta hacia las SISOs oportunas los datos leídos en memoria. Implementa el mecanismo de *parada total* para solventar las colisiones producidas.
- *Conmutador de escritura*. Conmuta hacia las memorias correspondientes los datos entregados por las SISOs. Implementa una *arquitectura TIBB* para subsanar las colisiones producidas.

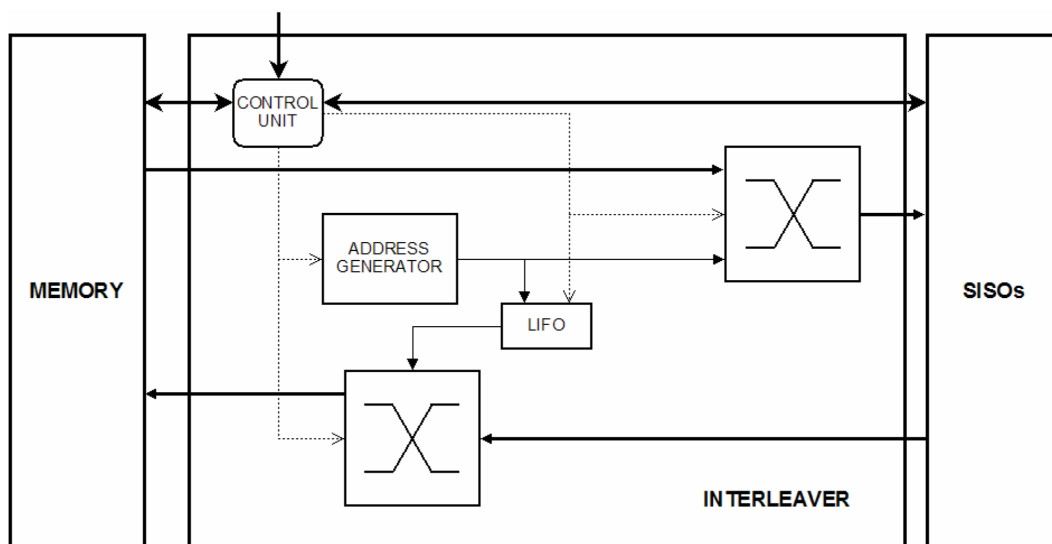


Ilustración 28. Entidades en la primera aproximación *top/down*.

- *Memoria LIFO doble*. Provee al conmutador de escritura de las direcciones y referencias a memoria que necesita.
- *Unidad de control central*. Controlará todos los procesos, como la conmutación de las LIFOs, el mecanismo de *parada total* en lectura, las señales de control del proceso, el enventanado de datos, etc…

Componentes como *conmutadores*, *unidad de control* y *LIFO* son considerados a este nivel partes funcionales por si mismas, ya que quedan bien definidas por su comportamiento. Sin embargo, el *generador de direcciones* aún posee un comportamiento demasiado general, por lo que se desciende en su jerarquía para poder describirlo en mayor detalle.

El *generador de direcciones* es el encargado de calcular las direcciones y memorias que se han de leer y/o escribir, ya sea en *orden* o en *desorden* (según el algoritmo de interleaver). Se deberá de inferir una entidad capaz de implementar el algoritmo interleaver paralelizado, así como una memoria ROM en la que se almacenen todos los parámetros de los que se sirve para realizar el cálculo de los índices (*tabla 1*). El algoritmo interleaver generará únicamente índices, no direcciones y referencias a memorias, por lo que se ha de disponer de una entidad que traduzca estos índices a referencias y direcciones. En la *ilustración 29* se observan las distintas entidades que compondrán el *generador de direcciones*, así como su jerarquía.

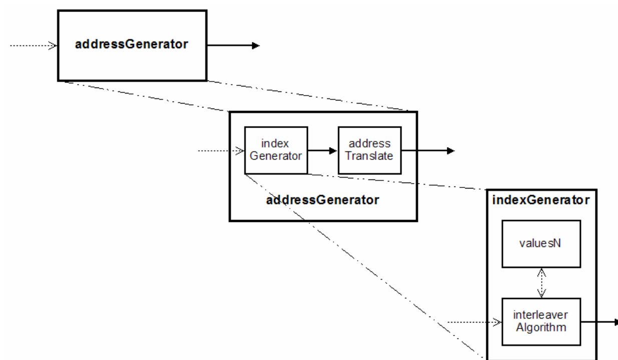


Ilustración 29. Generador de direcciones. Top/down.

En el siguiente árbol (*ilustración 30*) se muestran las entidades descritas hasta el momento, así como la jerarquía a la que están sujetas:

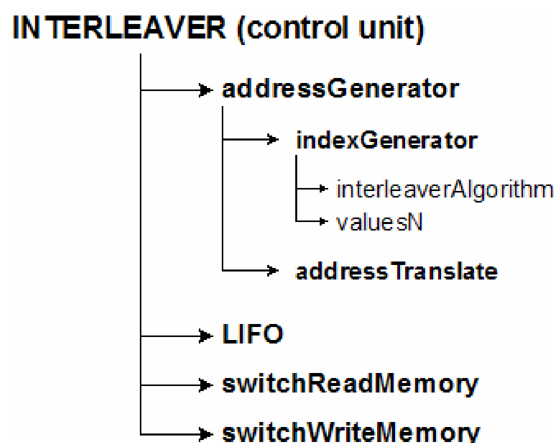


Ilustración 30. Interleaver, jerarquía de entidades.

Una vez completado el diseño *top/down* que difine las entidades que componen el interleaver y su jerarquía se debe recorrer el camino inverso, de abajo hacia arriba (*down/top*),

donde se antenderá en todo detalle el diseño de cada entidad. Se comenzará por el nivel jerárquico más bajo definido por *interleaverAlgorithm* y *valuesN*, y se subirá paulanitamente en la escala jerárquica hasta llegar, finalmente, al *interleaver*.

3.4. DISEÑO DOWN/TOP

En esta sección se describirán todas las entidades, desde el nivel de abstracción más bajo al más alto. A continuación se comienza el estudio de las entidades del nivel más bajo.

3.4.1. ALGORITMO INTERLAVER Y MEMORIA DE PARÁMETROS

Se estudiaran juntas estas dos entidades por estar situadas en el nivel más profundo de la jerarquía, y porque una no tiene sentido sin la otra. La entidad que describe la generación de índices según el algoritmo interleaver en su versión paralelizada será llamada *interleaverAlgorithm*, y la memoria ROM donde se almacenan los parámetros necesarios para procesar el algoritmo recibirá el nombre de *valuesN*.

3.4.1.1. MANIPULACIÓN DEL ALGORITMO

El algoritmo interleaver presentado en *texto 1* presenta operaciones altamente costosas de implementar, como son multiplicaciones, divisiones y módulos. Es posible manipular el algoritmo para que presente únicamente operaciones de suma, resta, y comparación, mucho menos costosas de implementar.

En la memoria ROM se introducirán los parámetros para realizar el algoritmo (*P1*, *P2*, *P3* y *P4*). Sin embargo, y como primera medida para reducir la complejidad del algoritmo implementado, se introducirán otros parámetros en lugar de éstos. Los parámetros *Pi* vienen en función de *N*, por lo que se puede realizar la siguiente simplificación:

for j = 0 … N-1

switch mod(j,4)

case 0: i = mod(P0·j + 1, N)

case 1: i = mod(P0·j + *cte2*, N) ⇒ ***cte2 = 1+N/2+P1***

case 2: i = mod(P0·j + *cte3*, N) ⇒ ***cte3 = 1+P2***

case 3: i = mod(P0·j + *cte4*, N) ⇒ ***cte4 = 1+N/2+P3***

De este modo se ahorrarán dos registros de desplazamiento a derecha (división entre dos) y unas cuantas sumas. El ahorro no es del todo significativo, pero es beneficioso. Así, se sustituirán en memoria los parámetros *P2*, *P3* y *P4* por los parámetros *cte2*, *cte3* y *cte4*.

Tal vez, la operación que más inconvenientes conlleva en implementaciones electrónicas sea la multiplicación, pues suele consumir numerosos recursos. Se observa en el algoritmo que solamente existe una multiplicación, la realizada por el parámetro *P0* al 'puntero' *j*, el cual crece de uno en uno en cada ciclo. Con estas características es fácilmente convertible esta

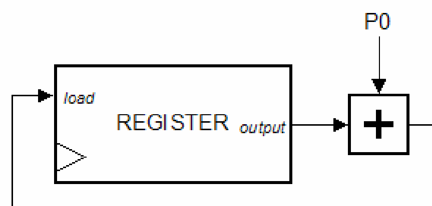


Ilustración 31. Producto implementado como suma.

multiplicación en una suma recursiva del parámetro *P0*. En la siguiente tabla (*tabla 13*) se observa como actúa ciclo a ciclo la multiplicación y como es transformable a una suma recursiva.

j	**P0·*j***		*j*	**P0·*j***
0	0		0	0
1	P0		1	+P0 = P0
2	2·P0	⇒	2	+P0 = 2·P0
⋮	⋮		⋮	⋮
N-1	(N-1)·P0		N-1	+P0 = (N-1)·P0

Tabla 13. Transformación de la multiplicación "*P0·j*" en una suma recursiva.

De la forma descrita, la costosa multiplicación es sustituida por una liviana suma y un registro donde almacenar el resultado anterior. De este modo el algoritmo queda como sigue:

```
for  j = 0 … N-1
        if  j = 0          →      acumulator = 0
        else               →      acumulator = acumulator + P0
        end if

        switch mod(j,4)
        case 0:       i = mod(acumulator + 1, N)
        case 1:       i = mod(acumulator + cte2, N)
        case 2:       i = mod(acumulator + cte3, N)
        case 3:       i = mod(acumulator + cte4, N)
```

El algoritmo aun no es del todo óptimo, pues falta abordar la implementación de las operaciones *módulo*, la cual atiende a la siguiente expresión:

$$\mathrm{mod}(x, y) = (x - y) \cdot \mathrm{floor}\left(\frac{x}{y}\right)$$

Siendo la función *floor* aquella que calcula la parte entera del número que recibe como argumento. La implementación de esta función tal como viene descrita es altamente costosa pues debería implementar una multiplicación y una división de parte entera. Sin embargo, existe un algoritmo recursivo que hace uso de pocos recursos para el calculo del *módulo*. Este algoritmo converge en un número indeterminado de recursiones hacia el resultado final:

```
x = mod(y,z)
        x = y
        while  x >= z          →       x = x – z
```

Al implementar el operador módulo de esta forma solamente será necesario un registro acumulador, un restador y un comparador de magnitud, recursos muy poco costosos comparados con la multiplicación y división anterior.

Se distingue ahora entre las dos distintas métricas que toma el operador *módulo*: *módulo-N* y *módulo-4*. La aritmética modular posee una característica curiosa entre la métrica

que aplica y la base del sistema numérico de trabajo. Esta característica se aplicará a los cálculos en *módulo-4*:

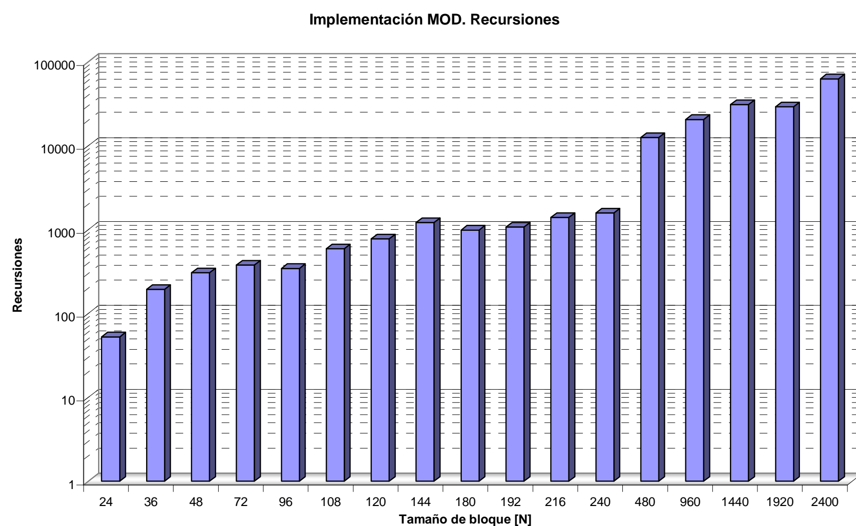
$$\text{mod}(A,4) = \text{2LSB}(\text{binario}(A))$$

Es decir, en binario puro bastará con seleccionar los dos bits menos significativos del número A cuando se quiera conocer su *módulo-4*. Con esta sencilla resolución de *módulo-4* el algoritmo queda de la siguiente manera:

```
for  j = 0 … N-1
        if  j = 0          →    acumulator = 0
        else               →    acumulator = acumulator + P0
        end if

        aux = 2LSB( binario( j ))

        if  aux = 00            →    i = acumulator + 1
        else if  aux = 01       →    i = acumulator + cte2
        else if  aux = 10       →    i = acumulator + cte3
        else if  aux = 11       →    i = acumulator + cte4
        end if

        while  i >= N           →    i = i – N
```

En esta última aproximación sólo se hace uso de comparadores de magnitud, registros, sumas y restas. Los recursos se han minimizado, pero se corre el riesgo de que la normalización de la salida *i* consuma un ingente número de recursiones.

Para validar esta nueva versión del algoritmo original se simula previamente su comportamiento a través de un script Matlab® (`recurrence.m`), el cual calcula y compara, para cada valor de *N*, la secuencia interleaver según la nueva versión y la original. El script también crea un informe que contiene el número total de recursiones tomadas para la



Gráfica 2. Recursiones consumidas al implementar módulo mediante un algoritmo recursivo.

normalización de cada secuencia.

La nueva versión del algoritmo es válida pues para los mismos valores de N se obtienen -en ambas implementaciones- las mismas secuencias. Sin embargo, tal como se observa en la *gráfica 2*, para todos los casos se han de realizar gran número de recursiones, por lo que la implementación de la nueva versión es inviable si se desea diseñar un interleaver de alta velocidad.

El elevado número de recursiones que se han de realizar se debe al crecimiento paulatino del registro *acumulador* (en el que se implementa el producto como una suma recursiva). Puesto que el operador *módulo* es lineal es posible implementar una normalización después de la suma y cargar al registro con el resultado de dicha normalización, con lo que el número de recursiones descenderá considerablemente. Haciendo uso de esta misma filosofía convendría normalizar previamente (antes de ser memorizados) los parámetros *cte2*, *cte3* y *cte4*, por lo que al sumarse con el registro *acumulador* para obtener la salida solamente sería necesaria una recursión para su normalización. Con estas nuevas consideraciones se reescribe el algoritmo de forma que no se hace necesaria ninguna recursión extra:

Siendo:

cte2 = mod(1+N/2+P1, N)

cte3 = mod(1+P2, N)

cte4 = mod(1+N/2+P3, N)

```
for  j = 0 … N-1
        if  j = 0          →     acumulator = 0
        else               →     acumulator = acumulator + P0
        end if
        if  acumulator >= N          →     acumulator = acumulator – N
        end if
        aux = 2LSB( binario( j ))
        if  aux = 00       →     i = acumulator + 1
        else if  aux = 01  →     i = acumulator + cte2
        else if  aux = 10  →     i = acumulator + cte3
        else if  aux = 11  →     i = acumulator + cte4
        end if
        if  i >= N         →     i = i – N
        end if
```

Al igual que en el caso anterior se comprueba la validez de esta versión del algoritmo mediante el script Matlab® `recurrence2.m`, pero esta vez sin consumir ninguna recursión extra.

Se ha conseguido una versión del algoritmo interleaver inicial (*texto 1*) el cual posee el mismo comportamiento que el original y que consume los mínimos recursos posibles para ello.

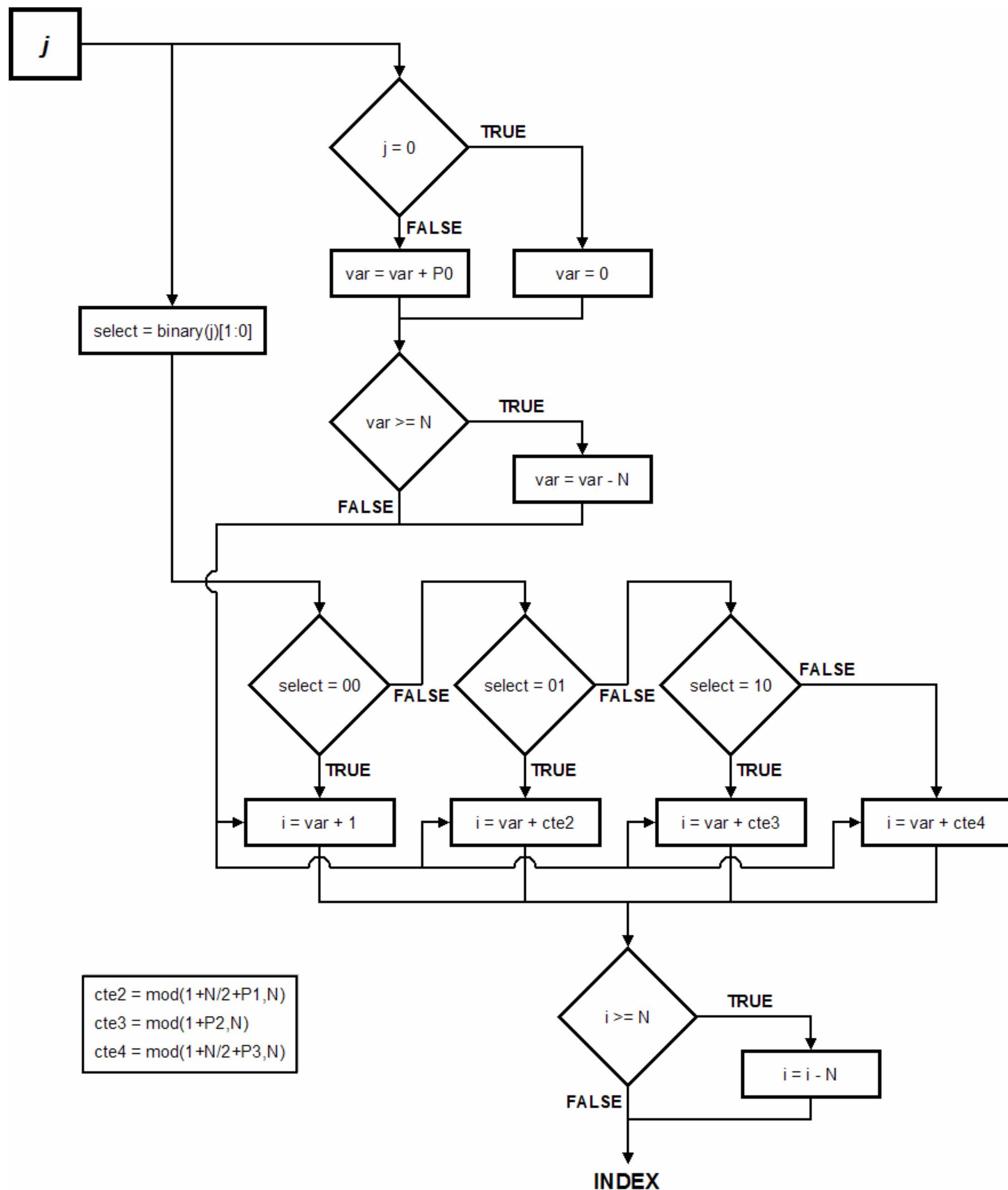


Ilustración 32. Versión implementable electrónicamente del algoritmo interleaver. Diagrama de flujo.

Sin embargo, la versión del algoritmo presentada en la *ilustración 32* no implementa la paralización deseada. En la sección 2.2.1 se estudió la forma de implementar la paralización y se comprobó que cada proceso debía recorrer el subrango de valores del índice *j*:

$$\text{proceso k} \quad \rightarrow \quad [k \cdot n, ..., (k+1) \cdot n - 1]$$

Por lo que bastaría, a priori, con que cada proceso inicie desde su índice correspondiente:

$$k{\cdot}n$$

Esta última afirmación no es del todo correcta. En la última versión del algoritmo (*ilustración 32*) no se trata la inicialización del índice *j* sino la inicialización del producto '*P0·j*' (*tabla 13*), por lo que no basta con sustituir el índice *j* por el de inicio de cada subloque '*n·k*', sino la sustitución del correspondiente '*P0·n·k*' En la *tabla 14* se observa la evolución de estos parámetros en relación con cada proceso:

	j	***P0·j***
	0	***0***
proceso 1	⋮	⋮
	n-1	n-1·P0
	n	***n·P0***
proceso 2	⋮	⋮
	2·n-1	(2·n-1)·P0
⋮	⋮	⋮
	(Pe-1)·n	***((Pe-1)·n)·P0***
proceso Pe	⋮	⋮
	n·Pe-1	(n·Pe-1)·P0

Tabla 14. Interleaver paralelo. Inicialización del parámetro "*P0·j*".

Así, en vez de inicializar en cero cada proceso, (primera condición en *ilustración 32*), se inicializa con la variable:

$$sumBegin(k) = P0 \cdot k \cdot n$$

Puesto que este nuevo parámetro depende de cada proceso y su complejidad es alta, será memorizado y tomado como entrada de cada proceso. Para asegurar que no se ha de consumir ninguna recursión extra en la normalización del registro *acumulador*, el nuevo parámetro *sumBegin* será normalizado antes de ser memorizado. Finalmente:

$$\boldsymbol{sumBegin(k) = mod(P0 \cdot k \cdot n,\ N)}$$

De esta forma se respeta en la inicialización el producto '*P0·j*'. En la *ilustración 33* se muestra el cambio introducido al diagrama de flujo de la *ilustración 32*.

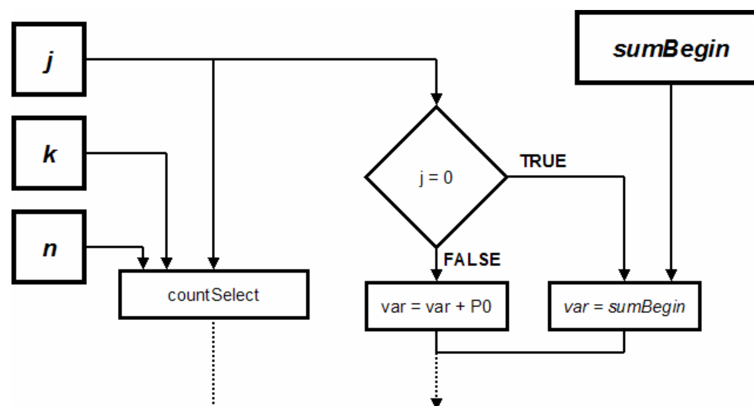


Ilustración 33. Cambios a realizar en el algoritmo para implementar la paralelización.

Pero este no es el único cambio introducido, también se ha de adaptar a la paralelización el procesado del *módulo-4*, que anteriormente se realizaba seleccionando los dos bits menos significativos del índice *j*. Al paralelizar existen algunos procesos que en su

inicio ($j = 0$) no se ha de seleccionar el "*case 0*", sino otro. A continuación se observa en la *tabla 15* un ejemplo, la evolución de los índices para $N = 24$ y $Pe = 8$:

PROCESO	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***
↓	1 (***0***)	4 (***3***)	7 (***2***)	10 (***1***)	13 (***0***)	16 (***3***)	19 (***2***)	22 (***1***)
	18 (*1*)	21 (*0*)	0 (*3*)	3 (*2*)	6 (*1*)	9 (*0*)	12 (*3*)	15 (*2*)
	11 (*2*)	14 (*1*)	17 (*0*)	20 (*3*)	23 (*2*)	2 (*1*)	5 (*0*)	8 (*3*)

Tabla 15. Evolución de los índices generados para $N = 24$ y $Pe = 8$. Entre paréntesis el "*case*" a seleccionar.

Se observa que no todos los "*cases*" iniciales son cero. Esto se debe a que n no es en todos los casos múltiplo de 4. Para solventar este problema se diseña la unidad *countSelect*. El algoritmo que implementa esta unidad –*countSelect*- se basa en el análisis de los dos bits menos significativos del tamaño de subloque n que son el *offset* entre los "*cases*" del proceso k y $k+1$.

De esta forma se realiza la *tabla 16*, la cual presenta la corrección de los "*cases*" a seleccionar según el tamaño de subloque n y el proceso k correspondiente. En cada casilla se representa la corrección a sumar a los dos bits menos significativos del índice j:

2LSB(*n*)	**PROCESO 1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
00	+ '00'	+ '00'	+ '00'	+ '00'	+ '00'	+ '00'	+ '00'	+ '00'
01	+ '00'	+ '01'	+ '10'	+ '11'	+ '00'	+ '01'	+ '10'	+ '11'
10	+ '00'	+ '10'	+ '00'	+ '10'	+ '00'	+ '10'	+ '00'	+ '10'
11	+ '00'	+ '11'	+ '10'	+ '01'	+ '00'	+ '11'	+ '10'	+ '01'

Tabla 16. Corrección del índice *j* implementada por "*countSelect*"

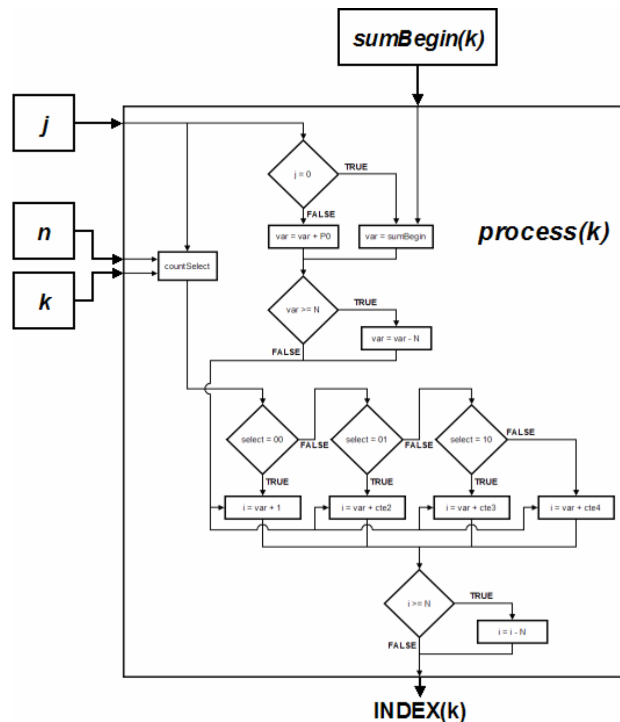


Ilustración 34. Interleaver paralelo implementable. Diagrama de flujo.

En la *ilustración 34* es presentado finalmente el diagrama de flujo que implementa el algoritmo paralelizado para un proceso *k* cualquiera, desde el primer proceso (*k = 0*) hasta el octavo y último (*k = 7*).

3.4.1.2. DISEÑO DE LA MEMORIA ROM. ENTIDAD '*VALUESN*'

Como se ha visto en la sección anterior, los parámetros necesarios que permiten realizar el algoritmo son los siguientes:

- ***n***: tamaño de subloque. Será introducido en memoria para así no tener que implementar una división. Atiende a la expresión:

$$n = \frac{N}{Pe}$$

- ***P0***, ***cte2***, ***cte3*** y ***cte4***: parámetros del interleaver dependientes del tamaño de bloque *N*. *P0* se encuentra en [7], los demás parámetros atienden a las expresiones:

$$cte2 = \mathrm{mod}\left(1 + \frac{N}{2} + P1, N\right)$$

$$cte3 = \mathrm{mod}(1 + P2, N)$$

$$cte4 = \mathrm{mod}\left(1 + \frac{N}{2} + P3, N\right)$$

- ***sumBegin(k)***: inicialización de la suma/producto de cada proceso, y por tanto existirá una por proceso. No se implementará este parámetro para el primer proceso (*k = 0*) pues para todos los casos será cero. Atenderá a la expresión:

$$sumBegin(k) = \mathrm{mod}\left(P0 \cdot (k-1) \cdot \frac{N}{Pe}, N\right)$$

Se comprueba que todos los parámetros a memorizar son dependientes de *N* y *Pe*, por lo que las salidas de la memoria dependerán exclusivamente de estos dos parámetros.

Es creado un script Matlab® llamado `parametersToLoad.m` que calcula y crea un archivo de texto con todos los parámetros expuestos según *N* y *Pe*. A partir de estos datos se calcula la longitud óptima para cada bus de salida y se crea el archivo VHDL de la memoria, llamado `valuesN.vhd`.

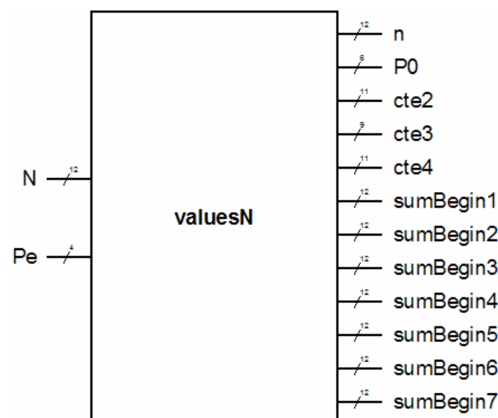


Ilustración 35. Entidad "*valuesN*"

En la *ilustración 35* se observan los puertos de entrada y salida de esta entidad, así como sus correspondientes anchuras de bus a tener en cuenta

3.4.1.3. ENTIDAD '*INTERLEAVERALGORITHM*'

En esta entidad (`interleaverAlgorithm.vhd`) se implementa el algoritmo interleaver en su versión modificada, anteriormente vista en la *ilustración 34*.

Pero no solamente se ha de implementar la generación de índices en *desorden*, sino también se debe contemplar la posibilidad de la generación de índices en *orden*, por lo que se ha de inferir un contador seguido de un sumador '$k \cdot n$' en cada entidad.

A continuación (*ilustración 36*) se muestra el esquema lógico que implementa el algoritmo interleaver en su versión modificada, junto con el contador y sumador que provee la generación de los índices en *orden*.

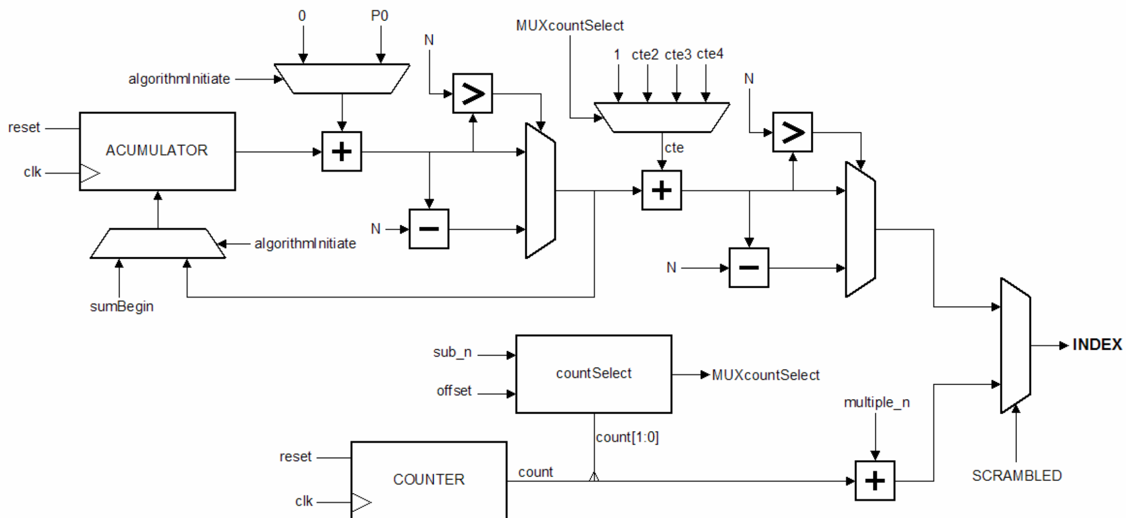


Ilustración 36. Esquema lógico de la entidad "*interleaverAlgorithm*".

La señal *offset* es equivalente a *k*, pues indica a la entidad el proceso al cual pertenece (desde el proceso 0 al proceso 7). La señal *multiple_n* representa el producto '$k \cdot n$', que será calculado exteriormente.

Por su parte, la señal *algorithmInitiate* toma el control de dos de los multiplexores para así escoger los parámetros correctos en el inicio del algoritmo. La señal de control síncrona *enable* congela la generación de índices hasta que vuelve a ser activada, será útil para implementar el mecanismo de *parada total*.

La sencilla maquina de estados que controla estas señales, así como el registro *acumulador* y el contador se presenta en la *ilustración 37*.

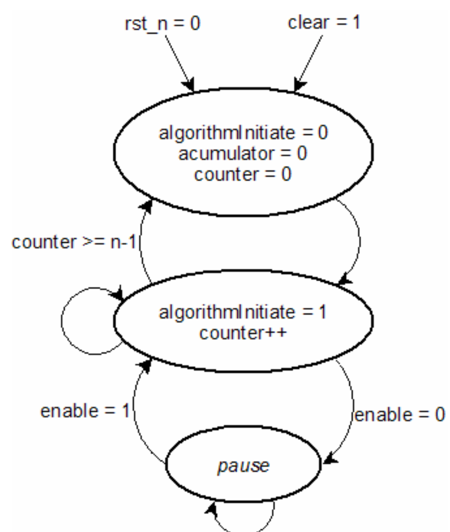


Ilustración 37. Maquina de estados de "interleaverAlgorithm".

En la *ilustración 38* se observa el aspecto como bloque funcional que presenta esta entidad.

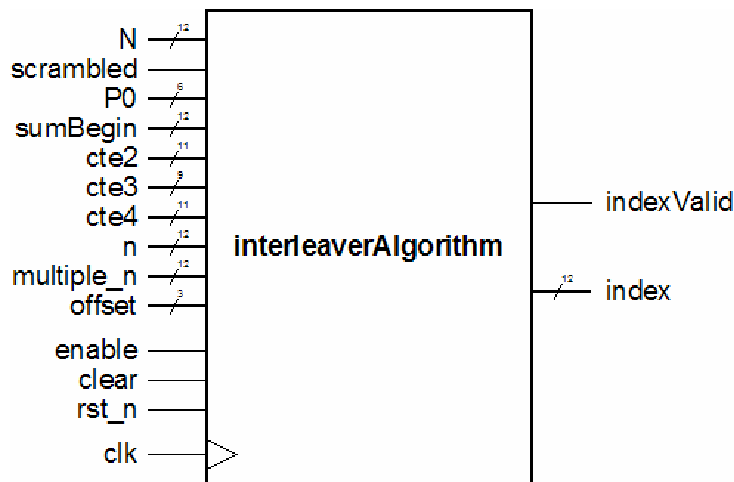


Ilustración 38. Entidad "*interleaverAlgorithm*".

3.4.2. GENERADOR DE ÍNDICES

3.4.2.1. ENTIDAD '*INDEXGENERATOR*'

El generador de índices (`indexGenerator.vhd`) alberga en su interior la memoria ROM anteriormente descrita y los ocho procesadores del algoritmo (*interleaverAlgorithm*).

Esta entidad solo se encargará del conexionado de estas entidades y de la generación de los múltiplos de *n* necesarios para los distintos procesos. Estos múltiplos son creados como se observa en la *ilustración 39*, haciendo únicamente uso de registros de desplazamiento y sumadores, que finalmente son conectados al array *multiple_n*, cuya primera posición (*multiple_n(0)*) estará conectada a tierra.

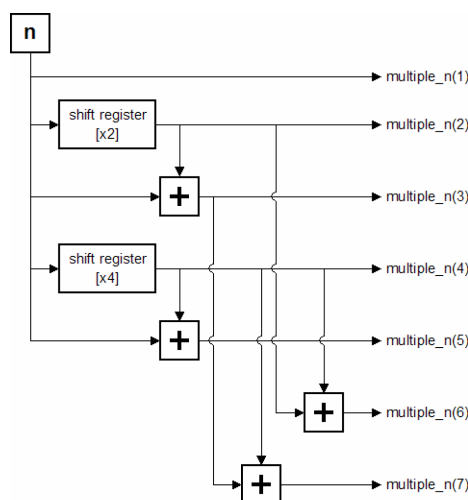


Ilustración 39. Implementación de los múltiplos de *n*.

Los múltiplos de *n* (de *2n* a *7n*), así como también *n*, se entregarán a la salida ya que el traductor de índices hará uso de ellos para la traducción. Formarán el bus *mult_nBus*.

Los índices generados por cada uno de los 8 procesos se encontrarán en el bus de salida *indexOut*, ocupando el *proceso 0* los 12 bits menos significativos y el *proceso 7* los 12 más significativos.

Para comprobar el conexionado interior de esta entidad acudir al código VHDL de ésta, que se encuentra en la segunda sección del anexo II, "*componentes y testbench VHDL"*.

A continuación (*ilustración 40*) se observa el bloque que define al generador de índices:

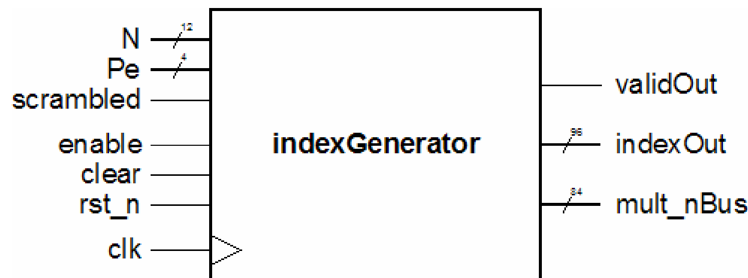


Ilustración 40. Entidad "*indexGenerator*".

3.4.2.2. TEST Y VALIDACIÓN

El generador de índices diseñado tiene un comportamiento definido: la generación secuenciada de índices de forma ordenada (*SCRAMBLED = 0*), o desordenada (*SCRAMBLED = 1*). Para cada uno de estos dos modos de trabajo son diseñados testbenchs que testean todas las posibles combinaciones de *N* y *Pe*.

tb_indexGenerator_report_ORDER.vhd (*SCRAMBLED = 0*)

tb_indexGenerator_report_SCRAMB.vhd (*SCRAMBLED = 1*)

Finalmente, cada testbench será validado por su correspondiente script Matlab® diseñado para tal fin:

verifORDER_indexGenerator.m (*SCRAMBLED = 0*)

verifSCRAMB_indexGenerator.m (*SCRAMBLED = 1*)

La validación llevada a cabo es positiva, la arquitectura de *indexGenerator* y de sus entidades interiores (*interleaverAlgorithm* y *valuesN*) es correcta.

3.4.3. TRADUCTOR DE ÍNDICES

Será la entidad encargada de traducir los índices generados en referencias a memoria y direcciones. Las 8 memorias serán numeradas de 0 a 7, para poder referenciarlas utilizando una codificación en binario puro de 3 bits.

3.4.3.1. ALGORITMO DE TRADUCCIÓN

En la sección 2.2.3 se expusieron unas expresiones matemáticas que calculaban a partir de cierto índice *j* la dirección y memoria a la que hace referencia este índice. Estas expresiones son:

$$\text{Referencia a memoria:} \ldots\ldots\ldots\ldots\ \text{fix}\left(\frac{j}{n}\right)$$

$$\text{Dirección:} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ \text{mod}(j, n)$$

Discusiones anteriores han dado a conocer que la implementación de la división y del módulo son altamente costosas, una en área consumida y otra en tiempo de proceso respectivamente. Es por esta razón que se ha de investigar un algoritmo óptimo, que evite en la medida de lo posible estas dos expresiones.

Como se vio anteriormente en la *ilustración 12* (sección 2.2.3), cada subloque es almacenado en las primeras posiciones de su correspondiente memoria. En un caso general se dispondrán de *N* bloques a dividir en *Pe* memorias, donde cada una almacenará *n* bloques, por lo que la relación entre índices, memorias y direcciones:

ÍNDICES	0	…	n-1	n	…	2n-1	….	N-1
MEMORIA	*memoria 0*			*memoria 1*			*memoria PE-1*	
DIRECCIÓN	0	…	n-1	0	…	n-1	…	n-1

Tabla 17. Relación genérica entre índices, memorias y direcciones.

Es posible realizar un algoritmo que calcule las referencias a memoria y direcciones basándose en comparaciones de los índices con los múltiplos del parámetro *n*. De esta forma, la comparación que debe realizarse sobre un índice *j* para seleccionar una memoria es:

$$k{\cdot}n <= j < (k+1){\cdot}n \qquad \text{memoria } k$$

Para calcular la dirección de la memoria *k* a la que pertenece el índice *j* se debe de normalizar éste a *n*, o lo que es lo mismo, restarle el múltiplo '*k·n*', con lo que la dirección es calculada.

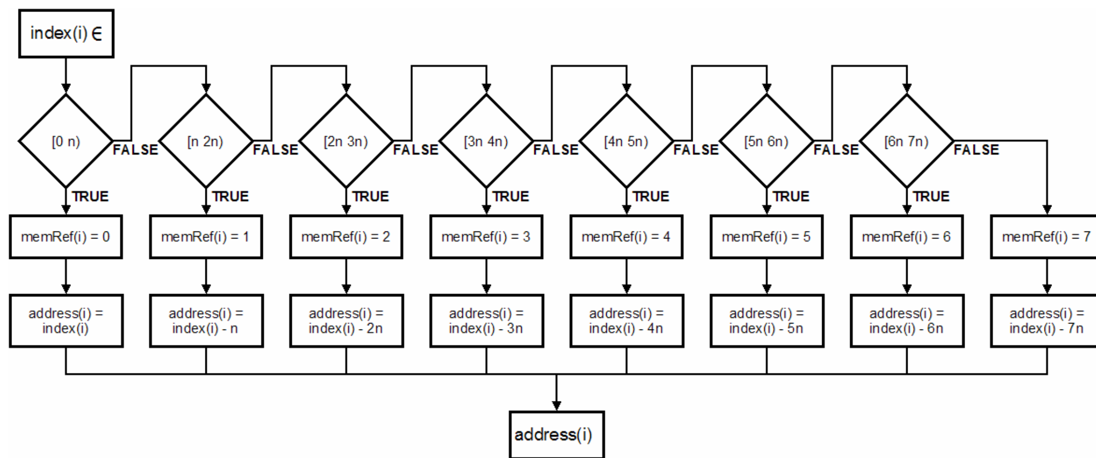


Ilustración 41. Algoritmo de traducción de índices.

Se muestra en la *ilustración 41* el diagrama de flujo que implementa este algoritmo, se observa que usando simplemente comparaciones y restas es posible resolver en un solo ciclo el problema de la traducción. Se ha comprobado también la necesidad de que el bloque *indexGenerator* proveyera de los múltiplos del parámetro *n*.

3.4.3.2. ENTIDAD '*ADDRESSTRANSLATE*'

Esta entidad asíncrona queda definida en el archivo `addressTranslator.vhd`. A continuación, *ilustración 42*, se dibuja el bloque que la representa:

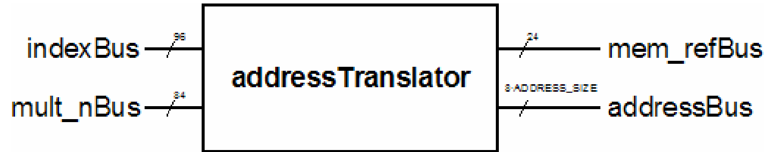


Ilustración 42. Entidad "*addressTranslator*".

Los dos buses de salidas contienen toda la información de direccionamiento de los 8 procesos, repartiéndose los bits como ya es habitual. De todas formas, la *tabla 18* muestra la partición de los buses:

PROCESO	MEM_REFBUS (bits)	ADDRESSBUS (bits)
7	[21 : 23]	[7·ADDRESS_SIZE : 8·ADDRESS_SIZE-1]
6	[18 : 20]	[6·ADDRESS_SIZE : 7·ADDRESS_SIZE-1]
5	[15 : 17]	[5·ADDRESS_SIZE : 6·ADDRESS_SIZE-1]
4	[12 : 14]	[4·ADDRESS_SIZE : 5·ADDRESS_SIZE-1]
3	[9 : 11]	[3·ADDRESS_SIZE : 4·ADDRESS_SIZE-1]
2	[6 : 8]	[2·ADDRESS_SIZE : 3·ADDRESS_SIZE-1]
1	[3 : 5]	[ADDRESS_SIZE : 2·ADDRESS_SIZE-1]
0	[0 : 2]	[0 : ADDRESS_SIZE-1]

Tabla 18. Buses de salida de "*addressTranslator*".

3.4.4. GENERADOR DE DIRECCIONES

3.4.4.1. ENTIDAD '*ADDRESSGENERATOR*'

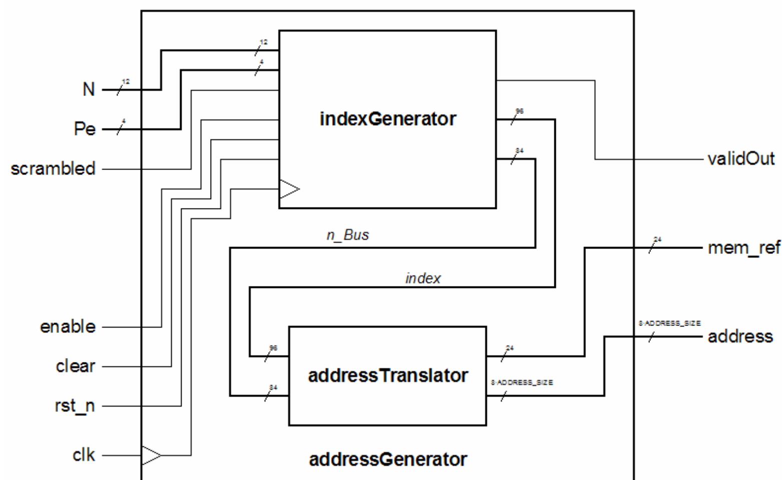


Ilustración 43. Entidad "*addressGenerator*".

Esta entidad (`addressGenerator.vhd`) contiene en su interior el generador de índices y el traductor a direcciones, por lo que con su creación se crea una unidad que genera de forma totalmente transparente al usuario el direccionamiento deseado a partir de los parámetros *N*, *Pe* y *SCRAMBLED*. Puede ser observado el bloque que la define en la *ilustración 43*.

3.4.4.2. TEST Y VALIDACIÓN

Puesto que el generador de direcciones posee un comportamiento definido se testeará la generación de direcciones y referencias a memoria en modo ordenado (*SCRAMBLED = 0*) y en modo desordenado (*SCRAMBLED = 1*). Para cada uno de estos dos modos de trabajo son diseñados testbenchs que testean todas las posibles combinaciones de *N* y *Pe*.

`tb_adxGen_ORDER.vhd` (*SCRAMBLED = 0*)

`tb_adxGen_SCRAMB.vhd` (*SCRAMBLED = 1*)

Finalmente, cada testbench será validado por su correspondiente script Matlab® diseñado para tal fin:

`verifORDER_adxGen.m` (*SCRAMBLED = 0*)

`verifSCRAMB_adxGen.m` (*SCRAMBLED = 1*)

La validación llevada a cabo de *addressGenerator* es correcta.

3.4.5. MEMORIA LIFO DOBLE

La función del interleaver, como se vio anteriormente, es leer datos de la memoria como escribirlos en ella una vez hayan sido procesados. En la sección 3.1.1 se describió el comportamiento del interleaver en el que se observaron las siguientes características:

- Los datos serán entregados por las SISOs (recursión *beta*) en orden inverso a como fueron leídos, es decir, del último al primero.
- Se leen/escriben bloques de datos llamados *ventanas*, cuyo tamaño máximo será 60 (*recomendación del VLSI del Politecnico di Torino*).
- La lectura de datos de la ventana *k* es contemporánea a la escritura de datos de la ventana anterior *k-1*.

Se podría proveer al interleaver de un segundo generador de direcciones que realizase los cálculos necesarios para escribir los datos procesados por las SISOs, pero es una solución de alto coste frente a una pequeña memoria temporal donde guardar las direcciones y referencias generadas en la lectura para usarlas a posteriori en escritura.

La entrega de datos en orden inverso en la recursión *beta* hace ideal el uso de una pila LIFO como elemento de memoria, pues la lectura inversa de datos se realiza de forma automática debido a su naturaleza (*Last-Input, First-Output*).

La pila LIFO almacena el direccionamiento producido en la recursión *alfa* (lectura), mientras que en la recursión *beta* (escritura) se extrae el direccionamiento almacenado anteriormente en ésta. Puesto que las dos recursiones coexisten en el mismo instante temporal, se provee de una doble LIFO para evitar que el direccionamiento producido en la

recursión *alfa* la ventana *k* se mezcle con el que se ha de leer en la recursión *beta* de la ventana *k-1* (*ilustración 24*).

3.4.5.1. ARQUITECTURA

Según las especificaciones anteriores, se dispondrá de una doble pila LIFO con un tamaño mínimo de 60 posiciones capaz de almacenar en ella las direcciones y referencias de los 8 procesos.

Se sobredimensionará el tamaño de la pila a 64 posiciones en lugar de 60 con lo que se deja un pequeño margen a posibles errores futuros relacionados con el enventanado de datos.

El tamaño de palabra de los datos a almacenar será de *8·ADDRESS_SIZE+24*, donde los 24 bits menos significativos pertenecen a las referencias de memoria de cada proceso (3 bits por proceso), y el resto a la dirección de cada uno de los 8 procesos. En la *tabla 19* se muestra la distribución que siguen estos bits.

	PROCESO	**BITS**
Direcciones	7	[8·ADDRESS_SIZE+23 : 7·ADDRESS_SIZE+23]
	⋮	⋮
	0	[ADDRESS_SIZE+23 : 24]
Referencias a memoria	7	[23:21]
	⋮	⋮
	0	[2:0]

Tabla 19. Distribución de las direcciones y las referencias a memoria en la palabra almacenada en la pila.

En la *ilustración 44* se muestra la arquitectura que implementa la doble pila LIFO. Gracias al conmutador se leen y escriben datos en pilas distintas y, llegado el momento, es conmutado el orden de éstas.

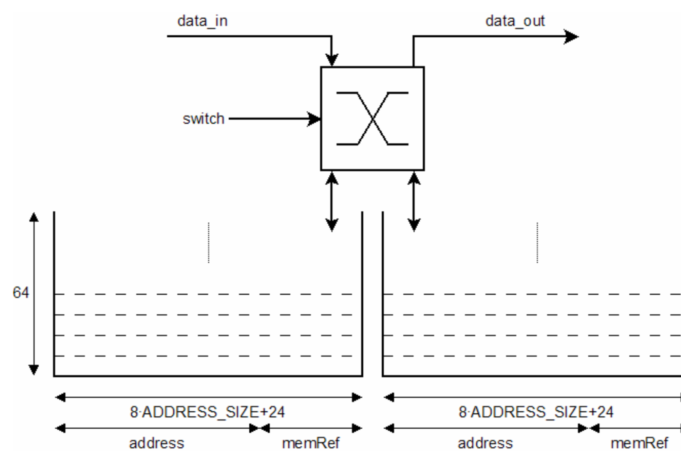


Ilustración 44. Arquitectura de la doble memoria LIFO.

3.4.5.2. CONDICIONES DE CONMUTACIÓN

El direccionamiento producido en el generador de direcciones será introducido en una pila, mientras que de la otra serán leidos los producidos en la ventana anterior. Al terminar el procesamiento de la ventana en curso una pila se encontrará vacía mientras que la otra estará llena, es entonces cuando llega el momento de conmutación de lectura/escritura de las pilas.

En la *ilustración 45* se observa la evolución de ocupación de las LIFO. La transición *A* corresponde a la finalización de procesamiento de una ventana, con lo que se ha de dar la conmutación de lectura/escritura (transición *B*).

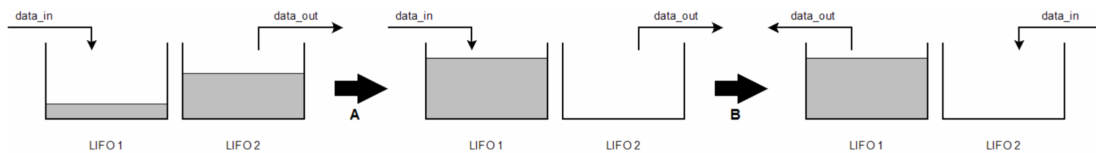


Ilustración 45. Evolución de ocupación de las pilas LIFO.

La conmutación (transición *B*) coincidirá en la mayoría de los casos con el inicio del procesamiento de una nueva ventana, que es gobernado por la señal exterior de control *START_READ*. Pero no siempre es así, pues la última recursión *beta* no corresponderá con el inicio del procesamiento de una ventana, pues ya todas han sido procesadas; sin embargo, en éste caso se da la condición de que se deseará leer de una pila vacía. Luego la expresión lógica que dicta la conmutación de las pilas es:

"*PROCESAMIENTO NUEVA VENTANA*" OR "*LECTURA DE UNA PILA VACÍA*"

3.4.5.3. ENTIDAD 'LIFOMULTIPLE'

La entidad definida por el archivo `LIFOmultiple.vhd` implementa el comportamiento de las pilas LIFO hasta ahora visto. Dispone de una habilitación de lectura, la cual presenta los datos en salida un ciclo después de haber sido activada; también se dispone de una habilitación de escritura de los datos de entrada (*data_in*). Se proporciona a la entidad de una señal de conmutación de pilas, *LIFO_change*, que es usada cuando se ha de conmutar las pilas por el comienzo del procesamiento de una nueva ventana (señal externa *START_READ*), debe de estar un ciclo en alta.

En la *ilustración 46* se observa el bloque funcional que representa esta entidad.

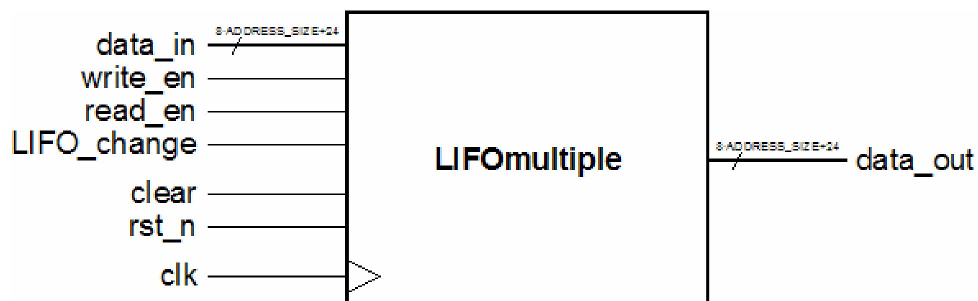


Ilustración 46. Entidad "*LIFOmultiple*".

3.4.6. CONMUTADOR DE LECTURA

Cada proceso ha de leer de distintas memorias en diversos instantes de tiempo, por lo que es necesario que exista una conmutación de datos entre la memoria y el interleaver, para que de esta forma se entreguen correctamente los datos a las SISOs. Además, ésta entidad estará preparada para soportar el mecanismo de *parada total*, adoptado para solventar las colisiones en lectura.

3.4.6.1. ARQUITECTURA

Como componente principal de la arquitectura se tiene una matriz de conmutación que realizará el encaminamiento de los datos hacia las SISOs correspondientes. El conmutador dispone de entradas de habilitación y salidas de validación, pues no siempre serán usados todos sus canales.

En la *ilustración 47* se observa la arquitectura implementada.

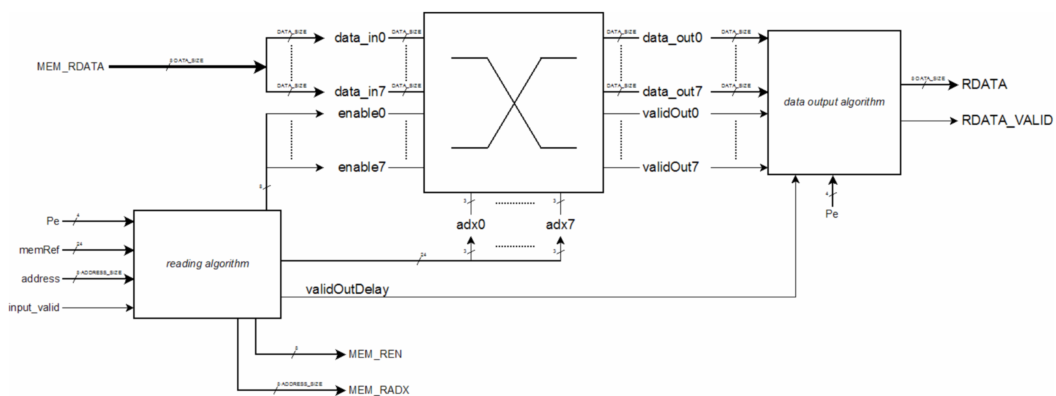


Ilustración 47. Arquitectura del conmutador de lectura.

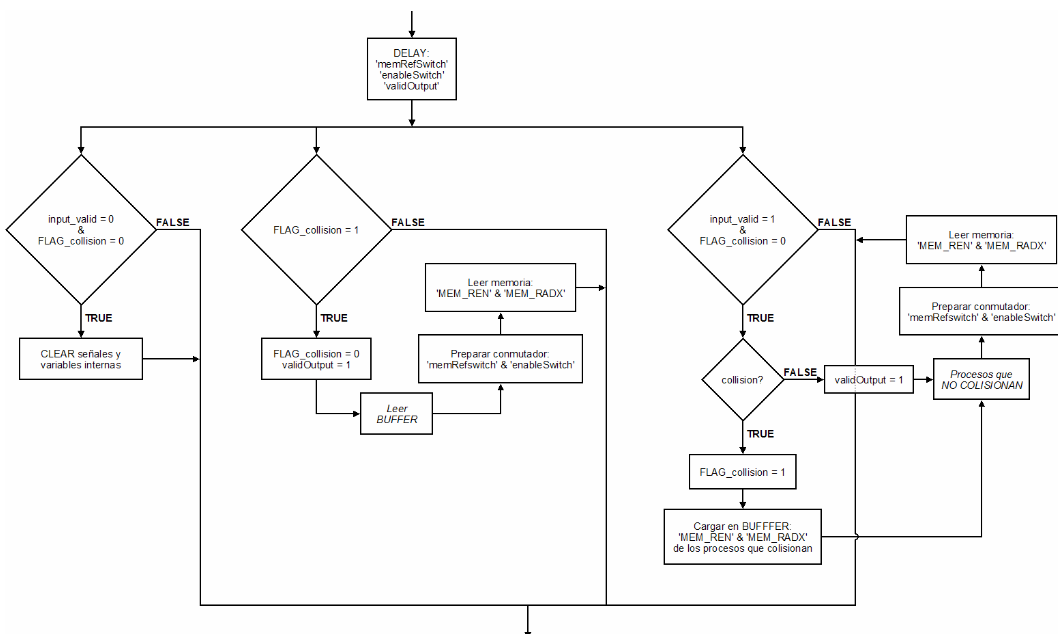


Ilustración 48. Algoritmo de lectura.

Además del conmutador central existen otros dos componentes aún más importantes que éste: el *algoritmo de lectura* y el *algoritmo de salida de datos*.

El *algoritmo de lectura* es presentado en la *ilustración 47*. Gracias a éste se envían las correspondientes habilitaciones y direcciones de lectura a la memoria principal, también es el responsable del correcto funcionamiento del conmutador. Este algoritmo detecta de manera automática cuando se produce una colisión, por lo que los procesos colisionantes son almacenados en un buffer temporal a la espera del siguiente ciclo para ser procesados; mientras tanto, los procesos que no han colisionado son procesados. Es una unidad externa (*unidad central del interleaver*) que será analizada más adelante, la que se encargará de detener al generador de direcciones cuando se produzca una colisión, por lo que a esta entidad se le puede considerar *pasiva* frente a las colisiones.

Cuando el conmutador indica a través de sus salidas de validación que los datos presentes son válidos entra funcionamiento el *algoritmo de salida de datos* (*ilustración 49*). Este sencilla algoritmo dispone en el bus de salida *RDATA* los datos que son válidos y cuando todos ellos (según el grado de paralelismo *Pe*) lo son, activa la señal de validación *RDATA_VALID*. El control sobre la validez de los datos en salida es controlada el *algoritmo de lectura*.

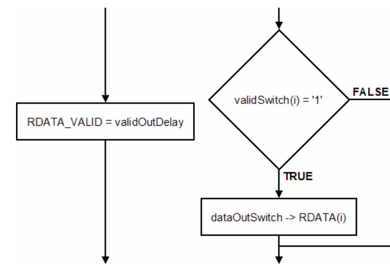


Ilustración 49. Algoritmo de salida de datos.

3.4.6.2. ENTIDAD '*SWITCHMATRIX*'

La entidad implementada en el archivo `switchMatrix.vhd` no se encuentra en el anexo II debido a su amplia descripción. Si el lector desea conocer más de cerca el código de esta entidad debe acudir al formato electrónico del proyecto.

En la *ilustración 50* se presenta su diagrama de bloque. Se observan las señales de entrada de datos (*data_in*) y habilitación de éstas (*enable*) a la izquierda; en la parte inferior las entradas de direccionamiento (*adx*) de cada canal, y a la derecha las validaciones (*validOut*) de las salidas de datos (*data_out*).

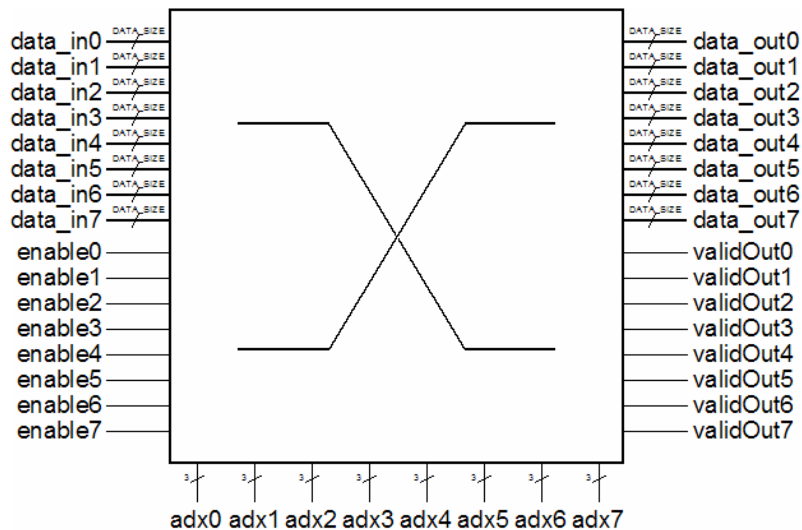


Ilustración 50. Entidad "*switchMatrix*".

3.4.6.3. ENTIDAD '*SWITCHREADMEMORY*'

La entidad hasta ahora descrita y presentada en el archivo `switchReadMemory.vhd` en el anexo II, corresponde con el bloque de la *ilustración 51*.

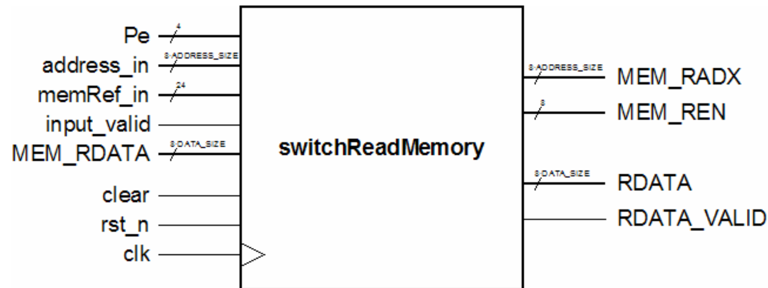


Ilustración 51. Entidad "*switchReadMemory*".

Las señales *MEM_RADX* y *MEM_REN* son las encargadas de habilitar la memoria para leer en una dirección, devolviendo los datos por *MEM_RDATA*. La distribución de los bits en estas señales se muestra en la siguiente tabla:

MEMORIA	***MEM_REN***	***MEM_RADX***	***MEM_RDATA***
0	[0]	[ADDRESS_SIZE-1 : 0]	[DATA_SIZE-1 : 0]
1	[1]	[2·ADDRESS_SIZE-1 : ADDRESS_SIZE]	[2·DATA_SIZE-1 : DATA_SIZE]
2	[2]	[3·ADDRESS_SIZE-1 : 2·ADDRESS_SIZE]	[3·DATA_SIZE-1 : 2·DATA_SIZE]
3	[3]	[4·ADDRESS_SIZE-1 : 3·ADDRESS_SIZE]	[4·DATA_SIZE-1 : 3·DATA_SIZE]
4	[4]	[5·ADDRESS_SIZE-1 : 4·ADDRESS_SIZE]	[5·DATA_SIZE-1 : 4·DATA_SIZE]
5	[5]	[6·ADDRESS_SIZE-1 : 5·ADDRESS_SIZE]	[6·DATA_SIZE-1 : 5·DATA_SIZE]
6	[6]	[7·ADDRESS_SIZE-1 : 6·ADDRESS_SIZE]	[7·DATA_SIZE-1 : 6·DATA_SIZE]
7	[7]	[8·ADDRESS_SIZE-1 : 7·ADDRESS_SIZE]	[8·DATA_SIZE-1 : 7·DATA_SIZE]

Tabla 20. Distribución de los bits en los buses de memoria.

Mientras tanto, las señales de direccionamiento de entrada producidas en el generador de direcciones, *address_in* y *memRef_in*, siguen el mismo esquema que el presentado en la *tabla 19*.

3.4.6.4. TEST Y VALIDACIÓN

Una hipotética entidad de orden superior conformada por el generador de direcciones y esta nueva entidad debería de ser capaz de realizar correctamente la lectura de las 8 distintas memorias y entregar correctamente los datos a las SISOs, superando el escollo que representan las colisiones; siempre y cuando en esa hipotética entidad sea implementado el mecanismo de *parada total*. Esta hipotética entidad será creada por un testbench que testeará el paso de datos desde 8 memorias (también creadas en el testbench) hacia las SISOs.

Como se ha venido realizando hasta el momento, serán dos los testbenchs creados para testar todas las posibles combinaciones de *N* y *Pe*, uno que testeará al conmutador de lectura en modo ordenado (*SCRAMBLED = 0*) y otro para testearlo en modo desordenado (*SCRAMBLED = 1*):

`tb_SRM_ORDER.vhd` (*SCRAMBLED = 0*)

`tb_SRM_SCRAMB.vhd` (*SCRAMBLED = 1*)

El testbench del modo desordenado, `tb_SRM_SCRAMB.vhd`, implementa el mecanismo de *parada total* para los casos conocidos de colisión. El conexionado de las dos entidades en test es observado en la *ilustración 52*.

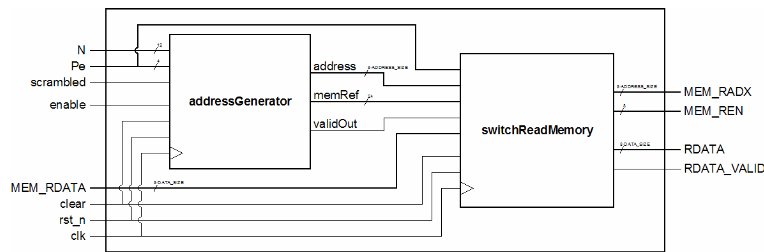


Ilustración 52. Testbench del conmutador de lectura. Entidad a test.

Finalmente, cada testbench es validado por su correspondiente script Matlab® diseñado para tal fin:

`verifORDER_SRM.m` (*SCRAMBLED = 0*)

`verifSCRAMB_SRM.m` (*SCRAMBLED = 1*)

La validación llevada a cabo de la entidad de la *ilustración 52* es correcta. Se puede concluir que el conmutador de lectura funciona correctamente así como que el mecanismo de *parada total* implementado.

3.4.7. CONMUTADOR DE ESCRITURA

Una vez que las SISOs han completado su trabajo han de devolver los datos procesados a memoria, por lo que al igual que ocurría en lectura, se hace necesaria la conmutación de los datos hacia su memoria correcta. Dentro del conmutador se implementará una arquitectura TIBB capaz de absorber las colisiones mediante el uso de buffers previos a las memorias. El direccionamiento de los datos hacia la memoria es proporcionado por la pila LIFO anteriormente discutida.

3.4.7.1. ARQUITECTURA

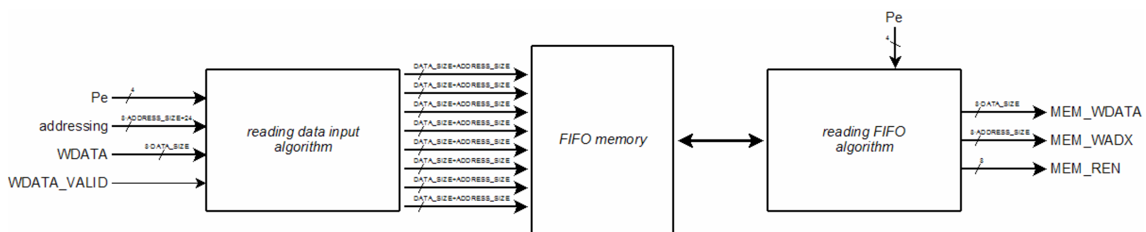


Ilustración 53. Arquitectura del conmutador de escritura.

Esta vez no será implementada una matriz de conmutación, sino que el propio código inferido será capaz de conmutar los datos provenientes de las SISOs hacia el buffer de la memoria correspondiente. Los buffers antepuestos a cada memoria atenderán a un comportamiento de pila FIFO (*First-Input, First-Output*).

Como se observa en la arquitectura del conmutador (*ilustración 53*) existen dos algoritmos principales: el *algoritmo de lectura de datos de entrada* y el *algoritmo de lectura de FIFO*. El *algoritmo de lectura de datos de entrada* (*ilustración 54*) es el encargado de sincronizar los datos provenientes de las SISOs (*WDATA*) con la lectura del direccionamiento (*addressing*) correspondiente almacenado en la pila LIFO, el cual tendrá un retardo de un ciclo. Una vez que se disponen de direcciones y referencias a memoria, se conmutan datos y direcciones -donde han de ser escritos éstos- hacia la FIFO de la memoria que corresponda.

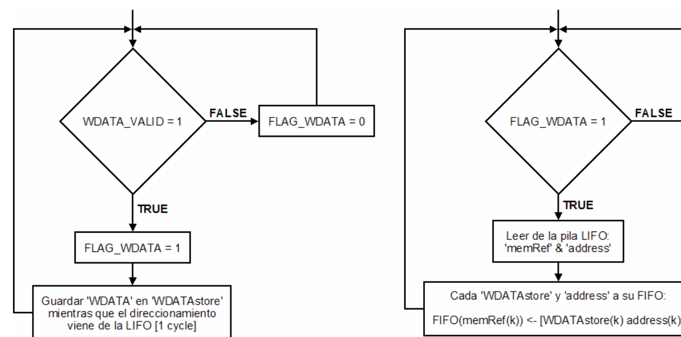


Ilustración 54. algoritmo de lectura de datos de entrada.

Cada FIFO almacena datos que han de ser escritos en memoria, así como la dirección donde han de ser escritos. La distribución de éstos dentro de cada FIFO se muestra a continuación en la *ilustración 55*:

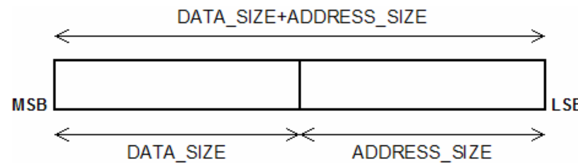


Ilustración 55. Distribuciión de datos y direcciones en FIFO.

El estado de ocupación de las FIFOs (vacías o no vacías) se conoce a través de unos bits de estado denominados *FIFOread[7:0]*. Cuando alguno de estos toma el valor 1 indica que el buffer correspondiente a su posición tiene algún elemento en su interior, y será entonces cuando entre en funcionamiento el *algoritmo de lectura de FIFO*, presentado en la *ilustración 56*. Este sencillo algoritmo extrae de la FIFO no vacía la dirección y dato almacenados en su interior y los escribe en su correspondiente memoria; y si por el contrario la

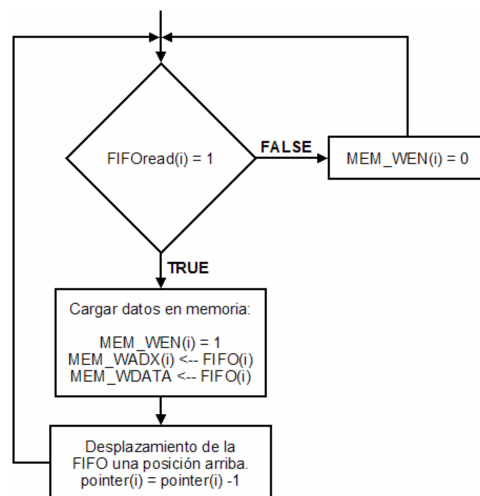


Ilustración 56. Algoritmo de lectura de FIFO.

FIFO se encuentra vacía, la habilitación de escritura de la memoria correspondiente será desactivada.

3.4.7.2. ENTIDAD '*SWITCHWRITEMEMORY*'

La arquitectura vista queda plasmada sobre la implementación VHDL llevada a cabo en el archivo presente en el anexo II llamado `switchWriteMemory.vhd`.

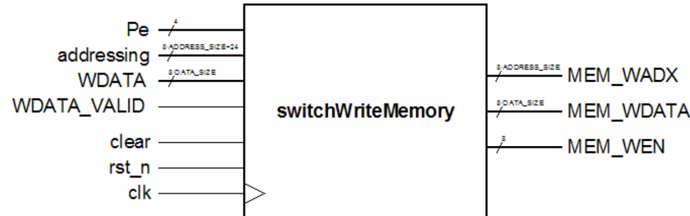


Ilustración 57. Entidad "*switchWriteMemory*".

En esta entidad se infiere como un genérico el tamaño de la FIFO que se implementa, aunque el tamaño mínimo para absorber las colisiones ha de ser de 6, tal y como se vio en el capítulo anterior. La señal de direccionamiento proveniente de la pila LIFO (*addressing*) posee una distribución de bits como la vista anteriormente en la *tabla 19*. Los datos provenientes de las SISOs se encuentran en el bus *WDATA*, el cual atiende a la siguiente distribución (*tabla 21*):

SISO	***WDATA***
0	[DATA_SIZE-1 : 0]
1	[2·DATA_SIZE-1 : DATA_SIZE]
2	[3·DATA_SIZE-1 : 2·DATA_SIZE]
3	[4·DATA_SIZE-1 : 3·DATA_SIZE]
4	[5·DATA_SIZE-1 : 4·DATA_SIZE]
5	[6·DATA_SIZE-1 : 5·DATA_SIZE]
6	[7·DATA_SIZE-1 : 6·DATA_SIZE]
7	[8·DATA_SIZE-1 : 7·DATA_SIZE]

Tabla 21. Distribución de los bits en el bus *WDATA*.

Los tres buses de salida de la entidad (*MEM_WADX*, *MEM_WDATA* y *MEM_WEN*) adoptan la misma distribución de bits que la presentada anteriormente en la *tabla 20*, salvaguardando las diferencias de memorias, unas de escritura y otras de lectura.

3.4.7.3. TEST Y VALIDACIÓN

Al igual que se realizó con el conmutador de lectura, se diseñará una entidad de prueba que verifique el correcto funcionamiento del conmutador de escritura. Esta entidad de test es la mostrada en la *ilustración 58*. Se observa que se implementa una pila LIFO, la cual no corresponde con la pila doble diseñada en la sección 3.4.5.

Como viene siendo habitual, serán dos los testbenchs creados para testar todas las posibles combinaciones de *N* y *Pe*, uno que testeará al conmutador de escritura en modo ordenado (*SCRAMBLED = 0*) y otro para testearlo en modo desordenado (*SCRAMBLED = 1*):

tb_SWM_ORDER.vhd (*SCRAMBLED = 0*)

tb_SWM_SCRAMB.vhd (*SCRAMBLED = 1*)

Finalmente, cada testbench es validado por su correspondiente script Matlab® diseñado para tal fin:

verifORDER_SWM.m (*SCRAMBLED = 0*)

verifSCRAMB_SWM.m (*SCRAMBLED = 1*)

La validación llevada a cabo es correcta, por lo que la arquitectura ITV y la implementación del conmutador de escritura son válidos.

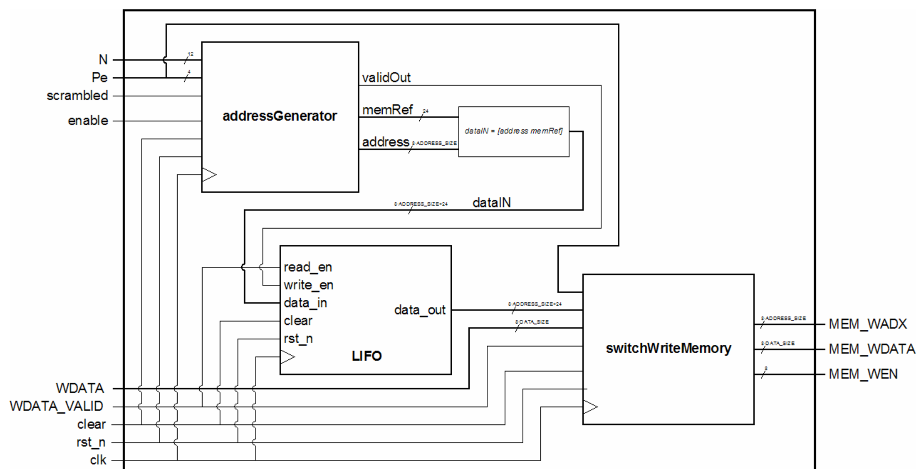


Ilustración 58. Testbench del conmutador de escritura. Entidad a test.

3.4.8. UNIDAD DE CONTROL CENTRAL

El fin de la unidad de control es el manejo del generador de direcciones para la lectura de datos, respetando el enventanado que rige el proceso e implementando el mecanismo de *parada total* en caso de colisión. Las señales exteriores que rigen la naturaleza de la lectura son las siguientes:

- **start_read**: dispara el proceso de lectura.
- **burst_length**: indica el tamaño de la ventana a procesar.
- **clear_addGen**: indica el comienzo de un nuevo bloque de *N* datos, reseteando el generador de direcciones.

Las entradas disponibles en el generador de direcciones capaces de controlar su comportamiento son: *enable* (habilitación), *clear* y *rst_n* (reset síncrono y asíncrono respectivamente). Sincronizando oportunamente estas señales se implementa una máquina de estados capaz de:

- Dar comienzo a la lectura de un nuevo bloque.
- Realizar la lectura de datos ventana a ventana

- Implementar el mecanismo de parada total en lectura en caso de colisión.

3.4.8.1. CRONOGRAMA DE LECTURA

Las señales externas *clear_addGen* y *start_read* controlarán la lectura de los *N* bloques de datos mediante el enventanado dictado por la señal *burst_length*. En la *ilustración 59* se observa un cronograma de éstas señales.

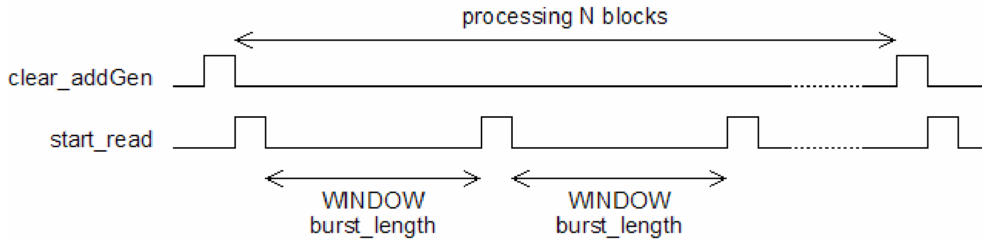


Ilustración 59. Cronograma de las señales de lectura *clear_addGen*, *start_read* y *burst_length*.

Entre dos *clear_addGen* consecutivos se produce la lectura de los *N* bloques a procesar. También es observado que entre *start_read* y *start_read* una ventana de *burst_length* elementos es procesada, hasta la llegada de un *clear_addGen* que reseteará el generador de direcciones para un nuevo comienzo de lectura. Así, después de haber recibido un *start_read* el generador de direcciones debe de activarse los ciclos suficientes como para producir un total de *burst_length* direcciones. Con esta información se puede diseñar la maquina de estados capaz de controlar el proceso.

3.4.8.2. MAQUINA DE ESTADOS

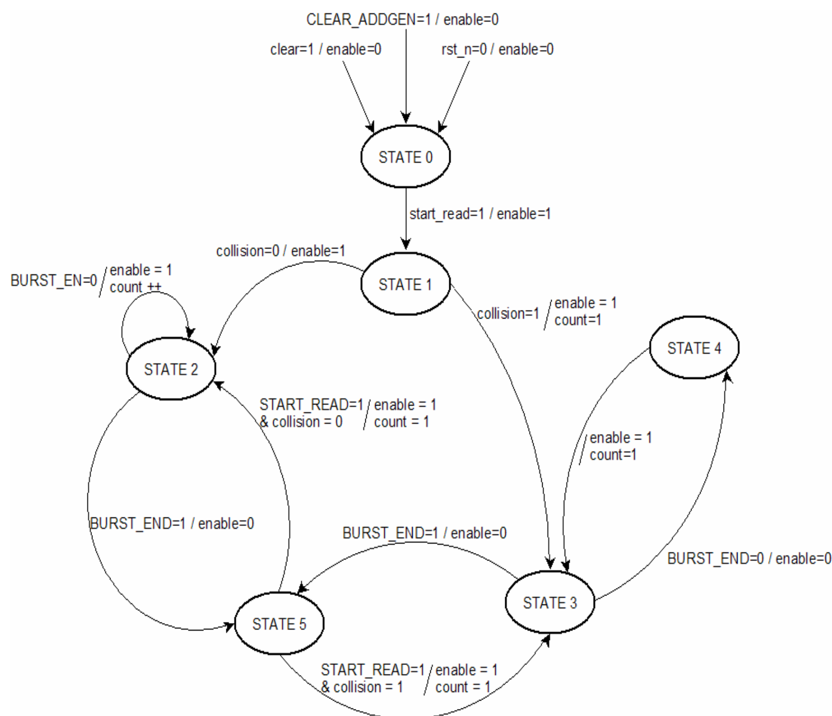


Ilustración 60. Maquina de estados de la unidad de control.

Del análisis de la arquitectura del *algoritmo interleaver* se deduce que siempre será tomado un ciclo ocioso en la inicialización de éste, y después de éste, cada ciclo que se encuentre activado el generador supondrá la producción de una dirección de lectura. Con esta información es diseñada la maquina de estados (*ilustración 60)*.

Esta maquina de estados implementa el comportamiento de la lectura enventanada descrito hasta ahora. También implementa el mecanismo de *parada total*: cuando se detecta que es posible que se produzca una colisión (variable *collision* a 1) la habilitación del generador de direcciones será alternada, ciclos activos y ciclos ociosos, parando así al generador de direcciones después de la producción de direcciones colisionantes. La variable interna *collision* responde a la siguiente expresión:

if (scrambled = 1 &
    (N = 108 & Pe = 2) or
    (N = 108 & Pe = 4) or
    (N = 108 & Pe = 6) or
    (N = 72 & Pe = 8))   → *collision* = 1
else   → *collision* = 0

La variable interna de la maquina de estados *BURST_END* realiza el recuento de las direcciones totales producidas en el generador, tomando el valor 1 cuando se hayan generado el número de direcciones justas como para completar una ventana.

3.4.8.3. ARQUITECTURTA

La unidad de control central se encuentra descrita en la entidad que define a todo el interleaver, `interleaver.vhd`, no realizando para ella una entidad especial. En la *ilustración 61* se observa el bloque que representa a ésta unidad.

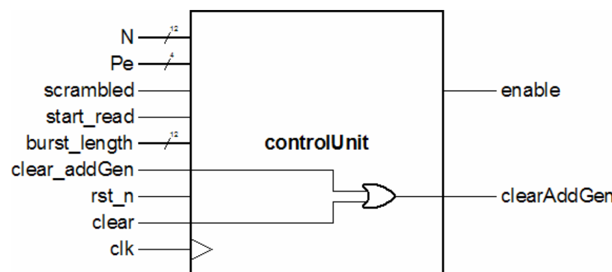


Ilustración 61. Unidad de control central.

Se implementa en ella la maquina de estados vista anteriormente, así como una puerta OR para el reseteo síncrono del generador de direcciones.

3.4.9. INTERLEAVER. CONEXIONADO DE ENTIDADES.

En la siguiente página se observa el conexionado de todas las entidades hasta ahora descritas (*ilustración 62*). El archivo que define este conexionado, así como la unidad de control es `interleaver.vhd`, presente en el anexo II.

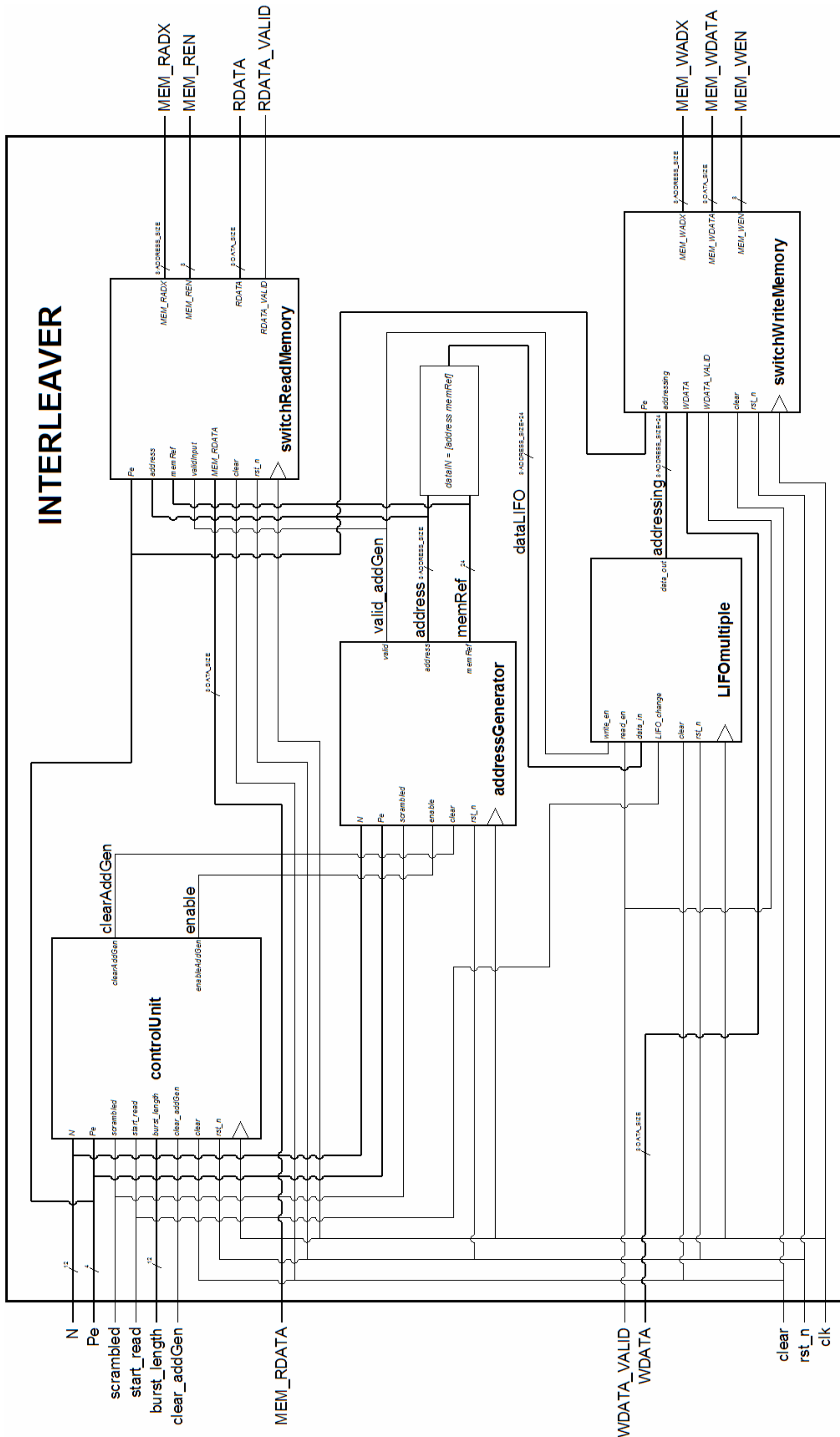


Ilustración 62. Interleaver. Conexiones interiores.

3.5. INTERLEAVER. TEST Y VALIDACIÓN.

Diseñado el interleaver, sólo resta su validación para dar por concluida la etapa de diseño. Como se ha venido realizando hasta ahora, se creará un testbench capaz de testear todas las posibles combinaciones de *N* y *Pe*, pero en esta ocasión serán introducidos algunos cambios: *la lectura enventanada*. El testbench ejecutará la lectura de datos enventanada, por lo que para cada valor de *N* se define un tamaño de ventana:

N	***TAMAÑO VENTANA***	***N***	***TAMAÑO VENTANA***
24	1	192	8
36	3	216	9
48	2	240	10
72	3	480	20
96	4	960	40
108	9	1440	60
120	5	1920	40
144	6	2400	50
180	5		

Tabla 22. Testbench interleaver. Tamaños de ventanas según *N*.

Cada ventana será procesada en orden inverso a su anterior, es decir, si la ventana *k* ha sido procesada en orden (*SCRAMBLED = 0*) la ventana *k+1* será procesada en orden alterado (*SCRAMBLED = 1*).

El testbench simulará por separado el flujo de datos de lectura y de escritura, por lo que para ambos procesos se conocerán los datos iniciales y se registraran sobre un archivo de salida los datos procesados. El testbench es presentado en el anexo II, con el nombre `tb_interleaver.vhd`. Para verificar los archivos generados por el testbench se crea el script Matlab® `verif_INTERLEAVER.m` presentado en el anexo I.

Tras la simulación del interleaver y la posterior verificación de ésta se comprueba que el diseño del interleaver es válido, así pues, ***el interleaver diseñado es correcto***.

4. RESULTADOS

4.1. RESULTADOS DE SÍNTESIS

Tras una síntesis en la FPGA Virtex II-Pro (XC2VP100) de Xilinx, y haciendo uso del software Synplify Pro® de Synplicity se obtienen los siguientes resultados:

Máxima frecuencia	29,4 MHz
LUTs totales	22139 (24%)
Registros (bits)	17500 (19%)

Tabla 23. Resultados de síntesis.

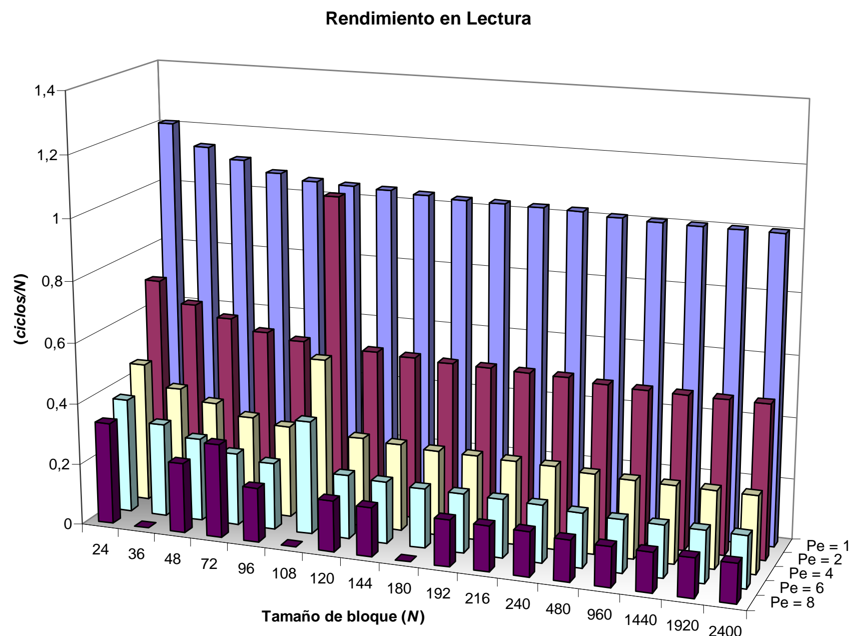
4.2. RENDIMIENTO

La forma en la que será expuesto el rendimiento será independiente de la tecnología final en la cual se implemente el *interleaver*, para ello se representará cada par *N-Pe* frente al cociente:

$$\frac{\text{ciclos consumidos}}{\text{N}}$$

Este cocientes es inversamente proporcional a la bondad del rendimiento, cuanto más próximo a cero sea el interleaver presentará un mejor rendimiento. De esta forma es rápidamente observable las prestaciones del interleaver para cada par *N-Pe*. Se hará distinción entre el rendimiento en lectura y el rendimiento en escritura, pues cada uno implementa una filosofía diferente del tratamiento de colisiones.

4.2.1. LECTURA

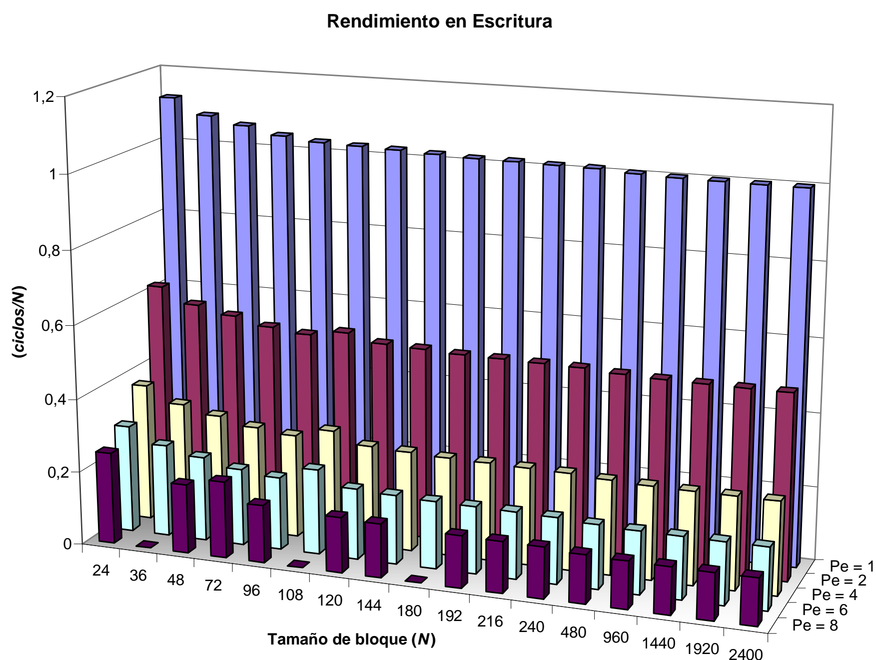


Gráfica 3. Rendimiento en lectura.

Se observa como conforme aumenta el tamaño del bloque, así como el grado de paralelismo el rendimiento del interleaver es mayor. En los casos en los cuales existe colisión (N = 108 y Pe = 2, 4 y 6; N = 72 y Pe = 8) existe un repuntamiento de la tendencia global a la baja debido a la necesidad de entregar los datos contemporáneamente a las SISOs. Este efecto es más palpable para Pe = 2 y N = 108, por lo que se aconsejaría al usuario final no seleccionar este grado de paralelismo para procesar bloques de tamaño N = 108.

4.2.2. ESCRITURA

En escritura no se muestra ningún empeoramiento grave del rendimiento debido a las colisiones, esto es debido a la no necesidad (e imposibilidad) de escribir los datos en las memorias contemporáneamente a su producción.



Gráfica 4. Rendiemiento en escritura.

5. CONCLUSIONES

Partiendo del análisis detallado del algoritmo interleaver presentado en el estándar y aplicando la conocida filosofía del "*divide y vencerás*" se ha diseñado un interleaver paralelo para ser introducido en el corazón de un decodificador de canal WiMax para turbo códigos.

El interleaver es capaz de dividir hasta en 8 partes el procesamiento de los datos. Paralelizar el interleaver no siempre es la panacea pues se suelen producir los mayores cuellos de botella de los decodificadores: las colisiones en memoria. Sin embargo, tras un arduo y detallado estudio se ha conseguido controlar el manejo de las colisiones para maximizar el rendimiento final del interleaver pero sin caer en el abuso de recursos, de forma que con los mínimos recursos posibles se obtiene la mínima latencia posible.

El interleaver formará parte de un decodificador que implementará alguno de los algoritmos de decodificación MAP vistos al inicio, soportando un enventanado máximo de hasta 60 datos.

La implementación del interleaver sobre un lenguaje de inferencia hardware como es el VHDL hace posible su portabilidad a cualquier tipo de tecnología, desde las típicas y socorridas FPGAs, hasta un diseño 100% ASIC.

6. REFERENCIAS

[1] C. Berrou, A. Glavieux, P. Thitimajshima, “**Near Shannon Limit Error. Correcting Coding and Decoding: Turbo Codes**”, 1993 IEEE.

[2] C. Berrou, M. Jézéquel, “**Non-Binary Convolutional Codes for Turbo Coding**”, Electronics Letters, 7 January 1999, Vol. 35, Nº 1.

[3] C. Berrou, M. Jézéquel, C. Douillard, S. Kerouédan, “**The Adventages of Non-Binary Turbo Codes**”, 2006, IEEE.

[4] C. Zhan, T. Arslan, “**An Efficient Decoder Scheme for Double Binary Circular Turbo Codes**”, 2001, IEEE.

[5] B. Baumgartner, M. Reinhardt, G. Richter, M. Bossert, “**Performance of Forward Error Correction for IEEE 802.16e**”, 2005, IEEE.

[6] **IEEE 802.16-2004**. IEEE Standard for Local and Metropolitan area Networks. Part 16: Air Interface for Fixed Broadband Wireless Access Systems.

[7] **IEEE 802.16e2005**. IEEE Standard for Local and Metropolitan area Networks. Part 16: Air Interface for Fixed Broadband Wireless Access Systems.

[8] S. Yoon, Y. Bar-Ness, "**A Parallel MAP Algorithm for Low Latency Turbo Decoding**", IEEE Communications Letters, Vol. 6, Nº 7, July 2002.

[9] F. Speziali, J. Zory, "**Scalable and Area Efficient Concurrent Interleaver for High Throughput Turbo-Decoders**", IEEE Computer Society.

[10] R. Dobkin, M. Peleg, R. Ginosar, “**Parallel Interleaver Design and VLSI Architecture for Low-Latency MAP Turbo Decoders**”, IEEE Transactions on VLSI Systems, Vol. 13, Nº 4, April 2005.

[11] A.J. Viterbi, “**An Intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes**”, IEEE Journal on Selected Areas in Communications, Vol. 16, Nº 2, February 1998.

[12] G. Masera, G. Piccinini, M.R. Roch, M. Zamboni, “**VLSI Architectures for Turbo Codes**”, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 7, Nº 3, September 1999.

[13] M.J. Thul, N. Wehn, L.P. Rao, “**Enabling High-Speed Turbo-Decoding Through Concurrent Interleaving**”, ISCAS’02, May 2002.

[14] M.J. Thul, F. Gilbert, N. When, “**Optimized Concurrent Interleaving Architecture for High-Throughput Turbo-Decoding**”, ICECS’02, Croatia, September 2002.

[15] “**Closing in on the Perfect Code**”, IEEE Spectrum, March 2004.

[16] http://www.wimaxforum.org/

ANEXO I

FUNCIONES Y SCRIPTS MATLAB®

I. *interleaverFunction.m*

```
%   Returns the interleaved sequence of CTC - IEEE 802.16e-2005
%
% output = interleaverFunction(N,P0,P1,P2,P3)
%
%       INPUTS:         N -> block length [integer]
%                       P0 -> parameter P0 [integer]
%                       P1 -> parameter P1 [integer]
%                       P2 -> parameter P2 [integer]
%                       P3 -> parameter P3 [integer]
%
%       OUTPUTS: output -> interleaved sequence

function output = interleaverFunction(N,P0,P1,P2,P3)

for j=0:N-1
  switch mod(j,4)
  case 0
    output( j+1 ) = mod( (P0*j+1), N);
  case 1
    output( j+1 ) = mod( (P0*j+1+(N/2)+P1), N);
  case 2
    output( j+1 ) = mod( (P0*j+1+P2), N);
  case 3
    output( j+1 ) = mod( (P0*j+1+(N/2)+P3), N);
  end
end
```

II. *optimal_param_Reduced.m*

```
%   Gives the interleaver parameters of CTC - IEEE 802.16e-2005.
%
% [N, P0, P1, P2, P3] = optimal_param_Reduced( x )
%
%    INPUTS:      x -> 1 ... 17 [integer]
%
%    OUTPUTS:     N -> block length [integer]
%                P0 -> parameter P0 [integer]
%                P1 -> parameter P1 [integer]
%                P2 -> parameter P2 [integer]
%                P3 -> parameter P3 [integer]

function [N, P0, P1, P2, P3] = optimal_param_Reduced( x )

if ( x > 17 | x < 1 )
  N = -1; P0 = -1; P1 = -1; P2 = -1; P3 = -1; Pe = -1;
else
  switch x
  case 1
    N = 24; P0 = 5; P1 = 0; P2 = 0; P3 = 0;
  case 2
    N = 36; P0 = 11; P1 = 18; P2 = 0; P3 = 18;
  case 3
    N = 48; P0 = 13; P1 = 24; P2 = 0; P3 = 24;
  case 4
    N = 72; P0 = 11; P1 = 6; P2 = 0; P3 = 6;
  case 5
    N = 96; P0 = 7; P1 = 48; P2 = 24; P3 = 72;
  case 6
    N = 108; P0 = 11; P1 = 54; P2 = 56; P3 = 2;
  case 7
    N = 120; P0 = 13; P1 = 60; P2 = 0; P3 = 60;
  case 8
    N = 144; P0 = 17; P1 = 74; P2 = 72; P3 = 2;
  case 9
    N = 180; P0 = 11; P1 = 90; P2 = 0; P3 = 90;
  case 10
    N = 192; P0 = 11; P1 = 96; P2 = 48; P3 = 144;
  case 11
    N = 216; P0 = 13; P1 = 108; P2 = 0; P3 = 108;
  case 12
    N = 240; P0 = 13; P1 = 120; P2 = 60; P3 = 180;
  case 13
    N = 480; P0 = 53; P1 = 62; P2 = 12; P3 = 2;
  case 14
    N = 960; P0 = 43; P1 = 64; P2 = 300; P3 = 824;
  case 15
    N = 1440; P0 = 43; P1 = 720; P2 = 360; P3 = 540;
  case 16
    N = 1920; P0 = 31; P1 = 8; P2 = 24; P3 = 16;
  case 17
    N = 2400; P0 = 53; P1 = 66; P2 = 24; P3 = 2;
  end
end
```

III. *optimal_param_CTC.m*

```
%   Gives the interleaver parameters of CTC - IEEE 802.16e-2005 by N
%
% [P0, P1, P2, P3] = optimal_param_CTC( N )
%
%    INPUTS:    N -> block length [integer]
%
%    OUTPUTS:  P0 -> parameter P0 [integer]
%              P1 -> parameter P1 [integer]
%              P2 -> parameter P2 [integer]
%              P3 -> parameter P3 [integer]

function [P0, P1, P2, P3] = optimal_param_CTC( N )

switch N
case 24
  P0 = 5; P1 = 0; P2 = 0; P3 = 0;
case 36
  P0 = 11; P1 = 18; P2 = 0; P3 = 18;
case 48
  P0 = 13; P1 = 24; P2 = 0; P3 = 24;
case 72
  P0 = 11; P1 = 6; P2 = 0; P3 = 6;
case 96
  P0 = 7; P1 = 48; P2 = 24; P3 = 72;
case 108
  P0 = 11; P1 = 54; P2 = 56; P3 = 2;
case 120
  P0 = 13; P1 = 60; P2 = 0; P3 = 60;
case 144
  P0 = 17; P1 = 74; P2 = 72; P3 = 2;
case 180
  P0 = 11; P1 = 90; P2 = 0; P3 = 90;
case 192
  P0 = 11; P1 = 96; P2 = 48; P3 = 144;
case 216
  P0 = 13; P1 = 108; P2 = 0; P3 = 108;
case 240
  P0 = 13; P1 = 120; P2 = 60; P3 = 180;
case 480
  P0 = 53; P1 = 62; P2 = 12; P3 = 2;
case 960
  P0 = 43; P1 = 64; P2 = 300; P3 = 824;
case 1440
  P0 = 43; P1 = 720; P2 = 360; P3 = 540;
case 1920
  P0 = 31; P1 = 8; P2 = 24; P3 = 16;
case 2400
  P0 = 53; P1 = 66; P2 = 24; P3 = 2;
otherwise
  P0 = -1; P1 = -1; P2 = -1; P3 = -1;
end
```

IV. *recurrenceInterleaverFuncion.m*

```
%   Returns the interleaved sequence of CTC - IEEE 802.16e-2005
%
% [output,maxLoops] = recurrenceInterleaverFunction(N,P0,P1,P2,P3)
%
%       INPUTS:          N -> block length [integer]
%                        P0 -> parameter P0 [integer]
%                        P1 -> parameter P1 [integer]
%                        P2 -> parameter P2 [integer]
%                        P3 -> parameter P3 [integer]
%
%       OUTPUTS:         output -> interleaved sequence
%                        loops -> total number of loops that have been taken

function [output,loops] = recurrenceInterleaverFunction(N,P0,P1,P2,P3)

cte2 = 1+(N/2)+P1;
cte3 = 1+P2;
cte4 = 1+(N/2)+P3;

loops = zeros(1,N);

for j=0:N-1

  if( j == 0 )
    acumulator = 0;
  else
    acumulator = acumulator + P0;
  end

  switch mod(j,4)
  case 0
    aux = acumulator+1;
  case 1
    aux = acumulator+cte2;
  case 2
    aux = acumulator+cte3;
  case 3
    aux = acumulator+cte4;
  end

  while( aux >= N )
    aux = aux - N;
    loops(j+1) = loops(j+1) + 1;
  end

  output(j+1) = aux;

end
```

V. *recurrenceInterleaverFuncion2.m*

```
%   Returns the interleaved sequence of CTC - IEEE 802.16e-2005
%
% output = recurrenceInterleaverFunction2(N,P0,P1,P2,P3)
%
%       INPUTS:          N -> block length [integer]
%                              P0 -> parameter P0 [integer]
%                              P1 -> parameter P1 [integer]
%                              P2 -> parameter P2 [integer]
%                              P3 -> parameter P3 [integer]
%
%       OUTPUTS:         output -> interleaved sequence

function output = recurrenceInterleaverFunction2(N,P0,P1,P2,P3)

cte2 = mod(1+(N/2)+P1,N);
cte3 = mod(1+P2,N);
cte4 = mod(1+(N/2)+P3,N);

loops = zeros(1,N);

for j=0:N-1

  if( j == 0 )
    acumulator = 0;
  else
    acumulator = acumulator + P0;
  end

  if( acumulator >= N )
    acumulator = acumulator - N;
  end

  switch mod(j,4)
  case 0
    aux = acumulator+1;
  case 1
    aux = acumulator+cte2;
  case 2
    aux = acumulator+cte3;
  case 3
    aux = acumulator+cte4;
  end

  if( aux >= N )
    aux = aux - N;
  end

  while( aux >= N )
    aux = aux - N;
    loops(j+1) = loops(j+1) + 1;
  end

  output(j+1) = aux;

end
```

VI. *fillBlank*

```
%       output = fillBlank(long, str)
%
%   Fills with blank spaces the string 'str'.
%   The output string will have 'long' length.

function output = fillBlank(long, str)


for k = 1:fix((long-length(str))/2)
        str = [' ' str];
end

for k = length(str)+1:long
        str = [str ' '];
end

output = str;
```

VII. *matrixInvest.m*

```
%   Invest the order of the input matrix/array
%
% output = matrixInvest(input)
%
%   INPUTS:   input -> matrix or array to invest its order
%
%   OUTPUTS:  output -> matrix or array with its order invested

function output = matrixInvest(input)

long = length(input(:,1));
width = length(input(1,:));
output = zeros(long,width);

if (long == 1 | width == 1)
  for i=1:length(input)
    output(length(input)-(i-1)) = input(i);
  end
else
  for i=1:long
    output(long-(i-1),:) = input(i,:);
  end
end
```

VIII. *collisions.m*

```
clear all;
close all;
clc;

%Creation of the output file
[fidOutput, messageOutput] = fopen('collision_report.txt', 'a+');
fclose(fidOutput);

%Values of Pe:
Pe_array = [2 4 6 8];
%Concurrent accesses probabilities:
prob = [];

for choice_Pe = 1:4

  Pe = Pe_array( choice_Pe );

  %Concurrent accesses:
  conACSS = zeros(1,9);

  for choice_N = 1:17

    %Interleaver parameters:
    [N, P0, P1, P2, P3] = optimal_param_Reduced( choice_N );

    %Verification if 'N' is divisible by 'Pe'
    if( fix(N/Pe) == N/Pe )

      %Sublock lenght:
      n = N/Pe;

      %Interleaver:
      posPj = interleaverFunction(N,P0,P1,P2,P3);

      %----------------------------------------------------
      %Pipeline:
      index = [];
      memRef = [];
      for i = 1:Pe
        increase = (i-1)*n;

        %Index & references to memory doing by interleaver function
        indexTMP = [];
        mRefTMP = [];
        for j = 1:n
          indexTMP = [indexTMP; posPj(j+increase)];
          mRefTMP = [mRefTMP; fix(posPj(j+increase)/n)];
        end

        index = [index indexTMP];
        memRef = [memRef mRefTMP];
      end
      index = mod(index,n);
      %----------------------------------------------------

      %----------------------------------------------------
      %Verification if there's some collison
      FLAG_collision = 0;
      memRef_s = '';
      probability = 0;
      for i = 1:n
        takeUpMEM = zeros(1,Pe);
```

```
    coll_TMP = 0;
    for j = 1:Pe
      if( takeUpMEM( memRef(i,j)+1 ) == 0 )
        takeUpMEM( memRef(i,j)+1 ) = 1;
      else
        %COLLISION!!!
        FLAG_collision = 1;
        coll_TMP = 1;
        takeUpMEM( memRef(i,j)+1 ) = takeUpMEM( memRef(i,j)+1 ) + 1;
      end
    end

    %Adding the last 'C' if there's a collision
    if( coll_TMP == 0 )
      memRef_s = strvcat(memRef_s, num2str(memRef(i,:)));
    else
      memRef_s = strvcat(memRef_s, [num2str(memRef(i,:)) ' C']);
      probability = probability + 1;
    end

    %Concurrent accesses calculation:
    for j = 1:Pe
      switch takeUpMEM(j)
      case 0
        %No access at mem 'j'
        conACSS(1) = conACSS(1) + 1;
      case 1
        %1 access at mem 'j'
        conACSS(2) = conACSS(2) + 1;
      case 2
        %2 accesses at mem 'j'
        conACSS(3) = conACSS(3) + 1;
      case 3
        %3 accesses at mem 'j'
        conACSS(4) = conACSS(4) + 1;
      case 4
        %4 accesses at mem 'j'
        conACSS(5) = conACSS(5) + 1;
      case 5
        %5 accesses at mem 'j'
        conACSS(6) = conACSS(6) + 1;
      case 6
        %6 accesses at mem 'j'
        conACSS(7) = conACSS(7) + 1;
      case 7
        %7 accesses at mem 'j'
        conACSS(8) = conACSS(8) + 1;
      case 8
        %8 accesses at mem 'j'
        conACSS(9) = conACSS(9) + 1;
      end
    end
  end
  %----------------------------------------------------

  %Collision probability:
  probability = 100*(probability/n);

  %Reporting a collision:
  if( FLAG_collision == 1 )
    s = [' ************' '\n'];
    s = [s 'N = ' num2str(N) '        Pe = ' num2str(Pe) '\n'];
    %Writing to file
    [fidOutput, messageOutput] = fopen('collision_report.txt', 'a+');
    fprintf(fidOutput, s, 'char');
```

```
        fclose(fidOutput);

        for i = 1:n
          %Writing to file
          s = [num2str(memRef_s(i,:)) '\n'];
          [fidOutput, messageOutput] = fopen('collision_report.txt', 'a+');
          fprintf(fidOutput, s, 'char');
          fclose(fidOutput);
        end

        s = [ '\n' 'Collision probability -> ' num2str(probability) '%%' '\n' '\n'];
        %Writing to file
        [fidOutput, messageOutput] = fopen('collision_report.txt', 'a+');
        fprintf(fidOutput, s, 'char');
        fclose(fidOutput);
      end

    end %Verification if 'N' is divisible by 'Pe'
  end %choice_N

  %CONCURRENT ACCESSES PROBABILITIES:
  s = strvcat(['Pe = ' num2str(Pe)], '----------', num2str((conACSS/sum(conACSS))'));
  prob = [prob, s];

end %choice_Pe

%Making the concurrent accesses report:
s = strvcat('Concurrent    ','accesses','0','1','2','3','4','5','6','7','8');
prob = [s, prob];
s = [];
for i = 1:length(prob(:,1))
  str = prob(i,:);
  s = [s str '\n'];
end
%Writing to file
[fidOutput, messageOutput] = fopen('collision_report.txt', 'a+');
fprintf(fidOutput, ['\n' s '\n'], 'char');
fclose(fidOutput);
```

IX. *bufferReading.m*

```
clear all;
close all;
clc;

%Creation of the output file
[fidOutput, messageOutput] = fopen('report_bufferReading.txt', 'a+');
fclose(fidOutput);

%Reporting
s = ['\n' '\n'];
s = [s '         TIBB ARCHITECTURE - READ' '\n' '\n'];
s = [s '  All operations needs ONE position in BUFFER_en and BUFFER_data,' '\n'];
s = [s ' also needs  1+(N/Pe)  cycles to complete the READING,' '\n'];
s = [s ' except:' '\n' '\n'];
%Writing to file
[fidOutput, messageOutput] = fopen('report_bufferReading.txt', 'a+');
fprintf(fidOutput, s, 'char');
fclose(fidOutput);

%Worst case:
wc_size_en = 0;
wc_size_data = 0;

%Values of Pe:
Pe_array = [2 4 6 8];

for choice_Pe = 1:4
  Pe = Pe_array( choice_Pe );

  for choice_N = 1:17

    %Interleaver parameters:
    [N, P0, P1, P2, P3] = optimal_param_Reduced( choice_N );

    %Verification if 'N' is divisible by 'Pe'
    if( fix(N/Pe) == N/Pe )

      %Sublock lenght:
      n = N/Pe;

      %Interleaver:
      posPj = interleaverFunction(N,P0,P1,P2,P3);

      %----------------------------------------------------
      %Pipeline:
      index = [];
      memRef = [];
      for i = 1:Pe
        increase = (i-1)*n;

        %Index & references to memory doing by interleaver function
        indexTMP = [];
        mRefTMP = [];
        for j = 1:n
          indexTMP = [indexTMP; posPj(j+increase)];
          mRefTMP = [mRefTMP; fix(posPj(j+increase)/n)];
        end

        index = [index indexTMP];
        memRef = [memRef mRefTMP];
      end
      index = mod(index,n);
```

```
%----------------------------------------------------

%----------------------------------------------------
%  BUFFER TO READING OPERATION
BUFFER_en = zeros(100,Pe);
POINTER_en = zeros(1,Pe);
BUFFER_data = zeros(100,Pe);
POINTER_data = zeros(1,Pe);

%DATA TO COMPLETE:
indexData = 1;
data = memRef(indexData,:);

%Maximum buffers size
maxSize_en = 0;
maxSize_data = 0;

for clk = 1:10*n

  %   1) Read from memory to BUFFER_data
  for i = 1:Pe
    if( BUFFER_en(1,i) == 1 )
      %Reading, to BUFFER_data
      POINTER_data(i) = POINTER_data(i) + 1;
      BUFFER_data( POINTER_data(i), i ) = 1;
      BUFFER_en(1:end,i) = [BUFFER_en(2:end,i); 0];
      POINTER_en(i) = POINTER_en(i) - 1;
      %Verify maximum BUFFER_data size
      maxSize_data = max(maxSize_data, POINTER_data(i));
    end
  end

  %   2) Verify if data in window is ready
  for i = 1:Pe
    if( BUFFER_data(1,i) == 1 )
      %Data ready to be readed in "mem i-1"
      %This data belongs to the current window?
      for j = 1:Pe
        if( data(j) == i-1 )
          %Data at "mem i-1" belongs to the current window
          %To BUFFER_data to WINDOWS
          data(j) = -1;
          BUFFER_data(1:end,i) = [BUFFER_data(2:end,i); 0];
          POINTER_data(i) = POINTER_data(i) - 1;
          break
        end
      end
    end
  end

  %   3) Verify if window is completed
  if( data == -ones(1,Pe) )
    %Data has been completed correctly
    %Next data
    indexData = indexData + 1;
    %If "indexData" > N/Pe --> COMPLETED WORK
    if( indexData > n )
      break;        %Out of "for clk = 1:10*n"
    else
      data = memRef(indexData,:);
    end
  end

  %   4) Input data to BUFFER_en
  if( clk <= n )
```

```
          for i = 1:Pe
            tmp = memRef(clk,i)+1;
            POINTER_en(tmp) = POINTER_en(tmp) + 1;
            BUFFER_en( POINTER_en(tmp), tmp ) = 1;
            %Verify maximum BUFFER_en size
            maxSize_en = max(maxSize_en, POINTER_en(tmp));
          end
        end
      end
      %----------------------------------------------------

      %Worst case buffers size:
      wc_size_en = max(wc_size_en, maxSize_en);
      wc_size_data = max(wc_size_data, maxSize_data);

      %Report
      s = ['   N = ' num2str(N) '        Pe = ' num2str(Pe) '\n'];
      s = [s '     Maximum BUFFER_en size: ' num2str(maxSize_en) '\n'];
      s = [s '     Maximum BUFFER_data size: ' num2str(maxSize_data) '\n'];
      s = [s '     Cycles to complete the reading: ' num2str(clk) '\n'];
      s = [s '\n'];
      if( maxSize_en > 1 | maxSize_data > 1 )
        %Writing to file
        [fidOutput, messageOutput] = fopen('report_bufferReading.txt', 'a+');
        fprintf(fidOutput, s, 'char');
        fclose(fidOutput);
      end

    end
  end
end

%Reporting the worst case:
s = ['\n' '\n'];
s = [s '  So, the worst buffers sizes will be:' '\n' '\n'];
s = [s '     Maximum BUFFER_en size: ' num2str(wc_size_en) '\n'];
s = [s '     Maximum BUFFER_data size: ' num2str(wc_size_data) '\n'];
%Writing to file
[fidOutput, messageOutput] = fopen('report_bufferReading.txt', 'a+');
fprintf(fidOutput, s, 'char');
fclose(fidOutput);
```

X. *bufferWriting.m*

```
clear all;
close all;
clc;

%Creation of the output file
[fidOutput, messageOutput] = fopen('report_bufferWriting.txt', 'a+');
fclose(fidOutput);

%Reporting
s = ['\n' '\n'];
s = [s '          TIBB ARCHITECTURE - WRITING' '\n' '\n'];
s = [s '  All operations needs ONE position in BUFFER,' '\n'];
s = [s ' also needs  1+(N/Pe)  cycles to complete the READING,' '\n'];
s = [s ' except:' '\n' '\n'];
%Writing to file
[fidOutput, messageOutput] = fopen('report_bufferWriting.txt', 'a+');
fprintf(fidOutput, s, 'char');
fclose(fidOutput);

%Worst case:
wc_size = 0;
wc_clk = 0;

%Values of Pe:
Pe_array = [2 4 6 8];

for choice_Pe = 1:4
  Pe = Pe_array( choice_Pe );

  for choice_N = 1:17

    %Interleaver parameters:
    [N, P0, P1, P2, P3] = optimal_param_Reduced( choice_N );

    %Verification if 'N' is divisible by 'Pe'
    if( fix(N/Pe) == N/Pe )

      %Sublock lenght:
      n = N/Pe;

      %Interleaver:
      posPj = interleaverFunction(N,P0,P1,P2,P3);

      %----------------------------------------------------
      %Pipeline:
      index = [];
      memRef = [];
      for i = 1:Pe
        increase = (i-1)*n;

        %Index & references to memory doing by interleaver function
        indexTMP = [];
        mRefTMP = [];
        for j = 1:n
          indexTMP = [indexTMP; posPj(j+increase)];
          mRefTMP = [mRefTMP; fix(posPj(j+increase)/n)];
        end

        index = [index indexTMP];
        memRef = [memRef mRefTMP];
      end
      index = mod(index,n);
```

```
      %----------------------------------------------------

      %----------------------------------------------------
      %  BUFFER TO WRITING OPERATION
      BUFFER = zeros(10,Pe);
      POINTER = zeros(1,Pe);

      %Maximum buffer size
      maxSize = 0;

      for clk = 1:10*n

        %   1) BUFFER to memory
        for i = 1:Pe
          if( POINTER(i) > 0 )
            BUFFER(1:end,i) = [BUFFER(2:end,i); 0];
            POINTER(i) = POINTER(i) - 1;
          end
        end

        %   2) Input data to BUFFER
        if( clk <= n )
          for i = 1:Pe
            tmp = memRef(clk,i)+1;
            POINTER(tmp) = POINTER(tmp) + 1;
            BUFFER( POINTER(tmp), tmp ) = 1;
            %Verify maximum BUFFER size
            maxSize = max(maxSize, POINTER(tmp));
          end
        else
          if( POINTER == zeros(1,Pe) )
            break;
          end
        end

      end
      %----------------------------------------------------

      %Cycles extra:
      clkEx = clk - (n+1);

      %Worst case buffer size:
      wc_size = max(wc_size, maxSize);

      %Worst case cycles extra:
      wc_clk = max(wc_clk, clkEx);

      %Report
      s = ['   N = ' num2str(N) '        Pe = ' num2str(Pe) '\n'];
      s = [s '     Maximum BUFFER size: ' num2str(maxSize) '\n'];
      s = [s '     Cycles to complete the reading: ' num2str(clk)];
      s = [s '  [' num2str(clkEx) '] extra cycles' '\n'];
      s = [s '\n'];
      if( maxSize > 1 )
        %Writing to file
        [fidOutput, messageOutput] = fopen('report_bufferWriting.txt', 'a+');
        fprintf(fidOutput, s, 'char');
        fclose(fidOutput);
      end

    end
  end
end

%Reporting the worst cases:
```

```
s = ['\n' '\n'];
s = [s '  So, the worst buffer size & cycles extra will be:' '\n' '\n'];
s = [s '     Maximum BUFFER size: ' num2str(wc_size) '\n'];
s = [s '     ' num2str(wc_clk) ' extra cycles' '\n'];
%Writing to file
[fidOutput, messageOutput] = fopen('report_bufferWriting.txt', 'a+');
fprintf(fidOutput, s, 'char');
fclose(fidOutput);
```

XI. *totalStalling.m*

```
clear all;
close all;
clc;

%Creation of the output file
[fidOutput, messageOutput] = fopen('report_totalStalling.txt', 'a+');
fclose(fidOutput);

%Reporting
s = ['\n' '\n'];
s = [s '         TOTAL STALLING MECHANISM - READ' '\n' '\n'];
s = [s '  All operations needs ONE position in BUFFER_en and BUFFER_data,' '\n'];
s = [s ' also needs  1+(N/Pe)  cycles to complete the READING,' '\n'];
s = [s ' except:' '\n' '\n'];
%Writing to file
[fidOutput, messageOutput] = fopen('report_totalStalling.txt', 'a+');
fprintf(fidOutput, s, 'char');
fclose(fidOutput);

%Worst case:
wc_size_en = 0;
wc_size_data = 0;

%Values of Pe:
Pe_array = [2 4 6 8];

for choice_Pe = 1:4
  Pe = Pe_array( choice_Pe );

  for choice_N = 1:17

    %Interleaver parameters:
    [N, P0, P1, P2, P3] = optimal_param_Reduced( choice_N );

    %Verification if 'N' is divisible by 'Pe'
    if( fix(N/Pe) == N/Pe )

      %Sublock lenght:
      n = N/Pe;

      %Interleaver:
      posPj = interleaverFunction(N,P0,P1,P2,P3);

      %----------------------------------------------------
      %Pipeline:
      index = [];
      memRef = [];
      for i = 1:Pe
        increase = (i-1)*n;

        %Index & references to memory doing by interleaver function
        indexTMP = [];
        mRefTMP = [];
        for j = 1:n
          indexTMP = [indexTMP; posPj(j+increase)];
          mRefTMP = [mRefTMP; fix(posPj(j+increase)/n)];
        end

        index = [index indexTMP];
        memRef = [memRef mRefTMP];
      end
      index = mod(index,n);
```

```
%----------------------------------------------------

%----------------------------------------------------
%  TOTAL STALLING WHEN A COLLISION APPEARS
%    N = 108  &  Pe = 2
%    N = 108  &  Pe = 4
%    N = 108  &  Pe = 6
%    N = 72  &  Pe = 8
BUFFER_en = zeros(100,Pe);
POINTER_en = zeros(1,Pe);
BUFFER_data = zeros(100,Pe);
POINTER_data = zeros(1,Pe);

%DATA TO COMPLETE:
indexData = 1;
data = memRef(indexData,:);

%Maximum buffers size
maxSize_en = 0;
maxSize_data = 0;

%Stalling variable:
stalling = 0;

%Auxiliar counter
counter = 1;

for clk = 1:10*n

  %   1) Read from memory to BUFFER_data
  for i = 1:Pe
    if( BUFFER_en(1,i) == 1 )
      %Reading, to BUFFER_data
      POINTER_data(i) = POINTER_data(i) + 1;
      BUFFER_data( POINTER_data(i), i ) = 1;
      BUFFER_en(1:end,i) = [BUFFER_en(2:end,i); 0];
      POINTER_en(i) = POINTER_en(i) - 1;
      %Verify maximum BUFFER_data size
      maxSize_data = max(maxSize_data, POINTER_data(i));
    end
  end

  %   2) Verify if data in window is ready
  for i = 1:Pe
    if( BUFFER_data(1,i) == 1 )
      %Data ready to be readed in "mem i-1"
      %This data belongs to the current window?
      for j = 1:Pe
        if( data(j) == i-1 )
          %Data at "mem i-1" belongs to the current window
          %To BUFFER_data to WINDOWS
          data(j) = -1;
          BUFFER_data(1:end,i) = [BUFFER_data(2:end,i); 0];
          POINTER_data(i) = POINTER_data(i) - 1;
          break
        end
      end
    end
  end

  %   3) Verify if window is completed
  if( data == -ones(1,Pe) )
    %Data has been completed correctly
    %Next data
    indexData = indexData + 1;
```

```
          %If "indexData" > N/Pe --> COMPLETED WORK
          if( indexData > n )
            break;        %Out of "for clk = 1:10*n"
          else
            data = memRef(indexData,:);
          end
        end

        %   4) Input data to BUFFER_en
        if( stalling == 0 )
          if( counter <= n )
            for i = 1:Pe
              tmp = memRef(counter,i)+1;
              POINTER_en(tmp) = POINTER_en(tmp) + 1;
              BUFFER_en( POINTER_en(tmp), tmp ) = 1;
              %Verify maximum BUFFER_en size
              maxSize_en = max(maxSize_en, POINTER_en(tmp));
            end
            counter = counter + 1;
          end

          if((N == 108 & (Pe == 2 | Pe == 4 | Pe == 6)) | (N == 72 & Pe == 8))
            stalling = 1;
          end
        else
          stalling = 0;
        end
      end
      %----------------------------------------------------

      %Worst case buffers size:
      wc_size_en = max(wc_size_en, maxSize_en);
      wc_size_data = max(wc_size_data, maxSize_data);

      %Report
      s = ['   N = ' num2str(N) '        Pe = ' num2str(Pe) '\n'];
      s = [s '     Maximum BUFFER_en size: ' num2str(maxSize_en) '\n'];
      s = [s '     Maximum BUFFER_data size: ' num2str(maxSize_data) '\n'];
      s = [s '     Cycles to complete the reading: ' num2str(clk) '\n'];
      s = [s '\n'];
      if( maxSize_en > 1 | maxSize_data > 1 )
        %Writing to file
        [fidOutput, messageOutput] = fopen('report_totalStalling.txt', 'a+');
        fprintf(fidOutput, s, 'char');
        fclose(fidOutput);
      end

    end
  end
end

%Reporting the worst case:
s = ['\n' '\n'];
s = [s '  So, the worst buffers sizes will be:' '\n' '\n'];
s = [s '     Maximum BUFFER_en size: ' num2str(wc_size_en) '\n'];
s = [s '     Maximum BUFFER_data size: ' num2str(wc_size_data) '\n'];
%Writing to file
[fidOutput, messageOutput] = fopen('report_totalStalling.txt', 'a+');
fprintf(fidOutput, s, 'char');
fclose(fidOutput);
```

XII. *recurrence.m*

```
clear all;
close all;
clc;

%Creation of the output file
[fidOutput, messageOutput] = fopen('recurrenceReport.txt', 'a+');
fclose(fidOutput);

%Values of Pe:
Pe_array = [2 4 6 8];

for choice_N = 1:17

  %Interleaver parameters:
  [N, P0, P1, P2, P3] = optimal_param_Reduced( choice_N );

  %Interleaver
  index_A = interleaverFunction(N,P0,P1,P2,P3);

  %Interleaver recurrence:
  [index_B,loops] = recurrenceInterleaverFunction(N,P0,P1,P2,P3);

  %Reporting & verifying:
  s = [' ************' '\n'];
  s = [s 'N = ' num2str(N) '\n'];
  if( sum(abs(index_A-index_B)) == 0 )
    s = [s '   OK!! Recurrence Interleaver correct!!!!' '\n'];
  else
    s = [s '   ---> ERROR!! Recurrence Interleaver incorrect!!!!' '\n'];
  end
  s = [s '   The recurrence interleaver has take ' num2str(sum(loops)) ];
  s = [s ' loops!!!!!' '\n'];
  %Writing to file
  [fidOutput, messageOutput] = fopen('recurrenceReport.txt', 'a+');
  fprintf(fidOutput, s, 'char');
  fclose(fidOutput);

end
```

XIII. *recurrence2.m*

```
clear all;
close all;
clc;

%Creation of the output file
[fidOutput, messageOutput] = fopen('recurrence2Report.txt', 'a+');
fclose(fidOutput);

%Values of Pe:
Pe_array = [2 4 6 8];

for choice_N = 1:17

  %Interleaver parameters:
  [N, P0, P1, P2, P3] = optimal_param_Reduced( choice_N );

  %Interleaver
  index_A = interleaverFunction(N,P0,P1,P2,P3);

  %Interleaver recurrence:
  [index_B,loops] = recurrenceInterleaverFunction2(N,P0,P1,P2,P3);

  %Reporting & verifying:
  s = [' ************' '\n'];
  s = [s 'N = ' num2str(N) '\n'];
  if( sum(abs(index_A-index_B)) == 0 )
    s = [s '   OK!! Recurrence Interleaver correct!!!!' '\n'];
  else
    s = [s '   ---> ERROR!! Recurrence Interleaver incorrect!!!!' '\n'];
  end
  s = [s '   The recurrence interleaver has take ' num2str(sum(loops)) ];
  s = [s ' loops!!!!!' '\n'];
  %Writing to file
  [fidOutput, messageOutput] = fopen('recurrence2Report.txt', 'a+');
  fprintf(fidOutput, s, 'char');
  fclose(fidOutput);

end
```

XIV. *parametersToLoad.m*

```
clear all;
close all;
clc;

%Creation of the output file
[fid, message] = fopen('ParametersToLoad.txt', 'a+');
fclose(fid);

s = ['------------------------------------------------'];
s = [ s '---------------------------------------------' '\n'];
s = [s '|      |     |     |    |      |      |      |'];
s = [ s '                   sumBegin                    |' '\n'];
s = [s '|  Pe  |  N  |  n  | P0 | cte2 | cte3 | cte4 |'];
s = [ s '  0  |  1  |  2  |  3  |  4  |  5  |  6  |  7  |' '\n'];
s = [s '|      |     |     |    |      |      |      |'];
s = [ s '     |     |     |     |     |     |     |     |' '\n'];
s = [s '----------------------------------------------'];
s = [ s '------------------------------------------------' '\n'];

%Writing to file
[fidOutput, messageOutput] = fopen('ParametersToLoad.txt', 'a+');
fprintf(fidOutput, s, 'char');
fclose(fidOutput);

%Values for Pe:
PeArray = [1,2,4,6,8];

%Spaces in black for each field
longPe = 6;
longN = 5;
longP0 = 4;
longCte = 6;
longSum = 5;

%Maximums:
max_N = 0;
max_Pe = 0;
max_nn = 0;
max_P0 = 0;
max_cte2 = 0;
max_cte3 = 0;
max_cte4 = 0;
max_sB = 0;

%Loops for all values (N & Pe):
for j = 1:5
  for i = 1:17

    [N,P0,P1,P2,P3] = optimal_param_Reduced(i);
    Pe = PeArray(j);

    %Calculation of parameters:
    n = N/Pe;
    cte2 = mod(1+(N/2)+P1,N);
    cte3 = mod(1+P2,N);
    cte4 = mod(1+(N/2)+P3,N);

    %"sumBegins"
    sB = zeros(1,Pe);
    for k = 1:Pe
      sB(k) = mod(P0*(k-1)*n,N);
    end
```

```
for k = 1:8
  if( k <= Pe )
    eval(['sB' num2str(k-1) '=' num2str(sB(k))]);
  else
    eval(['sB' num2str(k-1) '=[]']);
  end
end

%Geting the maximums:
max_N = max([N max_N]);
max_Pe = max([Pe max_Pe]);
max_nn = max([n max_nn]);
max_P0 = max([P0 max_P0]);
max_cte2 = max([cte2 max_cte2]);
max_cte3 = max([cte3 max_cte3]);
max_cte4 = max([cte4 max_cte4]);
max_sB = max([sB max_sB]);

%Numbers to string
%If n is not dividable between N --> blank
N = num2str( N );
Pe = num2str( Pe );
if( n ~= fix(n) )
  n = '';
  P0 = '';
  cte2 = ''; cte3 = ''; cte4 = '';
  sB0 = ''; sB1 = ''; sB2 = ''; sB3 = ''; sB4 = ''; sB5 = ''; sB6 = ''; sB7 = '';
else
  n = num2str( n );
  P0 = num2str( P0 );
  cte2 = num2str( cte2 );
  cte3 = num2str( cte3 );
  cte4 = num2str( cte4 );
  sB0 = num2str( sB0 );
  sB1 = num2str( sB1 );
  sB2 = num2str( sB2 );
  sB3 = num2str( sB3 );
  sB4 = num2str( sB4 );
  sB5 = num2str( sB5 );
  sB6 = num2str( sB6 );
  sB7 = num2str( sB7 );
end

%Fill with spaces in black the strings
Pe = fillBlank( longPe, Pe );
N = fillBlank( longN, N );
n = fillBlank( longN, n );
P0 = fillBlank( longP0, P0 );
cte2 = fillBlank( longCte, cte2 );
cte3 = fillBlank( longCte, cte3 );
cte4 = fillBlank( longCte, cte4 );
sB0 = fillBlank( longSum, sB0 );
sB1 = fillBlank( longSum, sB1 );
sB2 = fillBlank( longSum, sB2 );
sB3 = fillBlank( longSum, sB3 );
sB4 = fillBlank( longSum, sB4 );
sB5 = fillBlank( longSum, sB5 );
sB6 = fillBlank( longSum, sB6 );
sB7 = fillBlank( longSum, sB7 );

%Creation of the string to write
s = ['|' Pe '|' N '|' n '|' P0 '|' cte2 '|' cte3 '|' cte4 '|'];
s = [s sB0 '|' sB1 '|' sB2 '|' sB3 '|' sB4 '|' sB5 '|' sB6 '|' sB7 '|'];
s = [s '\n'];
```

```
    s = [s '--------     -------    ----------------------' '\n'];

    %WRITE TO FILE
    [fidOutput, messageOutput] = fopen('ParametersToLoad.txt', 'a+');
    fprintf(fidOutput, s, 'char');
    fclose(fidOutput);

  end
end

%Binary codification length for each parameter:
lng_N = length(dec2bin(max_N));
lng_Pe = length(dec2bin(max_Pe));
lng_nn = length(dec2bin(max_nn));
lng_P0 = length(dec2bin(max_P0));
lng_cte2 = length(dec2bin(max_cte2));
lng_cte3 = length(dec2bin(max_cte3));
lng_cte4 = length(dec2bin(max_cte4));
lng_sB = length(dec2bin(max_sB));

s = ['\n' '\n' '  The optimal length for each parameter is:' '\n'];
s = [s '   -> N         by ' num2str(lng_N) ' bits' '\n'];
s = [s '   -> Pe        by ' num2str(lng_Pe) ' bits' '\n'];
s = [s '   -> n         by ' num2str(lng_nn) ' bits' '\n'];
s = [s '   -> P0        by ' num2str(lng_P0) ' bits' '\n'];
s = [s '   -> cte2      by ' num2str(lng_cte2) ' bits' '\n'];
s = [s '   -> cte3      by ' num2str(lng_cte3) ' bits' '\n'];
s = [s '   -> cte4      by ' num2str(lng_cte4) ' bits' '\n'];
s = [s '   -> sumBegin  by ' num2str(lng_sB) ' bits' '\n'];

%WRITE TO FILE
[fidOutput, messageOutput] = fopen('ParametersToLoad.txt', 'a+');
fprintf(fidOutput, s, 'char');
fclose(fidOutput);


%MAKING "valuesN.vhd"
s= [];
s = [ s 'library ieee;' '\n' ];
s = [ s 'use ieee.std_logic_1164.all;' '\n' ];
s = [ s 'use ieee.std_logic_arith.all;' '\n' ];
s = [ s 'use ieee.std_logic_unsigned.all;' '\n' ];
s = [ s '\n' ];
s = [ s 'entity valuesN is' '\n' ];
s = [ s '\n' ];
s = [ s '  port (' '\n' ];
s = [ s '    N_in      : in  std_logic_vector(' num2str(lng_N-1) ' downto 0);' '\n' ];
s = [ s '    Pe_in     : in  std_logic_vector(' num2str(lng_Pe-1) ' downto 0);' '\n' ];
s = [ s '    n_out     : out std_logic_vector(' num2str(lng_nn-1) ' downto 0);' '\n' ];
s = [ s '    P0_out    : out std_logic_vector(' num2str(lng_P0-1) ' downto 0);' '\n' ];
s = [ s '    cte2_out  : out std_logic_vector(' num2str(lng_cte2-1) ' downto 0);' '\n' ];
s = [ s '    cte3_out  : out std_logic_vector(' num2str(lng_cte3-1) ' downto 0);' '\n' ];
s = [ s '    cte4_out  : out std_logic_vector(' num2str(lng_cte4-1) ' downto 0);' '\n' ];
s = [ s '    sumBegin1 : out std_logic_vector(' num2str(lng_sB-1) ' downto 0);' '\n' ];
s = [ s '    sumBegin2 : out std_logic_vector(' num2str(lng_sB-1) ' downto 0);' '\n' ];
s = [ s '    sumBegin3 : out std_logic_vector(' num2str(lng_sB-1) ' downto 0);' '\n' ];
s = [ s '    sumBegin4 : out std_logic_vector(' num2str(lng_sB-1) ' downto 0);' '\n' ];
s = [ s '    sumBegin5 : out std_logic_vector(' num2str(lng_sB-1) ' downto 0);' '\n' ];
s = [ s '    sumBegin6 : out std_logic_vector(' num2str(lng_sB-1) ' downto 0);' '\n' ];
s = [ s '    sumBegin7 : out std_logic_vector(' num2str(lng_sB-1) ' downto 0));' '\n' ];
s = [ s '\n' ];
s = [ s 'end valuesN;' '\n' ];
s = [ s '\n' ];
s = [ s 'architecture behav_valuesN of valuesN is' '\n' ];
s = [ s '\n' ];
```

```
s = [ s 'begin  -- behav_valuesN' '\n' ];
s = [ s '\n' ];
s = [ s '  selection: process (N_in, Pe_in)' '\n' ];
s = [ s '  begin  -- process selection' '\n' ];
s = [ s '\n' ];
s = [ s '    case Pe_in is' '\n' ];

[fid, message] = fopen('valuesN.vhd', 'a+');
fprintf(fidOutput, s, 'char');
fclose(fidOutput);
s = [];

for j = 1:5
  Pe = PeArray(j);
  s = [ '      -- --------------------------------'];
  s = [ s '--------------------------------------' '\n' ];
  s = [ s '      when "' dec2bin(Pe,lng_Pe) '" =>          '];
  s = [ s '                 -- Pe = ' num2str(Pe) '\n' ];
  s = [ s '\n' ];

  for i = 1:17

    [N,P0,P1,P2,P3] = optimal_param_Reduced(i);
    n = N/Pe;
    %Only the data that is valid:
    if( n == fix(n) )

      Nstr = [ '"' num2str( dec2bin(N,lng_N) ) '"' ];
      P0str = [ '"' num2str( dec2bin(P0,lng_P0) ) '"' ];
      if( i == 1 )
        s = [ s '        if (N_in = ' Nstr ') then        -- N = ' num2str(N) '\n' ];
      else
        s = [ s '        elsif (N_in = ' Nstr ') then     -- N = ' num2str(N) '\n' ];
      end

      %Calc of the params (n, cte2, cte3, cte4 & sumBegins)
      nstr = [ '"' num2str( dec2bin(n,lng_nn) ) '"' ];
      cte2 = [ '"' num2str( dec2bin(mod(1+(N/2)+P1,N),lng_cte2) ) '"' ];
      cte3 = [ '"' num2str( dec2bin(mod(1+P2,N),lng_cte3) ) '"' ];
      cte4 = [ '"' num2str( dec2bin(mod(1+(N/2)+P3,N),lng_cte4) ) '"' ];

      switch Pe
      case 1
        sB1 = '(others => \''0\'')';
        sB2 = '(others => \''0\'')';
        sB3 = '(others => \''0\'')';
        sB4 = '(others => \''0\'')';
        sB5 = '(others => \''0\'')';
        sB6 = '(others => \''0\'')';
        sB7 = '(others => \''0\'')';
      case 2
        sB1 = [ '"' num2str( dec2bin(mod(P0*n,N),lng_sB) ) '"' ];
        sB2 = '(others => \''0\'')';
        sB3 = '(others => \''0\'')';
        sB4 = '(others => \''0\'')';
        sB5 = '(others => \''0\'')';
        sB6 = '(others => \''0\'')';
        sB7 = '(others => \''0\'')';
      case 4
        sB1 = [ '"' num2str( dec2bin(mod(P0*n,N),lng_sB) ) '"' ];
        sB2 = [ '"' num2str( dec2bin(mod(P0*2*n,N),lng_sB) ) '"' ];
        sB3 = [ '"' num2str( dec2bin(mod(P0*3*n,N),lng_sB) ) '"' ];
        sB4 = '(others => \''0\'')';
        sB5 = '(others => \''0\'')';
        sB6 = '(others => \''0\'')';
```

```
        sB7 = '(others => \''0\'')';
      case 6
        sB1 = [ '"' num2str( dec2bin(mod(P0*n,N),lng_sB) ) '"' ];
        sB2 = [ '"' num2str( dec2bin(mod(P0*2*n,N),lng_sB) ) '"' ];
        sB3 = [ '"' num2str( dec2bin(mod(P0*3*n,N),lng_sB) ) '"' ];
        sB4 = [ '"' num2str( dec2bin(mod(P0*4*n,N),lng_sB) ) '"' ];
        sB5 = [ '"' num2str( dec2bin(mod(P0*5*n,N),lng_sB) ) '"' ];
        sB6 = '(others => \''0\'')';
        sB7 = '(others => \''0\'')';
      case 8
        sB1 = [ '"' num2str( dec2bin(mod(P0*n,N),lng_sB) ) '"' ];
        sB2 = [ '"' num2str( dec2bin(mod(P0*2*n,N),lng_sB) ) '"' ];
        sB3 = [ '"' num2str( dec2bin(mod(P0*3*n,N),lng_sB) ) '"' ];
        sB4 = [ '"' num2str( dec2bin(mod(P0*4*n,N),lng_sB) ) '"' ];
        sB5 = [ '"' num2str( dec2bin(mod(P0*5*n,N),lng_sB) ) '"' ];
        sB6 = [ '"' num2str( dec2bin(mod(P0*6*n,N),lng_sB) ) '"' ];
        sB7 = [ '"' num2str( dec2bin(mod(P0*7*n,N),lng_sB) ) '"' ];
      end

      %Strings:
      s = [ s '          n_out     <= ' nstr ';' '\n' ];
      s = [ s '          P0_out    <= ' P0str ';' '\n' ];
      s = [ s '          cte2_out  <= ' cte2 ';' '\n' ];
      s = [ s '          cte3_out  <= ' cte3 ';' '\n' ];
      s = [ s '          cte4_out  <= ' cte4 ';' '\n' ];
      s = [ s '          sumBegin1 <= ' sB1 ';' '\n' ];
      s = [ s '          sumBegin2 <= ' sB2 ';' '\n' ];
      s = [ s '          sumBegin3 <= ' sB3 ';' '\n' ];
      s = [ s '          sumBegin4 <= ' sB4 ';' '\n' ];
      s = [ s '          sumBegin5 <= ' sB5 ';' '\n' ];
      s = [ s '          sumBegin6 <= ' sB6 ';' '\n' ];
      s = [ s '          sumBegin7 <= ' sB7 ';' '\n' ];
      s = [ s '\n' ];
    end
  end

  s = [ s '        else' '\n' ];
  s = [ s '          n_out     <= (others => \''0\'');' '\n' ];
  s = [ s '          P0_out    <= (others => \''0\'');' '\n' ];
  s = [ s '          cte2_out  <= (others => \''0\'');' '\n' ];
  s = [ s '          cte3_out  <= (others => \''0\'');' '\n' ];
  s = [ s '          cte4_out  <= (others => \''0\'');' '\n' ];
  s = [ s '          sumBegin1 <= (others => \''0\'');' '\n' ];
  s = [ s '          sumBegin2 <= (others => \''0\'');' '\n' ];
  s = [ s '          sumBegin3 <= (others => \''0\'');' '\n' ];
  s = [ s '          sumBegin4 <= (others => \''0\'');' '\n' ];
  s = [ s '          sumBegin5 <= (others => \''0\'');' '\n' ];
  s = [ s '          sumBegin6 <= (others => \''0\'');' '\n' ];
  s = [ s '          sumBegin7 <= (others => \''0\'');' '\n' ];
  s = [ s '\n' ];
  s = [ s '        end if;' '\n' ];
  s = [ s '\n' ];

  %WRITE TO FILE
  [fid, message] = fopen('valuesN.vhd', 'a+');
  fprintf(fidOutput, s, 'char');
  fclose(fidOutput);

end

s = [ '      -- --------------------------------'];
s = [ s '-------------------------------------' '\n' ];
s = [ s '      when others =>' '\n' ];
s = [ s '        n_out     <= (others => \''0\'');' '\n' ];
s = [ s '        P0_out    <= (others => \''0\'');' '\n' ];
```

```
s = [ s '        cte2_out  <= (others => \''0\'');' '\n' ];
s = [ s '        cte3_out  <= (others => \''0\'');' '\n' ];
s = [ s '        cte4_out  <= (others => \''0\'');' '\n' ];
s = [ s '        sumBegin1 <= (others => \''0\'');' '\n' ];
s = [ s '        sumBegin2 <= (others => \''0\'');' '\n' ];
s = [ s '        sumBegin3 <= (others => \''0\'');' '\n' ];
s = [ s '        sumBegin4 <= (others => \''0\'');' '\n' ];
s = [ s '        sumBegin5 <= (others => \''0\'');' '\n' ];
s = [ s '        sumBegin6 <= (others => \''0\'');' '\n' ];
s = [ s '        sumBegin7 <= (others => \''0\'');' '\n' ];
s = [ s '\n' ];
s = [ s '    end case;' '\n' ];
s = [ s '\n' ];
s = [ s '  end process selection;' '\n' ];
s = [ s '\n' ];
s = [ s '\n' ];
s = [ s 'end behav_valuesN;' '\n' ];

%WRITE TO FILE
[fid, message] = fopen('valuesN.vhd', 'a+');
fprintf(fidOutput, s, 'char');
fclose(fidOutput);
s = [];
```

XV. *verifORDER_indexGenerator.m*

```
clear all;
close all;
clc;

%Open the report file
fidInput = fopen('indexGenerator/report_indexGenerator_ORDER_VHDL.txt');
%Creation of the output file
[fidOutput, messageOutput] = fopen('indexGenerator/OutputReport_ORDER.txt', 'a+');
fclose(fidOutput);

line = 1;          %where the lines will be stored
endPart = 1;
lines = '';

errorVerification = 0;

%Read all the file until the end
while( line ~= -1 )

  % end of file???????
  while( endPart ~= '***' )
    line = fgetl(fidInput);

    if( length(line) > 2 )
      if( line(1:3) == 'END' )
        break;
      end
    end

    while( length(line) < 100 )
      line = [line ' '];
    end

    endPart = line(1:3);
    lines = [lines; line];
  end

  if( line(1:3) == 'END' )
    break;
  end

  %Loading data
  N = str2num(lines(1,4:end));
  Pe = str2num(lines(2,5:end));
  n = N/Pe;
  index = [];

  for i = 0:n-1
    index = [index; str2num( lines(3+i,:) )];
  end

  %Reseting the reading objets:
  lines = '';
  endPart = 1;

  %**********************************************************************
  %Calculate, verification & reporting:
  result = [];
  for i = 1:n
    for j = 1:Pe
      result(i,j) = index(i,j) - ((i-1)+(n*(j-1)));
    end
```

```
  end
  result = sum(sum(abs(result)));

  %string to write
  s = [' *************** N = ' num2str(N) '   Pe = ' num2str(Pe) ' ***************'];

  if( result == 0 )
    s1 = '       OK!!!';
  else
    s1 = '    NOT CORRECT!!!!!!!!';
    errorVerification = 1;
  end

  s = [ s '\n' s1 '\n' '\n'];

  %WRITE TO FILE
  [fidOutput, messageOutput] = fopen('indexGenerator/OutputReport_ORDER.txt', 'a+');
  fprintf(fidOutput, s, 'char');
  fclose(fidOutput);

end

%Closing the input file
fclose(fidInput);

if( errorVerification == 1 )
  s2 = '  THE ARCHITECTURE IS NOT CORRECT !!!!!!!!!!';
else
  s2 = '  THE ARCHITECTURE IS CORRECT ! :)     OK!!!';
end

s1 = '! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ';
s = [ '\n' '\n' s1 '\n' s2 '\n' s1];
%WRITE TO FILE
[fidOutput, messageOutput] = fopen('indexGenerator/OutputReport_ORDER.txt', 'a+');
fprintf(fidOutput, s, 'char');
fclose(fidOutput);
```

XVI. *verifSCRAMB_indexGenerator.m*

```
clear all;
close all;
clc;

%Open the report file
fidInput = fopen('indexGenerator/report_indexGenerator_SCRAMB_VHDL.txt');
%Creation of the output file
[fidOutput, messageOutput] = fopen('indexGenerator/OutputReport_SCRAMB.txt', 'a+');
fclose(fidOutput);

line = 1;        %where the lines will be stored
endPart = 1;
lines = '';

errorVerification = 0;

%Read all the file until the end
while( line ~= -1 )

  % end of file???????
  while( endPart ~= '***' )
    line = fgetl(fidInput);
    if( length(line) > 2 )
      if( line(1:3) == 'END' )
        break;
      end
    end

    while( length(line) < 100 )
      line = [line ' '];
    end

    endPart = line(1:3);
    lines = [lines; line];
  end

  if( line(1:3) == 'END' )
    break;
  end

  %Loading data
  N = str2num(lines(1,4:end));
  Pe = str2num(lines(2,5:end));
  n = N/Pe;
  index = [];

  for i = 0:n-1
    index = [index; str2num( lines(3+i,:) )];
  end

  %Reseting the reading objets:
  lines = '';
  endPart = 1;

  %*********************************************************************
  %Calculate, verification & reporting
  [P0, P1, P2, P3] = optimal_param_CTC( N );

  %Interleaver
  inter = interleaverFunction(N,P0,P1,P2,P3);
  index_inter = [];
  result = [];
```

```
  for i = 1:Pe

    fileIncrease = (i-1)*n;

    %Index & references to memory doing by interleaver function
    temp_index = [];
    for j = 1:n
      temp_index = [temp_index; inter(j+fileIncrease)];
    end
    index_inter = [index_inter temp_index];
  end

  result = abs(index - index_inter);
  result = sum(sum(result));

  %string to write
  s = [' *************** N = ' num2str(N) '   Pe = ' num2str(Pe) ' ***************'];

  if( result == 0 )
    s1 = '       OK!!!';
  else
    s1 = '    NOT CORRECT!!!!!!!!';
    errorVerification = 1;
  end

  s = [ s '\n' s1 '\n' '\n'];

  %WRITE TO FILE
  [fidOutput, messageOutput] = fopen('indexGenerator/OutputReport_SCRAMB.txt', 'a+');
  fprintf(fidOutput, s, 'char');
  fclose(fidOutput);

end

%Closing the input file
fclose(fidInput);

if( errorVerification == 1 )
  s2 = '  THE ARCHITECTURE IS NOT CORRECT !!!!!!!!!!';
else
  s2 = '  THE ARCHITECTURE IS CORRECT ! :)     OK!!!';
end

s1 = '! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ';
s = [ '\n' '\n' s1 '\n' s2 '\n' s1];
%WRITE TO FILE
[fidOutput, messageOutput] = fopen('indexGenerator/OutputReport_SCRAMB.txt', 'a+');
fprintf(fidOutput, s, 'char');
fclose(fidOutput);
```

XVII. *verifORDER_adxGen.m*

```
clear all;
close all;
clc;

%Open the report file
fidInput = fopen('adxGen/report_addressGen_ORDER_VHDL.txt');
%Creation of the output file
[fidOutput, messageOutput] = fopen('adxGen/OutputReport_ORDER.txt', 'a+');
fclose(fidOutput);

line = 1;        %where the lines will be stored
endPart = 1;
lines = '';

errorVerification = 0;

%Read all the file until the end
while( line ~= -1 )

  % end of file???????
  while( endPart ~= '***' )
    line = fgetl(fidInput);

    if( line(1:3) == 'END' )
      break;
    end

    while( length(line) < 100 )
      line = [line ' '];
    end

    endPart = line(1:3);
    lines = [lines; line];
  end

  if( line(1:3) == 'END' )
    break;
  end

  %Loading data
  N = str2num(lines(1,4:end));
  Pe = str2num(lines(2,5:end));
  n = N/Pe;
  index = [];
  memRef = [];
  collision = [];

  for i = 0:n-1

    %Looking for the '-'
    colls = 0;
    for j = 1:length(lines(3+i,:))
      if( lines(3+i,j) == '-' )
        indexSeparate = j;
      end
    end

    %Separating the index & the references to memory
    index = [index; str2num( lines(3+i,1:indexSeparate-1) )];
    memRef = [memRef; str2num( lines(3+i,indexSeparate+1:end) )];
  end
```

```
  %Reseting the reading objets:
  lines = '';
  endPart = 1;

  %**********************************************************************
  %Calculate:
  index_calc = [];
  memRef_calc = [];

  for i = 1:n
    temp_index = (i-1)*ones(1,Pe);
    temp_mem = 0:(Pe-1);
    index_calc = [index_calc; temp_index];
    memRef_calc = [memRef_calc; temp_mem];
  end
  %**********************************************************************

  %VERIFICATION & REPORTING
  indexVerification = sum(abs(sum(abs(index-index_calc))));
  memRefVerification = sum(abs(sum(abs(memRef-memRef_calc))));

  %string to write
  s = [' *************** N = ' num2str(N) '   Pe = ' num2str(Pe) ' ***************'];

  if( indexVerification == 0 )
    s1 = '     ADDRESS --> correct!!!';
  else
    s1 = '     ADDRESS --> NOT correct!!!  CAUTION!!!!!!!!!! ! ! ! ! ! ! !';
    errorVerification = 1;
  end
  if( memRefVerification == 0 )
    s2 = '     MEMORIES REFERENCES --> correct!!!';
  else
    s2 = '     MEMORIES REFERENCES --> NOT correct!!!  CAUTION!!!!!!!!!! ! ! ! ! ! ! !';
    errorVerification = 1;
  end

  s = [ s '\n' s1 '\n' s2 '\n' '\n'];

  %WRITE TO FILE
  [fidOutput, messageOutput] = fopen('adxGen/OutputReport_ORDER.txt', 'a+');
  fprintf(fidOutput, s, 'char');
  fclose(fidOutput);

end

%Closing the input file
fclose(fidInput);

if( errorVerification == 1 )
  s2 = '  THE ARCHITECTURE IS NOT CORRECT !!!!!!!!!!';
else
  s2 = '  THE ARCHITECTURE IS CORRECT ! :)     OK!!!';
end

s1 = '! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ';
s = [ '\n' '\n' s1 '\n' s2 '\n' s1];
%WRITE TO FILE
[fidOutput, messageOutput] = fopen('adxGen/OutputReport_ORDER.txt', 'a+');
fprintf(fidOutput, s, 'char');
fclose(fidOutput);
```

XVIII. *verifSCRAMB_adxGen.m*

```
clear all;
close all;
clc;

%Open the report file
fidInput = fopen('adxGen/report_addressGen_SCRAMB_VHDL.txt');
%Creation of the output file
[fidOutput, messageOutput] = fopen('adxGen/OutputReport_SCRAMB.txt', 'a+');
fclose(fidOutput);

line = 1;        %where the lines will be stored
endPart = 1;
lines = '';

errorVerification = 0;

%Read all the file until the end
while( line ~= -1 )

  % end of file???????
  while( endPart ~= '***' )
    line = fgetl(fidInput);
    if( line(1:3) == 'END' )
      break;
    end
    while( length(line) < 100 )
      line = [line ' '];
    end
    endPart = line(1:3);
    lines = [lines; line];
  end

  if( line(1:3) == 'END' )
    break;
  end

  %Loading data
  N = str2num(lines(1,4:end));
  Pe = str2num(lines(2,5:end));
  n = N/Pe;
  index = [];
  memRef = [];

  for i = 0:n-1
    %Looking for the '-'
    for j = 1:length(lines(3+i,:))
      if( lines(3+i,j) == '-' )
        indexSeparate = j;
      end
    end

    %Separating the index & the references to memory
    index = [index; str2num( lines(3+i,1:indexSeparate-1) )];
    memRef = [memRef; str2num( lines(3+i,indexSeparate+1:end) )];
  end

  %Reseting the reading objets:
  lines = '';
  endPart = 1;

  %**********************************************************************
  %Calculate:
```

```
  [P0, P1, P2, P3] = optimal_param_CTC( N );

  %Interleaver
  inter = interleaverFunction(N,P0,P1,P2,P3);
  index_inter = [];
  memRef_inter = [];

  for i = 1:Pe
    fileIncrease = (i-1)*n;
    %Index & references to memory doing by interleaver function
    temp_index = [];
    temp_memref = [];
    for j = 1:n
      temp_index = [temp_index; inter(j+fileIncrease)];
      temp_memref = [temp_memref; fix(inter(j+fileIncrease)/n)];
    end
    index_inter = [index_inter temp_index];
    memRef_inter = [memRef_inter temp_memref];
  end

  index_inter = mod(index_inter,n);
  %*********************************************************************

  %VERIFICATION & REPORTING
  indexVerification = sum(abs(sum(abs(index-index_inter))));
  memRefVerification = sum(abs(sum(abs(memRef-memRef_inter))));

  %string to write
  s = [' *************** N = ' num2str(N) '   Pe = ' num2str(Pe) ' ***************'];
  if( indexVerification == 0 )
    s1 = '     ADDRESS --> correct!!!';
  else
    s1 = '     ADDRESS --> NOT correct!!!  CAUTION!!!!!!!!!! ! ! ! ! ! ! !';
    errorVerification = 1;
  end
  if( memRefVerification == 0 )
    s2 = '     MEMORIES REFERENCES --> correct!!!';
  else
    s2 = '     MEMORIES REFERENCES --> NOT correct!!!  CAUTION!!!!!!!!!! ! ! ! ! ! ! !';
    errorVerification = 1;
  end
  s = [ s '\n' s1 '\n' s2 '\n' '\n'];

  %WRITE TO FILE
  [fidOutput, messageOutput] = fopen('adxGen/OutputReport_SCRAMB.txt', 'a+');
  fprintf(fidOutput, s, 'char');
  fclose(fidOutput);

end

%Closing the input file
fclose(fidInput);

if( errorVerification == 1 )
  s2 = '  THE ARCHITECTURE IS NOT CORRECT !!!!!!!!!!';
else
  s2 = '  THE ARCHITECTURE IS CORRECT ! :)     OK!!!';
end
s1 = '! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ';
s = [ '\n' '\n' s1 '\n' s2 '\n' s1];
%WRITE TO FILE
[fidOutput, messageOutput] = fopen('adxGen/OutputReport_SCRAMB.txt', 'a+');
fprintf(fidOutput, s, 'char');
fclose(fidOutput);
```

XIX. *verifORDER_SRM.m*

```
clear all;
close all;
clc;

%Open the report file
fidInput = fopen('switchReadMemory/report_SRM_ORDER_VHDL.txt');
%Creation of the output file
[fidOutput, messageOutput] = fopen('switchReadMemory/OutputReport_ORDER.txt', 'a+');
fclose(fidOutput);

line = 1;        %where the lines will be stored
endPart = 1;
lines = '';

errorVerification = 0;

%Read all the file until the end
while( line ~= -1 )

  % end of file???????
  while( endPart ~= '***' )
    line = fgetl(fidInput);

    if( length(line) > 2 )
      if( line(1:3) == 'END' )
        break;
      end
    end

    while( length(line) < 100 )
      line = [line ' '];
    end

    endPart = line(1:3);
    lines = [lines; line];
  end

  if( line(1:3) == 'END' )
    break;
  end

  %Loading data
  N = str2num(lines(1,4:end));
  Pe = str2num(lines(2,5:end));
  n = N/Pe;
  index = [];

  for i = 0:n-1
    index = [index; str2num( lines(3+i,:) )];
  end

  %Reseting the reading objets:
  lines = '';
  endPart = 1;

  %*********************************************************************
  %Calculate, verification & reporting:
  result = [];
  for i = 1:n
    for j = 1:Pe
      result(i,j) = index(i,j) - ((10000*(j-1))+(i-1));
    end
```

```
  end
  result = sum(sum(abs(result)));

  %string to write
  s = [' *************** N = ' num2str(N) '   Pe = ' num2str(Pe) ' ***************'];

  if( result == 0 )
    s1 = '       OK!!!';
  else
    s1 = '    NOT CORRECT!!!!!!!!';
    errorVerification = 1;
  end

  s = [ s '\n' s1 '\n' '\n'];

  %WRITE TO FILE
  [fidOutput, messageOutput] = fopen('switchReadMemory/OutputReport_ORDER.txt', 'a+');
  fprintf(fidOutput, s, 'char');
  fclose(fidOutput);

end

%Closing the input file
fclose(fidInput);

if( errorVerification == 1 )
  s2 = '  THE ARCHITECTURE IS NOT CORRECT !!!!!!!!!!';
else
  s2 = '  THE ARCHITECTURE IS CORRECT ! :)     OK!!!';
end

s1 = '! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ';
s = [ '\n' '\n' s1 '\n' s2 '\n' s1];
%WRITE TO FILE
[fidOutput, messageOutput] = fopen('switchReadMemory/OutputReport_ORDER.txt', 'a+');
fprintf(fidOutput, s, 'char');
fclose(fidOutput);
```

XX. *verifSCRAMB_SRM.m*

```
clear all;
close all;
clc;

%Open the report file
fidInput = fopen('switchReadMemory/report_SRM_SCRAMB_VHDL.txt');
%Creation of the output file
[fidOutput, messageOutput] = fopen('switchReadMemory/OutputReport_SCRAMB.txt', 'a+');
fclose(fidOutput);

line = 1;        %where the lines will be stored
endPart = 1;
lines = '';

errorVerification = 0;

%Read all the file until the end
while( line ~= -1 )

  % end of file???????
  while( endPart ~= '***' )
    line = fgetl(fidInput);

    if( length(line) > 2 )
      if( line(1:3) == 'END' )
        break;
      end
    end

    while( length(line) < 100 )
      line = [line ' '];
    end

    endPart = line(1:3);
    lines = [lines; line];
  end

  if( line(1:3) == 'END' )
    break;
  end

  %Loading data
  N = str2num(lines(1,4:end));
  Pe = str2num(lines(2,5:end));
  n = N/Pe;
  index = [];

  for i = 0:n-1
    index = [index; str2num( lines(3+i,:) )];
  end

  %Reseting the reading objets:
  lines = '';
  endPart = 1;

  %**********************************************************************
  %Calculate, verification & reporting
  [P0, P1, P2, P3] = optimal_param_CTC( N );

  %Interleaver
  inter = interleaverFunction(N,P0,P1,P2,P3);
  index_inter = [];
```

```
  memRef_inter = [];
  result_inter = [];
  result = [];

  for i = 1:Pe

    fileIncrease = (i-1)*n;

    %Index & references to memory doing by interleaver function
    temp_index = [];
    temp_memref = [];
    for j = 1:n
      temp_index = [temp_index; inter(j+fileIncrease)];
      temp_memref = [temp_memref; fix(inter(j+fileIncrease)/n)];
    end
    index_inter = [index_inter temp_index];
    memRef_inter = [memRef_inter temp_memref];
  end
  index_inter = mod(index_inter,n);
  memRef_inter = memRef_inter*10000;
  result_inter = index_inter + memRef_inter;

  result = abs(result_inter - index);
  result = sum(sum(result));

  %string to write
  s = [' *************** N = ' num2str(N) '   Pe = ' num2str(Pe) ' ***************'];

  if( result == 0 )
    s1 = '       OK!!!';
  else
    s1 = '    NOT CORRECT!!!!!!!!';
    errorVerification = 1;
  end

  s = [ s '\n' s1 '\n' '\n'];

  %WRITE TO FILE
  [fidOutput, messageOutput] = fopen('switchReadMemory/OutputReport_SCRAMB.txt', 'a+');
  fprintf(fidOutput, s, 'char');
  fclose(fidOutput);

end

%Closing the input file
fclose(fidInput);

if( errorVerification == 1 )
  s2 = '  THE ARCHITECTURE IS NOT CORRECT !!!!!!!!!!';
else
  s2 = '  THE ARCHITECTURE IS CORRECT ! :)     OK!!!';
end

s1 = '! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ';
s = [ '\n' '\n' s1 '\n' s2 '\n' s1];
%WRITE TO FILE
[fidOutput, messageOutput] = fopen('switchReadMemory/OutputReport_SCRAMB.txt', 'a+');
fprintf(fidOutput, s, 'char');
fclose(fidOutput);
```

XXI. *verifORDER_SWM.m*

```
clear all;
close all;
clc;

%Open the report file
fidInput = fopen('switchWriteMemory/reportSWM_ORDER_VHDL.txt');
%Creation of the output file
[fidOutput, messageOutput] = fopen('switchWriteMemory/OutputReport_ORDER.txt', 'a+');
fclose(fidOutput);

line = 1;        %where the lines will be stored
endPart = 1;
lines = '';

errorVerification = 0;

%FILLING SISOs WITH DATA
SISO = [];
for i = 1:8
  temp = (10000*i)+(0:2399)';
  SISO = [SISO temp];
  clear temp;
end

%Read all the file until the end
while( line ~= -1 )

  % end of file???????
  while( endPart ~= '***' )
    line = fgetl(fidInput);

    if( line(1:3) == 'END' )
      break;
    end

    while( length(line) < 100 )
      line = [line ' '];
    end

    endPart = line(1:3);
    lines = [lines; line];
  end

  if( line(1:3) == 'END' )
    break;
  end

  %Loading data
  N = str2num(lines(1,4:end));
  Pe = str2num(lines(2,5:end));
  n = N/Pe;

  indexREADING = str2num(lines(3:end-1,:));

  %Reseting the reading objets:
  lines = '';
  endPart = 1;

  %**********************************************************************
  %Calculate:
  index_calc = [];
  memRef_calc = [];
```

```
  for i = 1:n
    temp_index = (i-1)*ones(1,Pe);
    temp_mem = 0:(Pe-1);
    index_calc = [index_calc; temp_index];
    memRef_calc = [memRef_calc; temp_mem];
  end

  %Matrix Investing
  index_calc = matrixInvest(index_calc);
  memRef_calc = matrixInvest(memRef_calc);

  %Writing to MEMORY
  MEM = zeros(2500,8);
  for i = 1:n
    for ii = 1:Pe
      MEM(index_calc(i,ii)+1,memRef_calc(i,ii)+1) = SISO(i,ii);
    end
  end
  %*********************************************************************

  %VERIFICATION & REPORTING
  indexVerification = sum(sum(abs(MEM-indexREADING)));

  %string to write
  s = [' *************** N = ' num2str(N) '   Pe = ' num2str(Pe) ' ***************'];

  if( indexVerification == 0 )
    s1 = '     OK';
  else
    s1 = '     --> NOT correct!!!  CAUTION!!!!!!!!!';
    errorVerification = 1;
  end

  s = [ s '\n' s1 '\n' '\n'];

  %WRITE TO FILE
  [fidOutput, messageOutput] = fopen('switchWriteMemory/OutputReport_ORDER.txt', 'a+');
  fprintf(fidOutput, s, 'char');
  fclose(fidOutput);

end

%Closing the input file
fclose(fidInput);

if( errorVerification == 1 )
  s2 = '  THE ARCHITECTURE IS NOT CORRECT !!!!!!!!!!';
else
  s2 = '  THE ARCHITECTURE IS CORRECT ! :)     OK!!!';
end

s1 = '! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ';
s = [ '\n' '\n' s1 '\n' s2 '\n' s1];
%WRITE TO FILE
[fidOutput, messageOutput] = fopen('switchWriteMemory/OutputReport_ORDER.txt', 'a+');
fprintf(fidOutput, s, 'char');
fclose(fidOutput);
```

XXII. *verifSCRAMB_SWM.m*

```
clear all;
close all;
clc;

%Open the report file
fidInput = fopen('switchWriteMemory/reportSWM_SCRAMB_VHDL.txt');
%Creation of the output file
[fidOutput, messageOutput] = fopen('switchWriteMemory/OutputReport_SCRAMB.txt', 'a+');
fclose(fidOutput);

line = 1;        %where the lines will be stored
endPart = 1;
lines = '';

errorVerification = 0;

%FILLING SISOs WITH DATA
SISO = [];
for i = 1:8
  temp = (10000*i)+(0:2399)';
  SISO = [SISO temp];
  clear temp;
end

%Read all the file until the end
while( line ~= -1 )

  % end of file???????
  while( endPart ~= '***' )
    line = fgetl(fidInput);

    if( line(1:3) == 'END' )
      break;
    end

    while( length(line) < 100 )
      line = [line ' '];
    end

    endPart = line(1:3);
    lines = [lines; line];
  end

  if( line(1:3) == 'END' )
    break;
  end

  %Loading data
  N = str2num(lines(1,4:end));
  Pe = str2num(lines(2,5:end));
  n = N/Pe;
  indexREADING = str2num(lines(3:end-1,:));

  %Reseting the reading objets:
  lines = '';
  endPart = 1;

  %**********************************************************************
  %Calculate:
  [P0, P1, P2, P3] = optimal_param_CTC( N );

  %Interleaver
```

```
  inter = interleaverFunction(N,P0,P1,P2,P3);
  index_inter = [];
  memRef_inter = [];

  for i = 1:Pe
    fileIncrease = (i-1)*n;
    %Index & references to memory doing by interleaver function
    temp_index = [];
    temp_memref = [];
    for j = 1:n
      temp_index = [temp_index; inter(j+fileIncrease)];
      temp_memref = [temp_memref; fix(inter(j+fileIncrease)/n)];
    end
    index_inter = [index_inter temp_index];
    memRef_inter = [memRef_inter temp_memref];
  end
  index_inter = mod(index_inter,n);

  %Matrix Investing
  index_inter = matrixInvest(index_inter);
  memRef_inter = matrixInvest(memRef_inter);

  %Writing to MEMORY
  MEM = zeros(2500,8);
  for i = 1:n
    for ii = 1:Pe
      MEM(index_inter(i,ii)+1,memRef_inter(i,ii)+1) = SISO(i,ii);
    end
  end
  %*********************************************************************

  %VERIFICATION & REPORTING
  indexVerification = sum(sum(abs(MEM-indexREADING)));
  %string to write
  s = [' *************** N = ' num2str(N) '   Pe = ' num2str(Pe) ' ***************'];
  if( indexVerification == 0 )
    s1 = '     OK';
  else
    s1 = '     --> NOT correct!!!  CAUTION!!!!!!!!!';
    errorVerification = 1;
  end
  s = [ s '\n' s1 '\n' '\n'];
  %WRITE TO FILE
  [fidOutput, messageOutput] = fopen('switchWriteMemory/OutputReport_SCRAMB.txt', 'a+');
  fprintf(fidOutput, s, 'char');
  fclose(fidOutput);

end

%Closing the input file
fclose(fidInput);

if( errorVerification == 1 )
  s2 = '  THE ARCHITECTURE IS NOT CORRECT !!!!!!!!!!';
else
  s2 = '  THE ARCHITECTURE IS CORRECT ! :)     OK!!!';
end
s1 = '! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ';
s = [ '\n' '\n' s1 '\n' s2 '\n' s1];
%WRITE TO FILE
[fidOutput, messageOutput] = fopen('switchWriteMemory/OutputReport_SCRAMB.txt', 'a+');
fprintf(fidOutput, s, 'char');
fclose(fidOutput);
```

XXIII. *verif_INTERLEAVER.m*

```
clear all;
close all;
clc;

%-----------------------------------------------------------------------
%-----------------------------------------------------------------------
%FIRST, verify the write into SISOs:

%Open the report file
fidInput = fopen('interleaver/report_INTER_SISOs_VHDL.txt');

line = 1;        %where the lines will be stored
endPart = 1;
lines = '';
reporting_A = [];
parameters = [];

%Read all the file until the end
while( line ~= -1 )

  % end of file???????
  while( endPart ~= '***' )
    line = fgetl(fidInput);

    if( length(line) > 2 )
      if( line(1:3) == 'END' )
        break;
      end
    end

    while( length(line) < 100 )
      line = [line ' '];
    end

    endPart = line(1:3);
    lines = [lines; line];
  end

  if( line(1:3) == 'END' )
    break;
  end

  %Loading data
  N = str2num(lines(1,4:end));
  Pe = str2num(lines(2,5:end));
  BL = str2num(lines(3,18:end));
  n = N/Pe;
  iterations = n/BL;
  index = [];

  for i = 0:n-1
    index = [index; str2num( lines(4+i,:) )];
  end

  %Reseting the reading objets:
  lines = '';
  endPart = 1;

  %*********************************************************************
  %Calculate, verification & reporting

  %ORDER:
```

```
order = [];
for i = 1:n
  for j = 1:Pe
    order(i,j) = ((10000*(j-1))+(i-1));
  end
end
%END ORDER

%SCRAMBLED
[P0, P1, P2, P3] = optimal_param_CTC( N );
inter = interleaverFunction(N,P0,P1,P2,P3);
index_inter = [];
memRef_inter = [];
scrambled = [];
for i = 1:Pe
  fileIncrease = (i-1)*n;
  %Index & references to memory doing by interleaver function
  temp_index = [];
  temp_memref = [];
  for j = 1:n
    temp_index = [temp_index; inter(j+fileIncrease)];
    temp_memref = [temp_memref; fix(inter(j+fileIncrease)/n)];
  end
  index_inter = [index_inter temp_index];
  memRef_inter = [memRef_inter temp_memref];
end
index_inter = mod(index_inter,n);
memRef_inter = memRef_inter*10000;
scrambled = index_inter + memRef_inter;
%END SCRAMBLED

%SWITCHING ORDER/SCRAMBLED
result = [];
firstINDEX = 1;
lastINDEX = BL;
scram = 0;

for i = 1:iterations
  if( scram == 0 )
    result(firstINDEX:lastINDEX,:) = order(firstINDEX:lastINDEX,:);
  else
    result(firstINDEX:lastINDEX,:) = scrambled(firstINDEX:lastINDEX,:);
  end

  firstINDEX = firstINDEX + BL;
  lastINDEX = lastINDEX + BL;
  scram = not( scram );

end

%VERIFYING
VERIFYING = abs(result - index);
VERIFYING = sum(sum(VERIFYING));

%REPORTING:
%      0 -> OK!!!!
%      1 -> NOT CORRECT!!!
if( VERIFYING == 0 )
  reporting_A = [reporting_A 0];
else
  reporting_A = [reporting_A 1];
end

s = ['  ----  N = ' num2str(N) '  ----  Pe = ' num2str(Pe)];
s = [s '  ----  BURST = ' num2str(BL) '  ----'];
```

```
  parameters = strvcat(parameters,s);

end

fclose(fidInput);
clear fidInput;


%--------------------------------------------------------------------
%--------------------------------------------------------------------
%SECOND, verify the write into memory:

%Open the report file
fidInput = fopen('interleaver/report_INTER_MEMs_VHDL.txt');

line = 1;        %where the lines will be stored
endPart = 1;
lines = '';
reporting_B = [];

SISO = [];
for i = 1:8
  temp = (10000*i)+(0:2399)';
  SISO = [SISO temp];
  clear temp;
end

%Read all the file until the end
while( line ~= -1 )

  % end of file???????
  while( endPart ~= '***' )
    line = fgetl(fidInput);

    if( line(1:3) == 'END' )
      break;
    end

    while( length(line) < 100 )
      line = [line ' '];
    end

    endPart = line(1:3);
    lines = [lines; line];
  end

  if( line(1:3) == 'END' )
    break;
  end

  %Loading data
  N = str2num(lines(1,4:end));
  Pe = str2num(lines(2,5:end));
  BL = str2num(lines(3,18:end));
  n = N/Pe;
  iterations = n/BL;

  indexREADING = str2num(lines(4:end-1,:));

  %Reseting the reading objets:
  lines = '';
  endPart = 1;

  %*********************************************************************
  %Calculate:
```

```
%Addressing ORDER:
index_ORDER = [];
memRef_ORDER = [];
for i = 1:n
  temp_index = (i-1)*ones(1,Pe);
  temp_mem = 0:(Pe-1);
  index_ORDER = [index_ORDER; temp_index];
  memRef_ORDER = [memRef_ORDER; temp_mem];
end

%Addressing SCRAMBLED:
[P0, P1, P2, P3] = optimal_param_CTC( N );
inter = interleaverFunction(N,P0,P1,P2,P3);
index_SCRAMB = [];
memRef_SCRAMB = [];
for i = 1:Pe
  fileIncrease = (i-1)*n;
  %Index & references to memory doing by interleaver function
  temp_index = [];
  temp_memref = [];
  for j = 1:n
    temp_index = [temp_index; inter(j+fileIncrease)];
    temp_memref = [temp_memref; fix(inter(j+fileIncrease)/n)];
  end
  index_SCRAMB = [index_SCRAMB temp_index];
  memRef_SCRAMB = [memRef_SCRAMB temp_memref];
end
index_SCRAMB = mod(index_SCRAMB,n);

%SWITCHING ADDRESSING ORDER/SCRAMBLED & LIFO "INVEST"
index = [];
memRef = [];
firstINDEX = 1;
lastINDEX = BL;
scram = 0;
for i = 1:iterations
  if( scram == 0 )
    %ORDER
    if( BL > 1)
      index = [index; matrixInvest(index_ORDER(firstINDEX:lastINDEX,:))];
      memRef = [memRef; matrixInvest(memRef_ORDER(firstINDEX:lastINDEX,:))];
    else
      index = [index; index_ORDER(firstINDEX:lastINDEX,:)];
      memRef = [memRef; memRef_ORDER(firstINDEX:lastINDEX,:)];
    end
  else
    %SCRAMB
    if( BL > 1)
      index = [index; matrixInvest(index_SCRAMB(firstINDEX:lastINDEX,:))];
      memRef = [memRef; matrixInvest(memRef_SCRAMB(firstINDEX:lastINDEX,:))];
    else
      index = [index; index_SCRAMB(firstINDEX:lastINDEX,:)];
      memRef = [memRef; memRef_SCRAMB(firstINDEX:lastINDEX,:)];
    end
  end
  firstINDEX = firstINDEX + BL;
  lastINDEX = lastINDEX + BL;
  scram = not( scram );
end

%Writing to MEMORY
MEM = zeros(2500,8);
for i = 1:n
  for ii = 1:Pe
    MEM(index(i,ii)+1,memRef(i,ii)+1) = SISO(i,ii);
```

```
    end
  end

  %VERIFYING
  VERIFYING = sum(sum(abs(MEM - indexREADING)));

  %REPORTING:
  %      0 -> OK!!!!
  %      1 -> NOT CORRECT!!!
  if( VERIFYING == 0 )
    reporting_B = [reporting_B 0];
  else
    reporting_B = [reporting_B 1];
  end

end
fclose(fidInput);
clear fidInput;


%OUTPUT FILE
for i = 1:length(reporting_A)
  [fidOutput, messageOutput] = fopen('interleaver/OutputReport.txt', 'a+');
  s = '';
  s = [s parameters(i,:) '\n'];
  if( reporting_A(i) == 0 )
    s = [s '      Writing into SISO --> OK!!!' '\n'];
  else
    s = [s '  ->  Writing into SISO --> NOT CORRECT!!!!!' '\n'];
  end
  if( reporting_B(i) == 0 )
    s = [s '      Writing into MEMORY --> OK!!!' '\n'];
  else
    s = [s '  ->  Writing into MEMORY --> NOT CORRECT!!!!!' '\n'];
  end
  s = [s '\n'];
  fprintf(fidOutput, s, 'char');
  fclose(fidOutput);
end

if( sum(reporting_A)+sum(reporting_B) == 0 )
  s = ['\n' '\n' '   ALL IS CORRECT!!!!!!! :D' ];
else
  s = ['\n' '\n' '   NOT CORRECT!!!!!!! :(' ];
end
[fidOutput, messageOutput] = fopen('interleaver/OutputReport.txt', 'a+');
fprintf(fidOutput, s, 'char');
fclose(fidOutput);
```

ANEXO II

COMPONENTES Y TESTBENCHS VHDL

I. *interleaverAlgorithm.vhd*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

-- ===========================================================================
--
--              INTERLEAVER ALGORITHM
--
--      The algorithm needs one cycle to begin
--
-- ===========================================================================

entity interleaverAlgorithm is

  port (
    clk        : in  std_logic;
    enable     : in  std_logic;
    clear      : in  std_logic;
    rst_n      : in  std_logic;
    scrambled  : in  std_logic;
    N_in       : in  std_logic_vector(11 downto 0);
    P0_in      : in  std_logic_vector(5 downto 0);
    sumBegin   : in  std_logic_vector(11 downto 0);
    cte2       : in  std_logic_vector(10 downto 0);
    cte3       : in  std_logic_vector(8 downto 0);
    cte4       : in  std_logic_vector(10 downto 0);
    sub_n      : in  std_logic_vector(11 downto 0);
    multiple_n : in  std_logic_vector(11 downto 0);
    offset     : in  std_logic_vector(2 downto 0);
    indexValid : out std_logic;
    index      : out std_logic_vector(11 downto 0));

end interleaverAlgorithm;


architecture behav_interleaverAlgorithm of interleaverAlgorithm is

  signal tempA : std_logic_vector(11 downto 0) := (others => '0');
  signal tempB : std_logic_vector(11 downto 0) := (others => '0');
  signal tempC : std_logic_vector(11 downto 0) := (others => '0');
  signal firstAdder : std_logic_vector(5 downto 0) := (others => '0');
  signal RESULT : std_logic_vector(11 downto 0) := (others => '0');

  signal acumulator : std_logic_vector(11 downto 0) := (others => '0');
  signal cte : std_logic_vector(10 downto 0) := (others => '0');

  signal algorithmInitiate : std_logic := '0';

  -- Auxiliar COUNTER signals
  signal count : std_logic_vector(11 downto 0) := (others => '0');
  signal countValid : std_logic := '0';

  -- Auxiliar COUNT SELECTOR signals:
  signal MUXcountSelec : std_logic_vector(1 downto 0) := (others => '0');


begin  -- behav_interleaverAlgorithm


  -- -------------------------------------------------------------------------
  --            ASYNCHRONOUS ALGORITHM OPERATIONS:
```

```
--
--   ACUMULATOR        IF > N                    IF > N
--       +        ==>     -N       ==>   + CTE   ==>     -N
--    FIRST_ADD
--
--    (tempA)         (tempB)        (tempC)    (RESULT)
--
tempA <= acumulator + firstAdder;

tempB <= tempA when (tempA < N_in) else
         (tempA - N_in);

tempC <= tempB + cte;

RESULT <= tempC when (tempC < N_in) else
          (tempC - N_in);


-- --------------------------------------------------------------------------
--                    Auxiliar PROCESS: COUNT SELECTOR
--
--            PURPOSE: Select the begin of the internal MUX (cte's) at the
--    interleaver algorithm, depends of "n" (if it's divisible by 4 or not).
--    Also, depends of the parallel process that the interleaver has to do.
--
--
--       offSet ->          0     1     2     3     4     5     6     7
--                        |-------------------------------------------------|
--   2LSB  sub_n  -> "00" | count count count count count count count count |
--                        |                                                 |
--                   "01" | count +"01" +"10" +"11" count +"01" +"10" +"11" |
--                        |                                                 |
--                   "10" | count +"10" count +"10" count +"10" count +"10" |
--                        |                                                 |
--                   "11" | count +"11" +"10" +"01" count +"11" +"10" +"01" |
--                        |-------------------------------------------------|
--
countSelect: process (offset, sub_n, count)

  variable subN : std_logic_vector(1 downto 0) := (others => '0');
  variable cnt : std_logic_vector(1 downto 0) := (others => '0');

begin  -- process countSelect

  -- Filling variables:
  subN := sub_n(1 downto 0);
  cnt := count(1 downto 0);

  -- count_2LSB
  if (subN = "00" or (offset = "000" or offset = "100") or
      (subN = "10" and (offset = "010" or offset = "110"))) then
    MUXcountSelec <= cnt;

  elsif (subN = "00") then
    MUXcountSelec <= cnt;

  -- count_2LSB + "01"
  elsif ((subN = "01" and (offset = "001" or offset = "101")) or
         (subN = "11" and (offset = "011" or offset = "111"))) then
    MUXcountSelec <= cnt + "01";

  -- count_2LSB + "10"
  elsif (((subN = "01" or subN = "11") and (offset = "010" or offset = "110")) or
         (subN = "10" and (offset = "001" or offset = "011" or offset = "101" or offset = "111")))
then
```

```
    MUXcountSelec <= cnt + "10";

  -- count_2LSB + "11"
  elsif ((subN = "01" and (offset = "011" or offset = "111")) or
         (subN = "11" and (offset = "001" or offset = "101"))) then
    MUXcountSelec <= cnt + "11";

  else
    MUXcountSelec <= (others => '0');

  end if;

end process countSelect;


-- --------------------------------------------------------------------------
--    PROCESS
-- Selection of the constant to sum to "aux1"
--    00 -> 1
--    01 -> P2 -> cte2
--    10 -> P3 -> cte3
--    11 -> P4 -> cte4
selCte: process (MUXcountSelec, cte2, cte3, cte4)
begin  -- process selCte

  if (MUXcountSelec = "00") then      -- case 0
    cte <= conv_std_logic_vector(1,11);
  elsif (MUXcountSelec = "01") then   -- case 1
    cte <= cte2;
  elsif (MUXcountSelec = "10") then   -- case 2
    cte(8 downto 0) <= cte3;
    cte(10 downto 9) <= (others => '0');
  elsif (MUXcountSelec = "11") then   -- case 3
    cte <= cte4;
  else
    cte <= (others => '0');
  end if;

end process selCte;


-- --------------------------------------------------------------------------
--    PROCESS
-- Counter & interleaver algorithm
count_IA: process (clk, rst_n)
begin  -- process count_IA

  if (rst_n = '0') then
    algorithmInitiate <= '0';
    acumulator <= (others => '0');
    firstAdder <= (others => '0');
    -- COUNTER signals:
    count <= (others => '0');
    countValid <= '0';
    -- OUTPUT signals:
    index <= (others => '0');
    indexValid <= '0';


  -- SYNC: Rising edge
  elsif (clk = '1' and clk'event) then
    if (clear = '1') then
      algorithmInitiate <= '0';
      acumulator <= (others => '0');
      firstAdder <= (others => '0');
```

```
    -- COUNTER signals:
    count <= (others => '0');
    countValid <= '0';
    -- OUTPUT signals:
    index <= (others => '0');
    indexValid <= '0';



  -- ENABLE NOT ACTIVE
  elsif (enable = '0') then
    indexValid <= '0';

  -- ENABLE ACTIVE
  elsif (enable = '1') then

    -- --------------------------------------------------------------------
    -- INTERLEAVER ALGORITHM
    if (algorithmInitiate = '0') then
      algorithmInitiate <= '1';
      acumulator <= sumBegin;

    elsif (algorithmInitiate = '1') then
      firstAdder <= P0_in;
      acumulator <= tempB;

    end if;
    -- --------------------------------------------------------------------

    -- -------------------------------------------------------------------
    -- COUNTER
    if (countValid = '0') then
      count <= (others => '0');
      countValid <= '1';
    else
      if (count < sub_n-1) then
        count <= count + '1';
      else
        -- Ended reset
        count <= (others => '0');
        countValid <= '0';
        algorithmInitiate <= '0';
        firstAdder <= (others => '0');
        acumulator <= (others => '0');
      end if;
    end if;
    -- --------------------------------------------------------------------

    -- --------------------------------------------------------------------
    -- SELECTING "ORDER" OR "SCRAMBLED"
    --       0 -> ORDER
    --       1 -> SCRAMBLED
    if (countValid = '1') then
      indexValid <= '1';

      if (scrambled = '0') then      -- ORDER
        index <= count + multiple_n;
      elsif (scrambled = '1') then   -- SCRAMBLED
        index <= RESULT;
      end if;

    else
      indexValid <= '0';
      index <= (others => '0');
    end if;
```

```
        -- --------------------------------------------------------------------

      end if;  -- enable & clear

    end if;  -- clk & rst_n

  end process count_IA;

end behav_interleaverAlgorithm;
```

II. *indexGenerator.vhd*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

-- ----------------------------------------------------------------------------
--
--               INDEX GENERATOR
--
--       Generate index in order & scrambled, creating also a bus with
--     multiples of 'n' that will be needed in other modules.
--
-- ----------------------------------------------------------------------------


entity indexGenerator is

  port (
    clk       : in  std_logic;
    enable    : in  std_logic;
    clear     : in  std_logic;
    rst_n     : in  std_logic;
    scrambled : in  std_logic;          -- order(0) or scrambled(1)
    N_in      : in  std_logic_vector(11 downto 0);
    Pe_in     : in  std_logic_vector(3 downto 0);
    validOut  : out std_logic;
    indexOut  : out std_logic_vector(95 downto 0);
    mult_nBus : out std_logic_vector(83 downto 0));  -- n to 7n

end indexGenerator;


architecture behav_indexGenerator of indexGenerator is


  -- --------------------------------------------------------------------------
  -- Auxiliar components: VALUES_N and INTERLEAVER_ALGORITHM

  component valuesN
    port (
      N_in      : in  std_logic_vector(11 downto 0);
      Pe_in     : in  std_logic_vector(3 downto 0);
      n_out     : out std_logic_vector(11 downto 0);
      P0_out    : out std_logic_vector(5 downto 0);
      cte2_out  : out std_logic_vector(10 downto 0);
      cte3_out  : out std_logic_vector(8 downto 0);
      cte4_out  : out std_logic_vector(10 downto 0);
      sumBegin1 : out std_logic_vector(11 downto 0);
      sumBegin2 : out std_logic_vector(11 downto 0);
      sumBegin3 : out std_logic_vector(11 downto 0);
      sumBegin4 : out std_logic_vector(11 downto 0);
      sumBegin5 : out std_logic_vector(11 downto 0);
      sumBegin6 : out std_logic_vector(11 downto 0);
      sumBegin7 : out std_logic_vector(11 downto 0));
  end component;

  component interleaverAlgorithm
    port (
      clk        : in  std_logic;
      enable     : in  std_logic;
      clear      : in  std_logic;
      rst_n      : in  std_logic;
```

```
      scrambled   : in  std_logic;
      N_in        : in  std_logic_vector(11 downto 0);
      P0_in       : in  std_logic_vector(5 downto 0);
      sumBegin    : in  std_logic_vector(11 downto 0);
      cte2        : in  std_logic_vector(10 downto 0);
      cte3        : in  std_logic_vector(8 downto 0);
      cte4        : in  std_logic_vector(10 downto 0);
      sub_n       : in  std_logic_vector(11 downto 0);
      multiple_n  : in  std_logic_vector(11 downto 0);
      offset      : in  std_logic_vector(2 downto 0);
      indexValid  : out std_logic;
      index       : out std_logic_vector(11 downto 0));
  end component;


  -- --------------------------------------------------------------------------
  -- Auxiliar "VALUES_N" signals
  signal nSub : std_logic_vector(11 downto 0) := (others => '0');
  signal P0 : std_logic_vector(5 downto 0) := (others => '0');
  signal cte2 : std_logic_vector(10 downto 0) := (others => '0');
  signal cte3 : std_logic_vector(8 downto 0) := (others => '0');
  signal cte4 : std_logic_vector(10 downto 0) := (others => '0');
  type arrayAux is array (7 downto 0) of std_logic_vector(11 downto 0);
  signal sumBegin : arrayAux := (others => (others => '0'));

  -- Auxiliar type for store the "n" MULTIPLES
  signal multiple_n : arrayAUX := (others => (others => '0'));

  -- Auxiliar signals: store the index of the "interleaver Algorithms" & valids
  signal indexArray : arrayAUX := (others => (others => '0'));
  signal indexValid : std_logic_vector(7 downto 0) := (others => '0');

  -- Auxiliar signal & type: store the offset's
  type arrayOffSet is array (7 downto 0) of std_logic_vector(2 downto 0);
  constant offsetArray : arrayOffSet := ( "111", "110", "101", "100",
                                          "011", "010", "001", "000");


begin  -- behav_indexGenerator


  -- Array -> Bus
  arrayToBus: for i in 6 downto 0 generate
    mult_nBus(12*(i+1)-1 downto 12*i) <= multiple_n(i+1);
  end generate arrayToBus;


  -- Unite the "indexValid"s in a signal at the output
  validOut <= '1' when (indexValid = "11111111") else
              '0';


  -- --------------------------------------------------------------------------
  --             CALCULATING "n" MULTIPLES
  --     multiple_n(0) <= 0
  --     multiple_n(1) <= n
  --     multiple_n(2) <= 2n
  --     multiple_n(3) <= 3n
  --     multiple_n(4) <= 4n
  --     multiple_n(5) <= 5n
  --     multiple_n(6) <= 6n
  --     multiple_n(7) <= 7n
  multiple_n(0) <= (others => '0');
  multiple_n(1) <= nSub;
  multiple_n(2)(11 downto 1) <= nSub(10 downto 0);
```

```
  multiple_n(2)(0) <= '0';
  multiple_n(3) <= nSub + multiple_n(2);
  multiple_n(4)(11 downto 2) <= nSub(9 downto 0);
  multiple_n(4)(1 downto 0) <= "00";
  multiple_n(5) <= nSub + multiple_n(4);
  multiple_n(6) <= multiple_n(2) + multiple_n(4);
  multiple_n(7) <= multiple_n(3) + multiple_n(4);


  -- -------------------------------------------------------------------------
  --            COMPONENT --> valuesN
  values : valuesN
    port map (
      N_in       => N_in,
      Pe_in      => Pe_in,
      n_out      => nSub,
      P0_out     => P0,
      cte2_out   => cte2,
      cte3_out   => cte3,
      cte4_out   => cte4,
      sumBegin1 => sumBegin(1),
      sumBegin2 => sumBegin(2),
      sumBegin3 => sumBegin(3),
      sumBegin4 => sumBegin(4),
      sumBegin5 => sumBegin(5),
      sumBegin6 => sumBegin(6),
      sumBegin7 => sumBegin(7));

  --    The first "sumBegin" -> ZEROS
  sumBegin(0) <= (others => '0');


  -- -------------------------------------------------------------------------
  --            COMPONENTs --> interleaverAlgorithm's
  --    Doing a generate to make 8 "interleaverAlgorithm's"
  genInterAlg: for i in 7 downto 0 generate

    interAlg : interleaverAlgorithm
      port map (
        clk         => clk,
        enable      => enable,
        clear       => clear,
        rst_n       => rst_n,
        scrambled   => scrambled,
        N_in        => N_in,
        P0_in       => P0,
        sumBegin    => sumBegin(i),
        cte2        => cte2,
        cte3        => cte3,
        cte4        => cte4,
        sub_n       => nSub,
        multiple_n => multiple_n(i),
        offset      => offsetArray(i),
        indexValid => indexValid(i),
        index       => indexArray(i));

    -- Array to bus
    indexOut(12*(i+1)-1 downto 12*i) <= indexArray(i);

  end generate genInterAlg;

end behav_indexGenerator;
```

III. *addressTranslate.vhd*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

-- ===========================================================================
--
--                      ADDRESS TRANSLATE
--
--      Give us the information about which memories must read/write and
--  his real address
--
--   INPUTS:    Pe_in => process parallels
--              indexBus => gets the index of all processes parallels
--              mult_nBus => contains the values of multiples of 'n' & 'n'
--
--   OUTPUTS:   validOut => says if outputs ara valid
--              addressBus => real address of the memory referenced with "ref_mem"
--                            [address_MEM7 address_MEM6 ... address_MEM0]
--
--              mem_refBus => indicates the memory where read/write [0 to 7]
--                              MEM 0 => "000"
--                              MEM 1 => "001"
--                              MEM 2 => "010"
--                              MEM 3 => "011"
--                              MEM 4 => "100"
--                              MEM 5 => "101"
--                              MEM 6 => "110"
--                              MEM 7 => "111"
--                        [ref_Pe7 ref_Pe6 ... ref_Pe0]
--
-- ===========================================================================

entity addressTranslate is

  generic (
    ADDRESS_SIZE : positive := 12);

  port (
    indexBus   : in  std_logic_vector(95 downto 0);
    mult_nBus  : in  std_logic_vector(83 downto 0);
    mem_refBus : out std_logic_vector(23 downto 0);
    addressBus : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0));

end addressTranslate;


architecture behav_addressTranslate of addressTranslate is

  -- Auxiliar types:
  type array3 is array (7 downto 0) of std_logic_vector(2 downto 0);
  type array12 is array (7 downto 0) of std_logic_vector(11 downto 0);
  type arrayAux12 is array (6 downto 0) of std_logic_vector(11 downto 0);
  type arrayADX is array (7 downto 0) of std_logic_vector(ADDRESS_SIZE-1 downto 0);
  type arrayINTEGER is array (7 downto 0) of integer;

  -- Auxiliar signals to separate the bus into an array
  signal memRefArray : array3 := (others => (others => '0'));
  signal indexArray : array12 := (others => (others => '0'));
  signal multi_n : arrayAux12 := (others => (others => '0'));
  signal adxArray : arrayADX := (others => (others => '0'));
```

```
  -- Auxiliar signal to store the multiple of "n" that should be subtract from
  -- each "index" input
  signal correct_n : array12 := (others => (others => '0'));

  -- Auxiliar INTEGER signal to store the "address":
  signal adx_integer : arrayINTEGER;


begin  -- behav_addressTranslate

  -- Bus to Array
  arrayToBus_1: for i in 7 downto 0 generate
    indexArray(i) <= indexBus(12*(i+1)-1 downto 12*i);
    mem_refBus(3*(i+1)-1 downto 3*i) <= memRefArray(i);
    addressBus(ADDRESS_SIZE*(i+1)-1 downto ADDRESS_SIZE*i) <= adxArray(i);
  end generate arrayToBus_1;

  arrayToBus_2: for i in 6 downto 0 generate
    multi_n(i) <= mult_nBus(12*(i+1)-1 downto 12*i);
  end generate arrayToBus_2;


  -- -------------------------------------------------------------------------
  --        COMPARATORS FOR CHOOSE THE MEMORY AND CALCULATE THE ADDRESS
  --
  --               indexArray < n         -->     [MEM 0]
  --          n <= indexArray < 2n        -->     [MEM 1]
  --         2n <= indexArray < 3n        -->     [MEM 2]
  --         3n <= indexArray < 4n        -->     [MEM 3]
  --         4n <= indexArray < 5n        -->     [MEM 4]
  --         5n <= indexArray < 6n        -->     [MEM 5]
  --         6n <= indexArray < 7n        -->     [MEM 6]
  --         7n <= indexArray             -->     [MEM 7]

  comparators: for i in 7 downto 0 generate

    memRefArray(i) <= "000" when (indexArray(i) < multi_n(0)) else
                      "001" when (multi_n(0) <= indexArray(i) and indexArray(i) < multi_n(1)) else
                      "010" when (multi_n(1) <= indexArray(i) and indexArray(i) < multi_n(2)) else
                      "011" when (multi_n(2) <= indexArray(i) and indexArray(i) < multi_n(3)) else
                      "100" when (multi_n(3) <= indexArray(i) and indexArray(i) < multi_n(4)) else
                      "101" when (multi_n(4) <= indexArray(i) and indexArray(i) < multi_n(5)) else
                      "110" when (multi_n(5) <= indexArray(i) and indexArray(i) < multi_n(6)) else
                      "111" when (multi_n(6) <= indexArray(i)) else
                      (others => '0');

    -- Choosing the correct multiple of 'n'
    correct_n(i) <= multi_n(0) when (memRefArray(i) = "001") else
                    multi_n(1) when (memRefArray(i) = "010") else
                    multi_n(2) when (memRefArray(i) = "011") else
                    multi_n(3) when (memRefArray(i) = "100") else
                    multi_n(4) when (memRefArray(i) = "101") else
                    multi_n(5) when (memRefArray(i) = "110") else
                    multi_n(6) when (memRefArray(i) = "111") else
                    (others => '0');

    -- Calculating the ADDRESS
    adx_integer(i) <= conv_integer(indexArray(i)) - conv_integer(correct_n(i));
    adxArray(i) <= conv_std_logic_vector(adx_integer(i), ADDRESS_SIZE);

  end generate comparators;

end behav_addressTranslate;
```

IV. *addressGenerator.vhd*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

-- ---------------------------------------------------------------------------
--
--                       ADDRESS GENERATOR
--
--      Returns the references to memories for read/write and the address of
--    these, also tell us when a collision happens.
--      Has an input to select the nature of the output: order or scrambled.
--
--  INPUT:   clk        -> Clock signal. Rising edge synchronous.
--           enable     -> Enable synchronous signal. High activation.
--           clear      -> Clear synchronous signal. High activation.
--           rst_n      -> Reset asynchronous signal. Low activation.
--           scrambled  -> Select if the output is in order or scrambled:
--                               '0' => ORDER
--                               '1' => SCRAMBLED
--           N_in       -> Number of elements in memory
--           Pe_in      -> Parallel processes (1, 2, 4, 6 or 8)
--
--  OUTPUT:  validOut   -> High if the output is valid, low not valid.
--           mem_ref    -> Bus. Contains the references of each parallel
--                         process to each memory for read/write in it.
--                               Pe_0 <= mem_ref(2 downto 0);
--                               Pe_1 <= mem_ref(5 downto 3);
--                               Pe_2 <= mem_ref(8 downto 6);
--                               Pe_3 <= mem_ref(11 downto 9);
--                               Pe_4 <= mem_ref(14 downto 12);
--                               Pe_5 <= mem_ref(17 downto 15);
--                               Pe_6 <= mem_ref(20 downto 18);
--                               Pe_7 <= mem_ref(23 downto 21);
--           address    -> Bus. Contains the address of the memories that is
--                         references by "mem_ref". [ 8*ADDRESS_SIZE bits ]
--
-- ---------------------------------------------------------------------------


entity addressGenerator is

  generic (
    ADDRESS_SIZE : positive := 15);

  port (
    clk       : in  std_logic;
    enable    : in  std_logic;
    clear     : in  std_logic;
    rst_n     : in  std_logic;
    scrambled : in  std_logic;
    N_in      : in  std_logic_vector(11 downto 0);
    Pe_in     : in  std_logic_vector(3 downto 0);
    validOut  : out std_logic;
    mem_ref   : out std_logic_vector(23 downto 0);
    address   : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0));

end addressGenerator;


architecture behav_addressGenerator of addressGenerator is
```

```
  -- -------------------------------------------------------------------------
  --              COMPONENTS: indexGenerator & addressTranslate

  component indexGenerator
    port (
      clk       : in  std_logic;
      enable    : in  std_logic;
      clear     : in  std_logic;
      rst_n     : in  std_logic;
      scrambled : in  std_logic;
      N_in      : in  std_logic_vector(11 downto 0);
      Pe_in     : in  std_logic_vector(3 downto 0);
      validOut  : out std_logic;
      indexOut  : out std_logic_vector(95 downto 0);
      mult_nBus : out std_logic_vector(83 downto 0));
  end component;

  component addressTranslate
    generic (
      ADDRESS_SIZE : positive);
    port (
    indexBus    : in  std_logic_vector(95 downto 0);
    mult_nBus   : in  std_logic_vector(83 downto 0);
    mem_refBus  : out std_logic_vector(23 downto 0);
    addressBus  : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0));
  end component;

  -- Auxiliar signals to connect "addressTranslate" and "indexGenerator"
  signal index : std_logic_vector(95 downto 0) := (others => '0');
  signal n_Bus : std_logic_vector(83 downto 0) := (others => '0');


begin  -- behav_addressGenerator

  -- Creation of the components and conexion
  indxGen : indexGenerator
    port map (
      clk       => clk,
      enable    => enable,
      clear     => clear,
      rst_n     => rst_n,
      scrambled => scrambled,
      N_in      => N_in,
      Pe_in     => Pe_in,
      validOut  => validOut,
      indexOut  => index,
      mult_nBus => n_Bus);

  addTrans : addressTranslate
    generic map (
      ADDRESS_SIZE => ADDRESS_SIZE)
    port map (
      indexBus    => index,
      mult_nBus   => n_Bus,
      mem_refBus  => mem_ref,
      addressBus  => address);


end behav_addressGenerator;
```

V. *LIFOmultiple.vhd*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


-- ============================================================================
--                  LIFO MULTIPLE
--
--      Has two LIFOs, and with two external signals we can change the
--    write/read operation.
--
--   INPUTS:
--      clk           -> clock
--      clear         -> clear signal (synchronous)
--      rst_n         -> reset signal (asynchronous)
--      write_en      -> write enable
--      read_en       -> read enable
--      LIFO_change   -> change of LIFO write/read
--      data_in       -> data input
--
--   OUTPUTS:
--      data_out      -> data output (when "read_en" is set to '1')
--
-- ============================================================================


entity LIFOmultiple is

  generic (
    ADDRESS_SIZE : positive := 12);

  port (
    clk         : in  std_logic;
    clear       : in  std_logic;
    rst_n       : in  std_logic;
    write_en    : in  std_logic;
    read_en     : in  std_logic;
    LIFO_change : in  std_logic;
    data_in     : in  std_logic_vector(8*ADDRESS_SIZE+23 downto 0);
    data_out    : out std_logic_vector(8*ADDRESS_SIZE+23 downto 0));

end LIFOmultiple;


architecture behav_LIFOmultiple of LIFOmultiple is

  -- LIFO's with 64 positions. We need 6 bits to drive the memory
  type memType is array (63 downto 0) of std_logic_vector(8*ADDRESS_SIZE+23 downto 0);
  signal memory_0 : memType := (others => (others => '0'));
  signal memory_1 : memType := (others => (others => '0'));
  signal pointer_0 : std_logic_vector(5 downto 0) := (others => '0');
  signal pointer_1 : std_logic_vector(5 downto 0) := (others => '0');
  signal emptyLIFO : std_logic_vector(1 downto 0) := "00";  -- emptyLIFO(0) -> LIFO 0
                                                            -- emptyLIFO(1) -> LIFO 1
  signal LIFOpointer : std_logic := '0';   -- '0' -> LIFO 0
                                           -- '1' -> LIFO 1

begin  -- behav_LIFOmultiple


  -- Verifying if LIFOs empty, if empty -> '1'
```

```
emptyLIFO(0) <= '1' when (pointer_0 = "000000") else
                '0';
emptyLIFO(1) <= '1' when (pointer_1 = "000000") else
                '0';

sync: process (clk, rst_n)

  variable pntLIFO : std_logic := '0';
  variable pnt_0 : integer := 0;
  variable pnt_1 : integer := 0;

begin  -- process sync

  if (rst_n = '0') then
    memory_0 <= (others => (others => '0'));
    memory_1 <= (others => (others => '0'));
    pointer_0 <= (others => '0');
    pointer_1 <= (others => '0');
    LIFOpointer <= '0';
    data_out <= (others => '0');

  elsif (clk = '1' and clk'event) then
    if (clear = '1') then
      memory_0 <= (others => (others => '0'));
      memory_1 <= (others => (others => '0'));
      pointer_0 <= (others => '0');
      pointer_1 <= (others => '0');
      LIFOpointer <= '0';
      data_out <= (others => '0');

    else

      -- Signals to variables:
      pntLIFO := LIFOpointer;
      pnt_0 := conv_integer(pointer_0);
      pnt_1 := conv_integer(pointer_1);


      -- LIFO change:
      if (LIFO_change = '1' or (read_en = '1' and emptyLIFO(conv_integer(pntLIFO)) = '1')) then
        pntLIFO := not(pntLIFO);
      end if;


      -- ---------------------------------------------------------------------
      -- LIFO OPERATION:       READ -> pntLIFO
      --                       WRITE -> NOT pntLIFO

      if (read_en = '1') then
        if (pntLIFO = '0') then      -- read from memory_0
          if (pnt_0 > 0) then
            pnt_0 := pnt_0 - 1;
            data_out <= memory_0(pnt_0);
          end if;
        else                         -- read from memory_1
          if (pnt_1 > 0) then
            pnt_1 := pnt_1 - 1;
            data_out <= memory_1(pnt_1);
          end if;
        end if;
      end if;

      if (write_en = '1') then
        if (pntLIFO = '0') then      -- write from memory_1
          if (64 > pnt_1) then
```

```
            memory_1(pnt_1) <= data_in;
            pnt_1 := pnt_1 + 1;
          end if;
        else                        -- write from memory_0
          if (64 > pnt_0) then
            memory_0(pnt_0) <= data_in;
            pnt_0 := pnt_0 + 1;
          end if;
        end if;
      end if;
      -- -------------------------------------------------------------------


      -- Variables to signals:
      LIFOpointer <= pntLIFO;
      pointer_0 <= conv_std_logic_vector(pnt_0,6);
      pointer_1 <= conv_std_logic_vector(pnt_1,6);
      -- --------------------------------------------------------------------



    end if;  -- CLEAR
  end if;  -- CLK & RST_N


 end process sync;


end behav_LIFOmultiple;
```

VI. *switchMatrix.vhd*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity switchMatrix is

  generic (
    DSIZE : integer := 12);           -- # bit llr

  port (
    data_in0   : in  std_logic_vector(DSIZE-1 downto 0);
    data_in1   : in  std_logic_vector(DSIZE-1 downto 0);
    data_in2   : in  std_logic_vector(DSIZE-1 downto 0);
    data_in3   : in  std_logic_vector(DSIZE-1 downto 0);
    data_in4   : in  std_logic_vector(DSIZE-1 downto 0);
    data_in5   : in  std_logic_vector(DSIZE-1 downto 0);
    data_in6   : in  std_logic_vector(DSIZE-1 downto 0);
    data_in7   : in  std_logic_vector(DSIZE-1 downto 0);
    adx0       : in  std_logic_vector(2 downto 0);
    adx1       : in  std_logic_vector(2 downto 0);
    adx2       : in  std_logic_vector(2 downto 0);
    adx3       : in  std_logic_vector(2 downto 0);
    adx4       : in  std_logic_vector(2 downto 0);
    adx5       : in  std_logic_vector(2 downto 0);
    adx6       : in  std_logic_vector(2 downto 0);
    adx7       : in  std_logic_vector(2 downto 0);
    enable0    : in  std_logic;
    enable1    : in  std_logic;
    enable2    : in  std_logic;
    enable3    : in  std_logic;
    enable4    : in  std_logic;
    enable5    : in  std_logic;
    enable6    : in  std_logic;
    enable7    : in  std_logic;
    validOut0  : out std_logic;
    validOut1  : out std_logic;
    validOut2  : out std_logic;
    validOut3  : out std_logic;
    validOut4  : out std_logic;
    validOut5  : out std_logic;
    validOut6  : out std_logic;
    validOut7  : out std_logic;
    data_out0  : out std_logic_vector(DSIZE-1 downto 0);
    data_out1  : out std_logic_vector(DSIZE-1 downto 0);
    data_out2  : out std_logic_vector(DSIZE-1 downto 0);
    data_out3  : out std_logic_vector(DSIZE-1 downto 0);
    data_out4  : out std_logic_vector(DSIZE-1 downto 0);
    data_out5  : out std_logic_vector(DSIZE-1 downto 0);
    data_out6  : out std_logic_vector(DSIZE-1 downto 0);
    data_out7  : out std_logic_vector(DSIZE-1 downto 0));

end switchMatrix;


architecture behav of switchMatrix is

  signal sel_mux0  : std_logic_vector(2 downto 0);  -- selector mux 0
  signal sel_mux1  : std_logic_vector(2 downto 0);  -- selector mux 1
  signal sel_mux2  : std_logic_vector(2 downto 0);  -- selector mux 2
  signal sel_mux3  : std_logic_vector(2 downto 0);  -- selector mux 3
  signal sel_mux4  : std_logic_vector(2 downto 0);  -- selector mux 4
```

```
signal sel_mux5  : std_logic_vector(2 downto 0);  -- selector mux 5
signal sel_mux6  : std_logic_vector(2 downto 0);  -- selector mux 6
signal sel_mux7  : std_logic_vector(2 downto 0);  -- selector mux 7
signal sel_mux0_0 : std_logic_vector(2 downto 0);
signal sel_mux0_1 : std_logic_vector(2 downto 0);
signal sel_mux0_2 : std_logic_vector(2 downto 0);
signal sel_mux0_3 : std_logic_vector(2 downto 0);
signal sel_mux0_4 : std_logic_vector(2 downto 0);
signal sel_mux0_5 : std_logic_vector(2 downto 0);
signal sel_mux0_6 : std_logic_vector(2 downto 0);
signal sel_mux0_7 : std_logic_vector(2 downto 0);
signal sel_mux1_0 : std_logic_vector(2 downto 0);
signal sel_mux1_1 : std_logic_vector(2 downto 0);
signal sel_mux1_2 : std_logic_vector(2 downto 0);
signal sel_mux1_3 : std_logic_vector(2 downto 0);
signal sel_mux1_4 : std_logic_vector(2 downto 0);
signal sel_mux1_5 : std_logic_vector(2 downto 0);
signal sel_mux1_6 : std_logic_vector(2 downto 0);
signal sel_mux1_7 : std_logic_vector(2 downto 0);
signal sel_mux2_0 : std_logic_vector(2 downto 0);
signal sel_mux2_1 : std_logic_vector(2 downto 0);
signal sel_mux2_2 : std_logic_vector(2 downto 0);
signal sel_mux2_3 : std_logic_vector(2 downto 0);
signal sel_mux2_4 : std_logic_vector(2 downto 0);
signal sel_mux2_5 : std_logic_vector(2 downto 0);
signal sel_mux2_6 : std_logic_vector(2 downto 0);
signal sel_mux2_7 : std_logic_vector(2 downto 0);
signal sel_mux3_0 : std_logic_vector(2 downto 0);
signal sel_mux3_1 : std_logic_vector(2 downto 0);
signal sel_mux3_2 : std_logic_vector(2 downto 0);
signal sel_mux3_3 : std_logic_vector(2 downto 0);
signal sel_mux3_4 : std_logic_vector(2 downto 0);
signal sel_mux3_5 : std_logic_vector(2 downto 0);
signal sel_mux3_6 : std_logic_vector(2 downto 0);
signal sel_mux3_7 : std_logic_vector(2 downto 0);
signal sel_mux4_0 : std_logic_vector(2 downto 0);
signal sel_mux4_1 : std_logic_vector(2 downto 0);
signal sel_mux4_2 : std_logic_vector(2 downto 0);
signal sel_mux4_3 : std_logic_vector(2 downto 0);
signal sel_mux4_4 : std_logic_vector(2 downto 0);
signal sel_mux4_5 : std_logic_vector(2 downto 0);
signal sel_mux4_6 : std_logic_vector(2 downto 0);
signal sel_mux4_7 : std_logic_vector(2 downto 0);
signal sel_mux5_0 : std_logic_vector(2 downto 0);
signal sel_mux5_1 : std_logic_vector(2 downto 0);
signal sel_mux5_2 : std_logic_vector(2 downto 0);
signal sel_mux5_3 : std_logic_vector(2 downto 0);
signal sel_mux5_4 : std_logic_vector(2 downto 0);
signal sel_mux5_5 : std_logic_vector(2 downto 0);
signal sel_mux5_6 : std_logic_vector(2 downto 0);
signal sel_mux5_7 : std_logic_vector(2 downto 0);
signal sel_mux6_0 : std_logic_vector(2 downto 0);
signal sel_mux6_1 : std_logic_vector(2 downto 0);
signal sel_mux6_2 : std_logic_vector(2 downto 0);
signal sel_mux6_3 : std_logic_vector(2 downto 0);
signal sel_mux6_4 : std_logic_vector(2 downto 0);
signal sel_mux6_5 : std_logic_vector(2 downto 0);
signal sel_mux6_6 : std_logic_vector(2 downto 0);
signal sel_mux6_7 : std_logic_vector(2 downto 0);
signal sel_mux7_0 : std_logic_vector(2 downto 0);
signal sel_mux7_1 : std_logic_vector(2 downto 0);
signal sel_mux7_2 : std_logic_vector(2 downto 0);
signal sel_mux7_3 : std_logic_vector(2 downto 0);
signal sel_mux7_4 : std_logic_vector(2 downto 0);
signal sel_mux7_5 : std_logic_vector(2 downto 0);
```

```
  signal sel_mux7_6 : std_logic_vector(2 downto 0);
  signal sel_mux7_7 : std_logic_vector(2 downto 0);
  signal validOut_0 : std_logic_vector(7 downto 0);
  signal validOut_1 : std_logic_vector(7 downto 0);
  signal validOut_2 : std_logic_vector(7 downto 0);
  signal validOut_3 : std_logic_vector(7 downto 0);
  signal validOut_4 : std_logic_vector(7 downto 0);
  signal validOut_5 : std_logic_vector(7 downto 0);
  signal validOut_6 : std_logic_vector(7 downto 0);
  signal validOut_7 : std_logic_vector(7 downto 0);
  signal validOut : std_logic_vector(7 downto 0);

begin

  validOut0 <= validOut(0);
  validOut1 <= validOut(1);
  validOut2 <= validOut(2);
  validOut3 <= validOut(3);
  validOut4 <= validOut(4);
  validOut5 <= validOut(5);
  validOut6 <= validOut(6);
  validOut7 <= validOut(7);

  mux0: process (sel_mux0, data_in0, data_in1, data_in2, data_in3,
                 data_in4, data_in5, data_in6, data_in7)
  begin  -- process mux0
    case sel_mux0 is
      when "000" => data_out0 <= data_in0;
      when "001" => data_out0 <= data_in1;
      when "010" => data_out0 <= data_in2;
      when "011" => data_out0 <= data_in3;
      when "100" => data_out0 <= data_in4;
      when "101" => data_out0 <= data_in5;
      when "110" => data_out0 <= data_in6;
      when "111" => data_out0 <= data_in7;
      when others => data_out0 <= data_in0;
    end case;
  end process mux0;

  mux1: process (sel_mux1, data_in0, data_in1, data_in2, data_in3,
                 data_in4, data_in5, data_in6, data_in7)
  begin  -- process mux1
    case sel_mux1 is
      when "000" => data_out1 <= data_in0;
      when "001" => data_out1 <= data_in1;
      when "010" => data_out1 <= data_in2;
      when "011" => data_out1 <= data_in3;
      when "100" => data_out1 <= data_in4;
      when "101" => data_out1 <= data_in5;
      when "110" => data_out1 <= data_in6;
      when "111" => data_out1 <= data_in7;
      when others => data_out1 <= data_in1;
    end case;
  end process mux1;

  mux2: process (sel_mux2, data_in0, data_in1, data_in2, data_in3,
                 data_in4, data_in5, data_in6, data_in7)
  begin  -- process mux2
    case sel_mux2 is
      when "000" => data_out2 <= data_in0;
      when "001" => data_out2 <= data_in1;
      when "010" => data_out2 <= data_in2;
      when "011" => data_out2 <= data_in3;
      when "100" => data_out2 <= data_in4;
      when "101" => data_out2 <= data_in5;
```

```
    when "110" => data_out2 <= data_in6;
    when "111" => data_out2 <= data_in7;
    when others => data_out2 <= data_in2;
  end case;
end process mux2;

mux3: process (sel_mux3, data_in0, data_in1, data_in2, data_in3,
               data_in4, data_in5, data_in6, data_in7)
begin  -- process mux3
  case sel_mux3 is
    when "000" => data_out3 <= data_in0;
    when "001" => data_out3 <= data_in1;
    when "010" => data_out3 <= data_in2;
    when "011" => data_out3 <= data_in3;
    when "100" => data_out3 <= data_in4;
    when "101" => data_out3 <= data_in5;
    when "110" => data_out3 <= data_in6;
    when "111" => data_out3 <= data_in7;
    when others => data_out3 <= data_in3;
  end case;
end process mux3;

mux4: process (sel_mux4, data_in0, data_in1, data_in2, data_in3,
               data_in4, data_in5, data_in6, data_in7)
begin  -- process mux4
  case sel_mux4 is
    when "000" => data_out4 <= data_in0;
    when "001" => data_out4 <= data_in1;
    when "010" => data_out4 <= data_in2;
    when "011" => data_out4 <= data_in3;
    when "100" => data_out4 <= data_in4;
    when "101" => data_out4 <= data_in5;
    when "110" => data_out4 <= data_in6;
    when "111" => data_out4 <= data_in7;
    when others => data_out4 <= data_in4;
  end case;
end process mux4;

mux5: process (sel_mux5, data_in0, data_in1, data_in2, data_in3,
               data_in4, data_in5, data_in6, data_in7)
begin  -- process mux5
  case sel_mux5 is
    when "000" => data_out5 <= data_in0;
    when "001" => data_out5 <= data_in1;
    when "010" => data_out5 <= data_in2;
    when "011" => data_out5 <= data_in3;
    when "100" => data_out5 <= data_in4;
    when "101" => data_out5 <= data_in5;
    when "110" => data_out5 <= data_in6;
    when "111" => data_out5 <= data_in7;
    when others => data_out5 <= data_in5;
  end case;
end process mux5;

mux6: process (sel_mux6, data_in0, data_in1, data_in2, data_in3,
               data_in4, data_in5, data_in6, data_in7)
begin  -- process mux6
  case sel_mux6 is
    when "000" => data_out6 <= data_in0;
    when "001" => data_out6 <= data_in1;
    when "010" => data_out6 <= data_in2;
    when "011" => data_out6 <= data_in3;
    when "100" => data_out6 <= data_in4;
    when "101" => data_out6 <= data_in5;
    when "110" => data_out6 <= data_in6;
```

```
    when "111" => data_out6 <= data_in7;
    when others => data_out6 <= data_in6;
  end case;
end process mux6;

mux7: process (sel_mux7, data_in0, data_in1, data_in2, data_in3,
               data_in4, data_in5, data_in6, data_in7)
begin  -- process mux7
  case sel_mux7 is
    when "000" => data_out7 <= data_in0;
    when "001" => data_out7 <= data_in1;
    when "010" => data_out7 <= data_in2;
    when "011" => data_out7 <= data_in3;
    when "100" => data_out7 <= data_in4;
    when "101" => data_out7 <= data_in5;
    when "110" => data_out7 <= data_in6;
    when "111" => data_out7 <= data_in7;
    when others => data_out7 <= data_in7;
  end case;
end process mux7;

process(adx0, enable0)
begin  -- process
  if (enable0 = '1') then
    case adx0 is
      when "000" =>
        validOut_0 <= "00000001";
        sel_mux0_0 <= conv_std_logic_vector(0,3);
        sel_mux0_1 <= conv_std_logic_vector(0,3);
        sel_mux0_2 <= conv_std_logic_vector(0,3);
        sel_mux0_3 <= conv_std_logic_vector(0,3);
        sel_mux0_4 <= conv_std_logic_vector(0,3);
        sel_mux0_5 <= conv_std_logic_vector(0,3);
        sel_mux0_6 <= conv_std_logic_vector(0,3);
        sel_mux0_7 <= conv_std_logic_vector(0,3);
      when "001" =>
        validOut_0 <= "00000010";
        sel_mux0_0 <= conv_std_logic_vector(0,3);
        sel_mux0_1 <= conv_std_logic_vector(0,3);
        sel_mux0_2 <= conv_std_logic_vector(0,3);
        sel_mux0_3 <= conv_std_logic_vector(0,3);
        sel_mux0_4 <= conv_std_logic_vector(0,3);
        sel_mux0_5 <= conv_std_logic_vector(0,3);
        sel_mux0_6 <= conv_std_logic_vector(0,3);
        sel_mux0_7 <= conv_std_logic_vector(0,3);
      when "010" =>
        validOut_0 <= "00000100";
        sel_mux0_0 <= conv_std_logic_vector(0,3);
        sel_mux0_1 <= conv_std_logic_vector(0,3);
        sel_mux0_2 <= conv_std_logic_vector(0,3);
        sel_mux0_3 <= conv_std_logic_vector(0,3);
        sel_mux0_4 <= conv_std_logic_vector(0,3);
        sel_mux0_5 <= conv_std_logic_vector(0,3);
        sel_mux0_6 <= conv_std_logic_vector(0,3);
        sel_mux0_7 <= conv_std_logic_vector(0,3);
      when "011" =>
        validOut_0 <= "00001000";
        sel_mux0_0 <= conv_std_logic_vector(0,3);
        sel_mux0_1 <= conv_std_logic_vector(0,3);
        sel_mux0_2 <= conv_std_logic_vector(0,3);
        sel_mux0_3 <= conv_std_logic_vector(0,3);
        sel_mux0_4 <= conv_std_logic_vector(0,3);
        sel_mux0_5 <= conv_std_logic_vector(0,3);
        sel_mux0_6 <= conv_std_logic_vector(0,3);
        sel_mux0_7 <= conv_std_logic_vector(0,3);
```

```
      when "100" =>
        validOut_0 <= "00010000";
        sel_mux0_0 <= conv_std_logic_vector(0,3);
        sel_mux0_1 <= conv_std_logic_vector(0,3);
        sel_mux0_2 <= conv_std_logic_vector(0,3);
        sel_mux0_3 <= conv_std_logic_vector(0,3);
        sel_mux0_4 <= conv_std_logic_vector(0,3);
        sel_mux0_5 <= conv_std_logic_vector(0,3);
        sel_mux0_6 <= conv_std_logic_vector(0,3);
        sel_mux0_7 <= conv_std_logic_vector(0,3);
      when "101" =>
        validOut_0 <= "00100000";
        sel_mux0_0 <= conv_std_logic_vector(0,3);
        sel_mux0_1 <= conv_std_logic_vector(0,3);
        sel_mux0_2 <= conv_std_logic_vector(0,3);
        sel_mux0_3 <= conv_std_logic_vector(0,3);
        sel_mux0_4 <= conv_std_logic_vector(0,3);
        sel_mux0_5 <= conv_std_logic_vector(0,3);
        sel_mux0_6 <= conv_std_logic_vector(0,3);
        sel_mux0_7 <= conv_std_logic_vector(0,3);
      when "110" =>
        validOut_0 <= "01000000";
        sel_mux0_0 <= conv_std_logic_vector(0,3);
        sel_mux0_1 <= conv_std_logic_vector(0,3);
        sel_mux0_2 <= conv_std_logic_vector(0,3);
        sel_mux0_3 <= conv_std_logic_vector(0,3);
        sel_mux0_4 <= conv_std_logic_vector(0,3);
        sel_mux0_5 <= conv_std_logic_vector(0,3);
        sel_mux0_6 <= conv_std_logic_vector(0,3);
        sel_mux0_7 <= conv_std_logic_vector(0,3);
      when "111" =>
        validOut_0 <= "10000000";
        sel_mux0_0 <= conv_std_logic_vector(0,3);
        sel_mux0_1 <= conv_std_logic_vector(0,3);
        sel_mux0_2 <= conv_std_logic_vector(0,3);
        sel_mux0_3 <= conv_std_logic_vector(0,3);
        sel_mux0_4 <= conv_std_logic_vector(0,3);
        sel_mux0_5 <= conv_std_logic_vector(0,3);
        sel_mux0_6 <= conv_std_logic_vector(0,3);
        sel_mux0_7 <= conv_std_logic_vector(0,3);
      when others =>
        validOut_0 <= (others => '0');
        sel_mux0_0 <= conv_std_logic_vector(0,3);
        sel_mux0_1 <= conv_std_logic_vector(0,3);
        sel_mux0_2 <= conv_std_logic_vector(0,3);
        sel_mux0_3 <= conv_std_logic_vector(0,3);
        sel_mux0_4 <= conv_std_logic_vector(0,3);
        sel_mux0_5 <= conv_std_logic_vector(0,3);
        sel_mux0_6 <= conv_std_logic_vector(0,3);
        sel_mux0_7 <= conv_std_logic_vector(0,3);
    end case;

  else
    validOut_0 <= (others => '0');
    sel_mux0_0 <= conv_std_logic_vector(0,3);
    sel_mux0_1 <= conv_std_logic_vector(0,3);
    sel_mux0_2 <= conv_std_logic_vector(0,3);
    sel_mux0_3 <= conv_std_logic_vector(0,3);
    sel_mux0_4 <= conv_std_logic_vector(0,3);
    sel_mux0_5 <= conv_std_logic_vector(0,3);
    sel_mux0_6 <= conv_std_logic_vector(0,3);
    sel_mux0_7 <= conv_std_logic_vector(0,3);

  end if;
end process;
```

```
process(adx1, enable1)
begin  -- process
  if (enable1 = '1') then
    case adx1 is
      when "000" =>
        validOut_1 <= "00000001";
        sel_mux1_0 <= conv_std_logic_vector(1,3);
        sel_mux1_1 <= conv_std_logic_vector(0,3);
        sel_mux1_2 <= conv_std_logic_vector(0,3);
        sel_mux1_3 <= conv_std_logic_vector(0,3);
        sel_mux1_4 <= conv_std_logic_vector(0,3);
        sel_mux1_5 <= conv_std_logic_vector(0,3);
        sel_mux1_6 <= conv_std_logic_vector(0,3);
        sel_mux1_7 <= conv_std_logic_vector(0,3);
      when "001" =>
        validOut_1 <= "00000010";
        sel_mux1_0 <= conv_std_logic_vector(0,3);
        sel_mux1_1 <= conv_std_logic_vector(1,3);
        sel_mux1_2 <= conv_std_logic_vector(0,3);
        sel_mux1_3 <= conv_std_logic_vector(0,3);
        sel_mux1_4 <= conv_std_logic_vector(0,3);
        sel_mux1_5 <= conv_std_logic_vector(0,3);
        sel_mux1_6 <= conv_std_logic_vector(0,3);
        sel_mux1_7 <= conv_std_logic_vector(0,3);
      when "010" =>
        validOut_1 <= "00000100";
        sel_mux1_0 <= conv_std_logic_vector(0,3);
        sel_mux1_1 <= conv_std_logic_vector(0,3);
        sel_mux1_2 <= conv_std_logic_vector(1,3);
        sel_mux1_3 <= conv_std_logic_vector(0,3);
        sel_mux1_4 <= conv_std_logic_vector(0,3);
        sel_mux1_5 <= conv_std_logic_vector(0,3);
        sel_mux1_6 <= conv_std_logic_vector(0,3);
        sel_mux1_7 <= conv_std_logic_vector(0,3);
      when "011" =>
        validOut_1 <= "00001000";
        sel_mux1_0 <= conv_std_logic_vector(0,3);
        sel_mux1_1 <= conv_std_logic_vector(0,3);
        sel_mux1_2 <= conv_std_logic_vector(0,3);
        sel_mux1_3 <= conv_std_logic_vector(1,3);
        sel_mux1_4 <= conv_std_logic_vector(0,3);
        sel_mux1_5 <= conv_std_logic_vector(0,3);
        sel_mux1_6 <= conv_std_logic_vector(0,3);
        sel_mux1_7 <= conv_std_logic_vector(0,3);
      when "100" =>
        validOut_1 <= "00010000";
        sel_mux1_0 <= conv_std_logic_vector(0,3);
        sel_mux1_1 <= conv_std_logic_vector(0,3);
        sel_mux1_2 <= conv_std_logic_vector(0,3);
        sel_mux1_3 <= conv_std_logic_vector(0,3);
        sel_mux1_4 <= conv_std_logic_vector(1,3);
        sel_mux1_5 <= conv_std_logic_vector(0,3);
        sel_mux1_6 <= conv_std_logic_vector(0,3);
        sel_mux1_7 <= conv_std_logic_vector(0,3);
      when "101" =>
        validOut_1 <= "00100000";
        sel_mux1_0 <= conv_std_logic_vector(0,3);
        sel_mux1_1 <= conv_std_logic_vector(0,3);
        sel_mux1_2 <= conv_std_logic_vector(0,3);
        sel_mux1_3 <= conv_std_logic_vector(0,3);
        sel_mux1_4 <= conv_std_logic_vector(0,3);
        sel_mux1_5 <= conv_std_logic_vector(1,3);
        sel_mux1_6 <= conv_std_logic_vector(0,3);
        sel_mux1_7 <= conv_std_logic_vector(0,3);
```

```
        when "110" =>
          validOut_1 <= "01000000";
          sel_mux1_0 <= conv_std_logic_vector(0,3);
          sel_mux1_1 <= conv_std_logic_vector(0,3);
          sel_mux1_2 <= conv_std_logic_vector(0,3);
          sel_mux1_3 <= conv_std_logic_vector(0,3);
          sel_mux1_4 <= conv_std_logic_vector(0,3);
          sel_mux1_5 <= conv_std_logic_vector(0,3);
          sel_mux1_6 <= conv_std_logic_vector(1,3);
          sel_mux1_7 <= conv_std_logic_vector(0,3);
        when "111" =>
          validOut_1 <= "10000000";
          sel_mux1_0 <= conv_std_logic_vector(0,3);
          sel_mux1_1 <= conv_std_logic_vector(0,3);
          sel_mux1_2 <= conv_std_logic_vector(0,3);
          sel_mux1_3 <= conv_std_logic_vector(0,3);
          sel_mux1_4 <= conv_std_logic_vector(0,3);
          sel_mux1_5 <= conv_std_logic_vector(0,3);
          sel_mux1_6 <= conv_std_logic_vector(0,3);
          sel_mux1_7 <= conv_std_logic_vector(1,3);
        when others =>
          validOut_1 <= (others => '0');
          sel_mux1_0 <= conv_std_logic_vector(0,3);
          sel_mux1_1 <= conv_std_logic_vector(0,3);
          sel_mux1_2 <= conv_std_logic_vector(0,3);
          sel_mux1_3 <= conv_std_logic_vector(0,3);
          sel_mux1_4 <= conv_std_logic_vector(0,3);
          sel_mux1_5 <= conv_std_logic_vector(0,3);
          sel_mux1_6 <= conv_std_logic_vector(0,3);
          sel_mux1_7 <= conv_std_logic_vector(0,3);
      end case;

    else
      validOut_1 <= (others => '0');
      sel_mux1_0 <= conv_std_logic_vector(0,3);
      sel_mux1_1 <= conv_std_logic_vector(0,3);
      sel_mux1_2 <= conv_std_logic_vector(0,3);
      sel_mux1_3 <= conv_std_logic_vector(0,3);
      sel_mux1_4 <= conv_std_logic_vector(0,3);
      sel_mux1_5 <= conv_std_logic_vector(0,3);
      sel_mux1_6 <= conv_std_logic_vector(0,3);
      sel_mux1_7 <= conv_std_logic_vector(0,3);

    end if;

  end process;

  process(adx2, enable2)
  begin  -- process
    if (enable2 = '1') then
      case adx2 is
        when "000" =>
          validOut_2 <= "00000001";
          sel_mux2_0 <= conv_std_logic_vector(2,3);
          sel_mux2_1 <= conv_std_logic_vector(0,3);
          sel_mux2_2 <= conv_std_logic_vector(0,3);
          sel_mux2_3 <= conv_std_logic_vector(0,3);
          sel_mux2_4 <= conv_std_logic_vector(0,3);
          sel_mux2_5 <= conv_std_logic_vector(0,3);
          sel_mux2_6 <= conv_std_logic_vector(0,3);
          sel_mux2_7 <= conv_std_logic_vector(0,3);
        when "001" =>
          validOut_2 <= "00000010";
          sel_mux2_0 <= conv_std_logic_vector(0,3);
          sel_mux2_1 <= conv_std_logic_vector(2,3);
```

```
    sel_mux2_2 <= conv_std_logic_vector(0,3);
    sel_mux2_3 <= conv_std_logic_vector(0,3);
    sel_mux2_4 <= conv_std_logic_vector(0,3);
    sel_mux2_5 <= conv_std_logic_vector(0,3);
    sel_mux2_6 <= conv_std_logic_vector(0,3);
    sel_mux2_7 <= conv_std_logic_vector(0,3);
  when "010" =>
    validOut_2 <= "00000100";
    sel_mux2_0 <= conv_std_logic_vector(0,3);
    sel_mux2_1 <= conv_std_logic_vector(0,3);
    sel_mux2_2 <= conv_std_logic_vector(2,3);
    sel_mux2_3 <= conv_std_logic_vector(0,3);
    sel_mux2_4 <= conv_std_logic_vector(0,3);
    sel_mux2_5 <= conv_std_logic_vector(0,3);
    sel_mux2_6 <= conv_std_logic_vector(0,3);
    sel_mux2_7 <= conv_std_logic_vector(0,3);
  when "011" =>
    validOut_2 <= "00001000";
    sel_mux2_0 <= conv_std_logic_vector(0,3);
    sel_mux2_1 <= conv_std_logic_vector(0,3);
    sel_mux2_2 <= conv_std_logic_vector(0,3);
    sel_mux2_3 <= conv_std_logic_vector(2,3);
    sel_mux2_4 <= conv_std_logic_vector(0,3);
    sel_mux2_5 <= conv_std_logic_vector(0,3);
    sel_mux2_6 <= conv_std_logic_vector(0,3);
    sel_mux2_7 <= conv_std_logic_vector(0,3);
  when "100" =>
    validOut_2 <= "00010000";
    sel_mux2_0 <= conv_std_logic_vector(0,3);
    sel_mux2_1 <= conv_std_logic_vector(0,3);
    sel_mux2_2 <= conv_std_logic_vector(0,3);
    sel_mux2_3 <= conv_std_logic_vector(0,3);
    sel_mux2_4 <= conv_std_logic_vector(2,3);
    sel_mux2_5 <= conv_std_logic_vector(0,3);
    sel_mux2_6 <= conv_std_logic_vector(0,3);
    sel_mux2_7 <= conv_std_logic_vector(0,3);
  when "101" =>
    validOut_2 <= "00100000";
    sel_mux2_0 <= conv_std_logic_vector(0,3);
    sel_mux2_1 <= conv_std_logic_vector(0,3);
    sel_mux2_2 <= conv_std_logic_vector(0,3);
    sel_mux2_3 <= conv_std_logic_vector(0,3);
    sel_mux2_4 <= conv_std_logic_vector(0,3);
    sel_mux2_5 <= conv_std_logic_vector(2,3);
    sel_mux2_6 <= conv_std_logic_vector(0,3);
    sel_mux2_7 <= conv_std_logic_vector(0,3);
  when "110" =>
    validOut_2 <= "01000000";
    sel_mux2_0 <= conv_std_logic_vector(0,3);
    sel_mux2_1 <= conv_std_logic_vector(0,3);
    sel_mux2_2 <= conv_std_logic_vector(0,3);
    sel_mux2_3 <= conv_std_logic_vector(0,3);
    sel_mux2_4 <= conv_std_logic_vector(0,3);
    sel_mux2_5 <= conv_std_logic_vector(0,3);
    sel_mux2_6 <= conv_std_logic_vector(2,3);
    sel_mux2_7 <= conv_std_logic_vector(0,3);
  when "111" =>
    validOut_2 <= "10000000";
    sel_mux2_0 <= conv_std_logic_vector(0,3);
    sel_mux2_1 <= conv_std_logic_vector(0,3);
    sel_mux2_2 <= conv_std_logic_vector(0,3);
    sel_mux2_3 <= conv_std_logic_vector(0,3);
    sel_mux2_4 <= conv_std_logic_vector(0,3);
    sel_mux2_5 <= conv_std_logic_vector(0,3);
    sel_mux2_6 <= conv_std_logic_vector(0,3);
```

```
        sel_mux2_7 <= conv_std_logic_vector(2,3);
      when others =>
        validOut_2 <= (others => '0');
        sel_mux2_0 <= conv_std_logic_vector(0,3);
        sel_mux2_1 <= conv_std_logic_vector(0,3);
        sel_mux2_2 <= conv_std_logic_vector(0,3);
        sel_mux2_3 <= conv_std_logic_vector(0,3);
        sel_mux2_4 <= conv_std_logic_vector(0,3);
        sel_mux2_5 <= conv_std_logic_vector(0,3);
        sel_mux2_6 <= conv_std_logic_vector(0,3);
        sel_mux2_7 <= conv_std_logic_vector(0,3);
    end case;

  else
    validOut_2 <= (others => '0');
    sel_mux2_0 <= conv_std_logic_vector(0,3);
    sel_mux2_1 <= conv_std_logic_vector(0,3);
    sel_mux2_2 <= conv_std_logic_vector(0,3);
    sel_mux2_3 <= conv_std_logic_vector(0,3);
    sel_mux2_4 <= conv_std_logic_vector(0,3);
    sel_mux2_5 <= conv_std_logic_vector(0,3);
    sel_mux2_6 <= conv_std_logic_vector(0,3);
    sel_mux2_7 <= conv_std_logic_vector(0,3);

  end if;
end process;

process(adx3, enable3)
begin  -- process
  if (enable3 = '1') then
    case adx3 is
      when "000" =>
        validOut_3 <= "00000001";
        sel_mux3_0 <= conv_std_logic_vector(3,3);
        sel_mux3_1 <= conv_std_logic_vector(0,3);
        sel_mux3_2 <= conv_std_logic_vector(0,3);
        sel_mux3_3 <= conv_std_logic_vector(0,3);
        sel_mux3_4 <= conv_std_logic_vector(0,3);
        sel_mux3_5 <= conv_std_logic_vector(0,3);
        sel_mux3_6 <= conv_std_logic_vector(0,3);
        sel_mux3_7 <= conv_std_logic_vector(0,3);
      when "001" =>
        validOut_3 <= "00000010";
        sel_mux3_0 <= conv_std_logic_vector(0,3);
        sel_mux3_1 <= conv_std_logic_vector(3,3);
        sel_mux3_2 <= conv_std_logic_vector(0,3);
        sel_mux3_3 <= conv_std_logic_vector(0,3);
        sel_mux3_4 <= conv_std_logic_vector(0,3);
        sel_mux3_5 <= conv_std_logic_vector(0,3);
        sel_mux3_6 <= conv_std_logic_vector(0,3);
        sel_mux3_7 <= conv_std_logic_vector(0,3);
      when "010" =>
        validOut_3 <= "00000100";
        sel_mux3_0 <= conv_std_logic_vector(0,3);
        sel_mux3_1 <= conv_std_logic_vector(0,3);
        sel_mux3_2 <= conv_std_logic_vector(3,3);
        sel_mux3_3 <= conv_std_logic_vector(0,3);
        sel_mux3_4 <= conv_std_logic_vector(0,3);
        sel_mux3_5 <= conv_std_logic_vector(0,3);
        sel_mux3_6 <= conv_std_logic_vector(0,3);
        sel_mux3_7 <= conv_std_logic_vector(0,3);
      when "011" =>
        validOut_3 <= "00001000";
        sel_mux3_0 <= conv_std_logic_vector(0,3);
        sel_mux3_1 <= conv_std_logic_vector(0,3);
```

```
      sel_mux3_2 <= conv_std_logic_vector(0,3);
      sel_mux3_3 <= conv_std_logic_vector(3,3);
      sel_mux3_4 <= conv_std_logic_vector(0,3);
      sel_mux3_5 <= conv_std_logic_vector(0,3);
      sel_mux3_6 <= conv_std_logic_vector(0,3);
      sel_mux3_7 <= conv_std_logic_vector(0,3);
    when "100" =>
      validOut_3 <= "00010000";
      sel_mux3_0 <= conv_std_logic_vector(0,3);
      sel_mux3_1 <= conv_std_logic_vector(0,3);
      sel_mux3_2 <= conv_std_logic_vector(0,3);
      sel_mux3_3 <= conv_std_logic_vector(0,3);
      sel_mux3_4 <= conv_std_logic_vector(3,3);
      sel_mux3_5 <= conv_std_logic_vector(0,3);
      sel_mux3_6 <= conv_std_logic_vector(0,3);
      sel_mux3_7 <= conv_std_logic_vector(0,3);
    when "101" =>
      validOut_3 <= "00100000";
      sel_mux3_0 <= conv_std_logic_vector(0,3);
      sel_mux3_1 <= conv_std_logic_vector(0,3);
      sel_mux3_2 <= conv_std_logic_vector(0,3);
      sel_mux3_3 <= conv_std_logic_vector(0,3);
      sel_mux3_4 <= conv_std_logic_vector(0,3);
      sel_mux3_5 <= conv_std_logic_vector(3,3);
      sel_mux3_6 <= conv_std_logic_vector(0,3);
      sel_mux3_7 <= conv_std_logic_vector(0,3);
    when "110" =>
      validOut_3 <= "01000000";
      sel_mux3_0 <= conv_std_logic_vector(0,3);
      sel_mux3_1 <= conv_std_logic_vector(0,3);
      sel_mux3_2 <= conv_std_logic_vector(0,3);
      sel_mux3_3 <= conv_std_logic_vector(0,3);
      sel_mux3_4 <= conv_std_logic_vector(0,3);
      sel_mux3_5 <= conv_std_logic_vector(0,3);
      sel_mux3_6 <= conv_std_logic_vector(3,3);
      sel_mux3_7 <= conv_std_logic_vector(0,3);
    when "111" =>
      validOut_3 <= "10000000";
      sel_mux3_0 <= conv_std_logic_vector(0,3);
      sel_mux3_1 <= conv_std_logic_vector(0,3);
      sel_mux3_2 <= conv_std_logic_vector(0,3);
      sel_mux3_3 <= conv_std_logic_vector(0,3);
      sel_mux3_4 <= conv_std_logic_vector(0,3);
      sel_mux3_5 <= conv_std_logic_vector(0,3);
      sel_mux3_6 <= conv_std_logic_vector(0,3);
      sel_mux3_7 <= conv_std_logic_vector(3,3);
    when others =>
      validOut_3 <= (others => '0');
      sel_mux3_0 <= conv_std_logic_vector(0,3);
      sel_mux3_1 <= conv_std_logic_vector(0,3);
      sel_mux3_2 <= conv_std_logic_vector(0,3);
      sel_mux3_3 <= conv_std_logic_vector(0,3);
      sel_mux3_4 <= conv_std_logic_vector(0,3);
      sel_mux3_5 <= conv_std_logic_vector(0,3);
      sel_mux3_6 <= conv_std_logic_vector(0,3);
      sel_mux3_7 <= conv_std_logic_vector(0,3);
  end case;

else
  validOut_3 <= (others => '0');
  sel_mux3_0 <= conv_std_logic_vector(0,3);
  sel_mux3_1 <= conv_std_logic_vector(0,3);
  sel_mux3_2 <= conv_std_logic_vector(0,3);
  sel_mux3_3 <= conv_std_logic_vector(0,3);
  sel_mux3_4 <= conv_std_logic_vector(0,3);
```

```
    sel_mux3_5 <= conv_std_logic_vector(0,3);
    sel_mux3_6 <= conv_std_logic_vector(0,3);
    sel_mux3_7 <= conv_std_logic_vector(0,3);

  end if;
end process;

process(adx4, enable4)
begin  -- process
  if (enable4 = '1') then
    case adx4 is
      when "000" =>
        validOut_4 <= "00000001";
        sel_mux4_0 <= conv_std_logic_vector(4,3);
        sel_mux4_1 <= conv_std_logic_vector(0,3);
        sel_mux4_2 <= conv_std_logic_vector(0,3);
        sel_mux4_3 <= conv_std_logic_vector(0,3);
        sel_mux4_4 <= conv_std_logic_vector(0,3);
        sel_mux4_5 <= conv_std_logic_vector(0,3);
        sel_mux4_6 <= conv_std_logic_vector(0,3);
        sel_mux4_7 <= conv_std_logic_vector(0,3);
      when "001" =>
        validOut_4 <= "00000010";
        sel_mux4_0 <= conv_std_logic_vector(0,3);
        sel_mux4_1 <= conv_std_logic_vector(4,3);
        sel_mux4_2 <= conv_std_logic_vector(0,3);
        sel_mux4_3 <= conv_std_logic_vector(0,3);
        sel_mux4_4 <= conv_std_logic_vector(0,3);
        sel_mux4_5 <= conv_std_logic_vector(0,3);
        sel_mux4_6 <= conv_std_logic_vector(0,3);
        sel_mux4_7 <= conv_std_logic_vector(0,3);
      when "010" =>
        validOut_4 <= "00000100";
        sel_mux4_0 <= conv_std_logic_vector(0,3);
        sel_mux4_1 <= conv_std_logic_vector(0,3);
        sel_mux4_2 <= conv_std_logic_vector(4,3);
        sel_mux4_3 <= conv_std_logic_vector(0,3);
        sel_mux4_4 <= conv_std_logic_vector(0,3);
        sel_mux4_5 <= conv_std_logic_vector(0,3);
        sel_mux4_6 <= conv_std_logic_vector(0,3);
        sel_mux4_7 <= conv_std_logic_vector(0,3);
      when "011" =>
        validOut_4 <= "00001000";
        sel_mux4_0 <= conv_std_logic_vector(0,3);
        sel_mux4_1 <= conv_std_logic_vector(0,3);
        sel_mux4_2 <= conv_std_logic_vector(0,3);
        sel_mux4_3 <= conv_std_logic_vector(4,3);
        sel_mux4_4 <= conv_std_logic_vector(0,3);
        sel_mux4_5 <= conv_std_logic_vector(0,3);
        sel_mux4_6 <= conv_std_logic_vector(0,3);
        sel_mux4_7 <= conv_std_logic_vector(0,3);
      when "100" =>
        validOut_4 <= "00010000";
        sel_mux4_0 <= conv_std_logic_vector(0,3);
        sel_mux4_1 <= conv_std_logic_vector(0,3);
        sel_mux4_2 <= conv_std_logic_vector(0,3);
        sel_mux4_3 <= conv_std_logic_vector(0,3);
        sel_mux4_4 <= conv_std_logic_vector(4,3);
        sel_mux4_5 <= conv_std_logic_vector(0,3);
        sel_mux4_6 <= conv_std_logic_vector(0,3);
        sel_mux4_7 <= conv_std_logic_vector(0,3);
      when "101" =>
        validOut_4 <= "00100000";
        sel_mux4_0 <= conv_std_logic_vector(0,3);
        sel_mux4_1 <= conv_std_logic_vector(0,3);
```

```
        sel_mux4_2 <= conv_std_logic_vector(0,3);
        sel_mux4_3 <= conv_std_logic_vector(0,3);
        sel_mux4_4 <= conv_std_logic_vector(0,3);
        sel_mux4_5 <= conv_std_logic_vector(4,3);
        sel_mux4_6 <= conv_std_logic_vector(0,3);
        sel_mux4_7 <= conv_std_logic_vector(0,3);
      when "110" =>
        validOut_4 <= "01000000";
        sel_mux4_0 <= conv_std_logic_vector(0,3);
        sel_mux4_1 <= conv_std_logic_vector(0,3);
        sel_mux4_2 <= conv_std_logic_vector(0,3);
        sel_mux4_3 <= conv_std_logic_vector(0,3);
        sel_mux4_4 <= conv_std_logic_vector(0,3);
        sel_mux4_5 <= conv_std_logic_vector(0,3);
        sel_mux4_6 <= conv_std_logic_vector(4,3);
        sel_mux4_7 <= conv_std_logic_vector(0,3);
      when "111" =>
        validOut_4 <= "10000000";
        sel_mux4_0 <= conv_std_logic_vector(0,3);
        sel_mux4_1 <= conv_std_logic_vector(0,3);
        sel_mux4_2 <= conv_std_logic_vector(0,3);
        sel_mux4_3 <= conv_std_logic_vector(0,3);
        sel_mux4_4 <= conv_std_logic_vector(0,3);
        sel_mux4_5 <= conv_std_logic_vector(0,3);
        sel_mux4_6 <= conv_std_logic_vector(0,3);
        sel_mux4_7 <= conv_std_logic_vector(4,3);
      when others =>
        validOut_4 <= (others => '0');
        sel_mux4_0 <= conv_std_logic_vector(0,3);
        sel_mux4_1 <= conv_std_logic_vector(0,3);
        sel_mux4_2 <= conv_std_logic_vector(0,3);
        sel_mux4_3 <= conv_std_logic_vector(0,3);
        sel_mux4_4 <= conv_std_logic_vector(0,3);
        sel_mux4_5 <= conv_std_logic_vector(0,3);
        sel_mux4_6 <= conv_std_logic_vector(0,3);
        sel_mux4_7 <= conv_std_logic_vector(0,3);
    end case;

  else
    validOut_4 <= (others => '0');
    sel_mux4_0 <= conv_std_logic_vector(0,3);
    sel_mux4_1 <= conv_std_logic_vector(0,3);
    sel_mux4_2 <= conv_std_logic_vector(0,3);
    sel_mux4_3 <= conv_std_logic_vector(0,3);
    sel_mux4_4 <= conv_std_logic_vector(0,3);
    sel_mux4_5 <= conv_std_logic_vector(0,3);
    sel_mux4_6 <= conv_std_logic_vector(0,3);
    sel_mux4_7 <= conv_std_logic_vector(0,3);

  end if;
end process;

process(adx5, enable5)
begin  -- process
  if (enable5 = '1') then
    case adx5 is
      when "000" =>
        validOut_5 <= "00000001";
        sel_mux5_0 <= conv_std_logic_vector(5,3);
        sel_mux5_1 <= conv_std_logic_vector(0,3);
        sel_mux5_2 <= conv_std_logic_vector(0,3);
        sel_mux5_3 <= conv_std_logic_vector(0,3);
        sel_mux5_4 <= conv_std_logic_vector(0,3);
        sel_mux5_5 <= conv_std_logic_vector(0,3);
        sel_mux5_6 <= conv_std_logic_vector(0,3);
```

```
    sel_mux5_7 <= conv_std_logic_vector(0,3);
  when "001" =>
    validOut_5 <= "00000010";
    sel_mux5_0 <= conv_std_logic_vector(0,3);
    sel_mux5_1 <= conv_std_logic_vector(5,3);
    sel_mux5_2 <= conv_std_logic_vector(0,3);
    sel_mux5_3 <= conv_std_logic_vector(0,3);
    sel_mux5_4 <= conv_std_logic_vector(0,3);
    sel_mux5_5 <= conv_std_logic_vector(0,3);
    sel_mux5_6 <= conv_std_logic_vector(0,3);
    sel_mux5_7 <= conv_std_logic_vector(0,3);
  when "010" =>
    validOut_5 <= "00000100";
    sel_mux5_0 <= conv_std_logic_vector(0,3);
    sel_mux5_1 <= conv_std_logic_vector(0,3);
    sel_mux5_2 <= conv_std_logic_vector(5,3);
    sel_mux5_3 <= conv_std_logic_vector(0,3);
    sel_mux5_4 <= conv_std_logic_vector(0,3);
    sel_mux5_5 <= conv_std_logic_vector(0,3);
    sel_mux5_6 <= conv_std_logic_vector(0,3);
    sel_mux5_7 <= conv_std_logic_vector(0,3);
  when "011" =>
    validOut_5 <= "00001000";
    sel_mux5_0 <= conv_std_logic_vector(0,3);
    sel_mux5_1 <= conv_std_logic_vector(0,3);
    sel_mux5_2 <= conv_std_logic_vector(0,3);
    sel_mux5_3 <= conv_std_logic_vector(5,3);
    sel_mux5_4 <= conv_std_logic_vector(0,3);
    sel_mux5_5 <= conv_std_logic_vector(0,3);
    sel_mux5_6 <= conv_std_logic_vector(0,3);
    sel_mux5_7 <= conv_std_logic_vector(0,3);
  when "100" =>
    validOut_5 <= "00010000";
    sel_mux5_0 <= conv_std_logic_vector(0,3);
    sel_mux5_1 <= conv_std_logic_vector(0,3);
    sel_mux5_2 <= conv_std_logic_vector(0,3);
    sel_mux5_3 <= conv_std_logic_vector(0,3);
    sel_mux5_4 <= conv_std_logic_vector(5,3);
    sel_mux5_5 <= conv_std_logic_vector(0,3);
    sel_mux5_6 <= conv_std_logic_vector(0,3);
    sel_mux5_7 <= conv_std_logic_vector(0,3);
  when "101" =>
    validOut_5 <= "00100000";
    sel_mux5_0 <= conv_std_logic_vector(0,3);
    sel_mux5_1 <= conv_std_logic_vector(0,3);
    sel_mux5_2 <= conv_std_logic_vector(0,3);
    sel_mux5_3 <= conv_std_logic_vector(0,3);
    sel_mux5_4 <= conv_std_logic_vector(0,3);
    sel_mux5_5 <= conv_std_logic_vector(5,3);
    sel_mux5_6 <= conv_std_logic_vector(0,3);
    sel_mux5_7 <= conv_std_logic_vector(0,3);
  when "110" =>
    validOut_5 <= "01000000";
    sel_mux5_0 <= conv_std_logic_vector(0,3);
    sel_mux5_1 <= conv_std_logic_vector(0,3);
    sel_mux5_2 <= conv_std_logic_vector(0,3);
    sel_mux5_3 <= conv_std_logic_vector(0,3);
    sel_mux5_4 <= conv_std_logic_vector(0,3);
    sel_mux5_5 <= conv_std_logic_vector(0,3);
    sel_mux5_6 <= conv_std_logic_vector(5,3);
    sel_mux5_7 <= conv_std_logic_vector(0,3);
  when "111" =>
    validOut_5 <= "10000000";
    sel_mux5_0 <= conv_std_logic_vector(0,3);
    sel_mux5_1 <= conv_std_logic_vector(0,3);
```

```
        sel_mux5_2 <= conv_std_logic_vector(0,3);
        sel_mux5_3 <= conv_std_logic_vector(0,3);
        sel_mux5_4 <= conv_std_logic_vector(0,3);
        sel_mux5_5 <= conv_std_logic_vector(0,3);
        sel_mux5_6 <= conv_std_logic_vector(0,3);
        sel_mux5_7 <= conv_std_logic_vector(5,3);
      when others =>
        validOut_5 <= (others => '0');
        sel_mux5_0 <= conv_std_logic_vector(0,3);
        sel_mux5_1 <= conv_std_logic_vector(0,3);
        sel_mux5_2 <= conv_std_logic_vector(0,3);
        sel_mux5_3 <= conv_std_logic_vector(0,3);
        sel_mux5_4 <= conv_std_logic_vector(0,3);
        sel_mux5_5 <= conv_std_logic_vector(0,3);
        sel_mux5_6 <= conv_std_logic_vector(0,3);
        sel_mux5_7 <= conv_std_logic_vector(0,3);
    end case;

  else
    validOut_5 <= (others => '0');
    sel_mux5_0 <= conv_std_logic_vector(0,3);
    sel_mux5_1 <= conv_std_logic_vector(0,3);
    sel_mux5_2 <= conv_std_logic_vector(0,3);
    sel_mux5_3 <= conv_std_logic_vector(0,3);
    sel_mux5_4 <= conv_std_logic_vector(0,3);
    sel_mux5_5 <= conv_std_logic_vector(0,3);
    sel_mux5_6 <= conv_std_logic_vector(0,3);
    sel_mux5_7 <= conv_std_logic_vector(0,3);

  end if;
end process;

process(adx6, enable6)
begin  -- process
  if (enable6 = '1') then
    case adx6 is
      when "000" =>
        validOut_6 <= "00000001";
        sel_mux6_0 <= conv_std_logic_vector(6,3);
        sel_mux6_1 <= conv_std_logic_vector(0,3);
        sel_mux6_2 <= conv_std_logic_vector(0,3);
        sel_mux6_3 <= conv_std_logic_vector(0,3);
        sel_mux6_4 <= conv_std_logic_vector(0,3);
        sel_mux6_5 <= conv_std_logic_vector(0,3);
        sel_mux6_6 <= conv_std_logic_vector(0,3);
        sel_mux6_7 <= conv_std_logic_vector(0,3);
      when "001" =>
        validOut_6 <= "00000010";
        sel_mux6_0 <= conv_std_logic_vector(0,3);
        sel_mux6_1 <= conv_std_logic_vector(6,3);
        sel_mux6_2 <= conv_std_logic_vector(0,3);
        sel_mux6_3 <= conv_std_logic_vector(0,3);
        sel_mux6_4 <= conv_std_logic_vector(0,3);
        sel_mux6_5 <= conv_std_logic_vector(0,3);
        sel_mux6_6 <= conv_std_logic_vector(0,3);
        sel_mux6_7 <= conv_std_logic_vector(0,3);
      when "010" =>
        validOut_6 <= "00000100";
        sel_mux6_0 <= conv_std_logic_vector(0,3);
        sel_mux6_1 <= conv_std_logic_vector(0,3);
        sel_mux6_2 <= conv_std_logic_vector(6,3);
        sel_mux6_3 <= conv_std_logic_vector(0,3);
        sel_mux6_4 <= conv_std_logic_vector(0,3);
        sel_mux6_5 <= conv_std_logic_vector(0,3);
        sel_mux6_6 <= conv_std_logic_vector(0,3);
```

```
        sel_mux6_7 <= conv_std_logic_vector(0,3);
      when "011" =>
        validOut_6 <= "00001000";
        sel_mux6_0 <= conv_std_logic_vector(0,3);
        sel_mux6_1 <= conv_std_logic_vector(0,3);
        sel_mux6_2 <= conv_std_logic_vector(0,3);
        sel_mux6_3 <= conv_std_logic_vector(6,3);
        sel_mux6_4 <= conv_std_logic_vector(0,3);
        sel_mux6_5 <= conv_std_logic_vector(0,3);
        sel_mux6_6 <= conv_std_logic_vector(0,3);
        sel_mux6_7 <= conv_std_logic_vector(0,3);
      when "100" =>
        validOut_6 <= "00010000";
        sel_mux6_0 <= conv_std_logic_vector(0,3);
        sel_mux6_1 <= conv_std_logic_vector(0,3);
        sel_mux6_2 <= conv_std_logic_vector(0,3);
        sel_mux6_3 <= conv_std_logic_vector(0,3);
        sel_mux6_4 <= conv_std_logic_vector(6,3);
        sel_mux6_5 <= conv_std_logic_vector(0,3);
        sel_mux6_6 <= conv_std_logic_vector(0,3);
        sel_mux6_7 <= conv_std_logic_vector(0,3);
      when "101" =>
        validOut_6 <= "00100000";
        sel_mux6_0 <= conv_std_logic_vector(0,3);
        sel_mux6_1 <= conv_std_logic_vector(0,3);
        sel_mux6_2 <= conv_std_logic_vector(0,3);
        sel_mux6_3 <= conv_std_logic_vector(0,3);
        sel_mux6_4 <= conv_std_logic_vector(0,3);
        sel_mux6_5 <= conv_std_logic_vector(6,3);
        sel_mux6_6 <= conv_std_logic_vector(0,3);
        sel_mux6_7 <= conv_std_logic_vector(0,3);
      when "110" =>
        validOut_6 <= "01000000";
        sel_mux6_0 <= conv_std_logic_vector(0,3);
        sel_mux6_1 <= conv_std_logic_vector(0,3);
        sel_mux6_2 <= conv_std_logic_vector(0,3);
        sel_mux6_3 <= conv_std_logic_vector(0,3);
        sel_mux6_4 <= conv_std_logic_vector(0,3);
        sel_mux6_5 <= conv_std_logic_vector(0,3);
        sel_mux6_6 <= conv_std_logic_vector(6,3);
        sel_mux6_7 <= conv_std_logic_vector(0,3);
      when "111" =>
        validOut_6 <= "10000000";
        sel_mux6_0 <= conv_std_logic_vector(0,3);
        sel_mux6_1 <= conv_std_logic_vector(0,3);
        sel_mux6_2 <= conv_std_logic_vector(0,3);
        sel_mux6_3 <= conv_std_logic_vector(0,3);
        sel_mux6_4 <= conv_std_logic_vector(0,3);
        sel_mux6_5 <= conv_std_logic_vector(0,3);
        sel_mux6_6 <= conv_std_logic_vector(0,3);
        sel_mux6_7 <= conv_std_logic_vector(6,3);
      when others =>
        validOut_6 <= (others => '0');
        sel_mux6_0 <= conv_std_logic_vector(0,3);
        sel_mux6_1 <= conv_std_logic_vector(0,3);
        sel_mux6_2 <= conv_std_logic_vector(0,3);
        sel_mux6_3 <= conv_std_logic_vector(0,3);
        sel_mux6_4 <= conv_std_logic_vector(0,3);
        sel_mux6_5 <= conv_std_logic_vector(0,3);
        sel_mux6_6 <= conv_std_logic_vector(0,3);
        sel_mux6_7 <= conv_std_logic_vector(0,3);
    end case;

  else
    validOut_6 <= (others => '0');
```

```
    sel_mux6_0 <= conv_std_logic_vector(0,3);
    sel_mux6_1 <= conv_std_logic_vector(0,3);
    sel_mux6_2 <= conv_std_logic_vector(0,3);
    sel_mux6_3 <= conv_std_logic_vector(0,3);
    sel_mux6_4 <= conv_std_logic_vector(0,3);
    sel_mux6_5 <= conv_std_logic_vector(0,3);
    sel_mux6_6 <= conv_std_logic_vector(0,3);
    sel_mux6_7 <= conv_std_logic_vector(0,3);

  end if;
end process;

process(adx7, enable7)
begin  -- process
  if (enable7 = '1') then
    case adx7 is
      when "000" =>
        validOut_7 <= "00000001";
        sel_mux7_0 <= conv_std_logic_vector(7,3);
        sel_mux7_1 <= conv_std_logic_vector(0,3);
        sel_mux7_2 <= conv_std_logic_vector(0,3);
        sel_mux7_3 <= conv_std_logic_vector(0,3);
        sel_mux7_4 <= conv_std_logic_vector(0,3);
        sel_mux7_5 <= conv_std_logic_vector(0,3);
        sel_mux7_6 <= conv_std_logic_vector(0,3);
        sel_mux7_7 <= conv_std_logic_vector(0,3);
      when "001" =>
        validOut_7 <= "00000010";
        sel_mux7_0 <= conv_std_logic_vector(0,3);
        sel_mux7_1 <= conv_std_logic_vector(7,3);
        sel_mux7_2 <= conv_std_logic_vector(0,3);
        sel_mux7_3 <= conv_std_logic_vector(0,3);
        sel_mux7_4 <= conv_std_logic_vector(0,3);
        sel_mux7_5 <= conv_std_logic_vector(0,3);
        sel_mux7_6 <= conv_std_logic_vector(0,3);
        sel_mux7_7 <= conv_std_logic_vector(0,3);
      when "010" =>
        validOut_7 <= "00000100";
        sel_mux7_0 <= conv_std_logic_vector(0,3);
        sel_mux7_1 <= conv_std_logic_vector(0,3);
        sel_mux7_2 <= conv_std_logic_vector(7,3);
        sel_mux7_3 <= conv_std_logic_vector(0,3);
        sel_mux7_4 <= conv_std_logic_vector(0,3);
        sel_mux7_5 <= conv_std_logic_vector(0,3);
        sel_mux7_6 <= conv_std_logic_vector(0,3);
        sel_mux7_7 <= conv_std_logic_vector(0,3);
      when "011" =>
        validOut_7 <= "00001000";
        sel_mux7_0 <= conv_std_logic_vector(0,3);
        sel_mux7_1 <= conv_std_logic_vector(0,3);
        sel_mux7_2 <= conv_std_logic_vector(0,3);
        sel_mux7_3 <= conv_std_logic_vector(7,3);
        sel_mux7_4 <= conv_std_logic_vector(0,3);
        sel_mux7_5 <= conv_std_logic_vector(0,3);
        sel_mux7_6 <= conv_std_logic_vector(0,3);
        sel_mux7_7 <= conv_std_logic_vector(0,3);
      when "100" =>
        validOut_7 <= "00010000";
        sel_mux7_0 <= conv_std_logic_vector(0,3);
        sel_mux7_1 <= conv_std_logic_vector(0,3);
        sel_mux7_2 <= conv_std_logic_vector(0,3);
        sel_mux7_3 <= conv_std_logic_vector(0,3);
        sel_mux7_4 <= conv_std_logic_vector(7,3);
        sel_mux7_5 <= conv_std_logic_vector(0,3);
        sel_mux7_6 <= conv_std_logic_vector(0,3);
```

```
        sel_mux7_7 <= conv_std_logic_vector(0,3);
      when "101" =>
        validOut_7 <= "00100000";
        sel_mux7_0 <= conv_std_logic_vector(0,3);
        sel_mux7_1 <= conv_std_logic_vector(0,3);
        sel_mux7_2 <= conv_std_logic_vector(0,3);
        sel_mux7_3 <= conv_std_logic_vector(0,3);
        sel_mux7_4 <= conv_std_logic_vector(0,3);
        sel_mux7_5 <= conv_std_logic_vector(7,3);
        sel_mux7_6 <= conv_std_logic_vector(0,3);
        sel_mux7_7 <= conv_std_logic_vector(0,3);
      when "110" =>
        validOut_7 <= "01000000";
        sel_mux7_0 <= conv_std_logic_vector(0,3);
        sel_mux7_1 <= conv_std_logic_vector(0,3);
        sel_mux7_2 <= conv_std_logic_vector(0,3);
        sel_mux7_3 <= conv_std_logic_vector(0,3);
        sel_mux7_4 <= conv_std_logic_vector(0,3);
        sel_mux7_5 <= conv_std_logic_vector(0,3);
        sel_mux7_6 <= conv_std_logic_vector(7,3);
        sel_mux7_7 <= conv_std_logic_vector(0,3);
      when "111" =>
        validOut_7 <= "10000000";
        sel_mux7_0 <= conv_std_logic_vector(0,3);
        sel_mux7_1 <= conv_std_logic_vector(0,3);
        sel_mux7_2 <= conv_std_logic_vector(0,3);
        sel_mux7_3 <= conv_std_logic_vector(0,3);
        sel_mux7_4 <= conv_std_logic_vector(0,3);
        sel_mux7_5 <= conv_std_logic_vector(0,3);
        sel_mux7_6 <= conv_std_logic_vector(0,3);
        sel_mux7_7 <= conv_std_logic_vector(7,3);
      when others =>
        validOut_7 <= (others => '0');
        sel_mux7_0 <= conv_std_logic_vector(0,3);
        sel_mux7_1 <= conv_std_logic_vector(0,3);
        sel_mux7_2 <= conv_std_logic_vector(0,3);
        sel_mux7_3 <= conv_std_logic_vector(0,3);
        sel_mux7_4 <= conv_std_logic_vector(0,3);
        sel_mux7_5 <= conv_std_logic_vector(0,3);
        sel_mux7_6 <= conv_std_logic_vector(0,3);
        sel_mux7_7 <= conv_std_logic_vector(0,3);
    end case;

  else
    validOut_7 <= (others => '0');
    sel_mux7_0 <= conv_std_logic_vector(0,3);
    sel_mux7_1 <= conv_std_logic_vector(0,3);
    sel_mux7_2 <= conv_std_logic_vector(0,3);
    sel_mux7_3 <= conv_std_logic_vector(0,3);
    sel_mux7_4 <= conv_std_logic_vector(0,3);
    sel_mux7_5 <= conv_std_logic_vector(0,3);
    sel_mux7_6 <= conv_std_logic_vector(0,3);
    sel_mux7_7 <= conv_std_logic_vector(0,3);

  end if;
end process;

process(sel_mux0_0, sel_mux1_0, sel_mux2_0, sel_mux3_0,
        sel_mux4_0, sel_mux5_0, sel_mux6_0, sel_mux7_0,
        validOut_0, validOut_1, validOut_2, validOut_3,
        validOut_4, validOut_5, validOut_6, validOut_7)
  variable tmp : std_logic_vector(2 downto 0);
  variable tmpBit : std_logic;
begin  -- process
  tmp := sel_mux0_0;
```

```
    tmp := tmp or sel_mux1_0;
    tmp := tmp or sel_mux2_0;
    tmp := tmp or sel_mux3_0;
    tmp := tmp or sel_mux4_0;
    tmp := tmp or sel_mux5_0;
    tmp := tmp or sel_mux6_0;
    tmp := tmp or sel_mux7_0;
    sel_mux0 <= tmp;

    tmpBit := validOut_0(0);
    tmpBit := tmpBit or validOut_1(0);
    tmpBit := tmpBit or validOut_2(0);
    tmpBit := tmpBit or validOut_3(0);
    tmpBit := tmpBit or validOut_4(0);
    tmpBit := tmpBit or validOut_5(0);
    tmpBit := tmpBit or validOut_6(0);
    tmpBit := tmpBit or validOut_7(0);
    validOut(0) <= tmpBit;
  end process;

  process(sel_mux0_1, sel_mux1_1, sel_mux2_1, sel_mux3_1,
          sel_mux4_1, sel_mux5_1, sel_mux6_1, sel_mux7_1,
          validOut_0, validOut_1, validOut_2, validOut_3,
          validOut_4, validOut_5, validOut_6, validOut_7)
    variable tmp : std_logic_vector(2 downto 0);
    variable tmpBit : std_logic;
  begin  -- process
    tmp := sel_mux0_1;
    tmp := tmp or sel_mux1_1;
    tmp := tmp or sel_mux2_1;
    tmp := tmp or sel_mux3_1;
    tmp := tmp or sel_mux4_1;
    tmp := tmp or sel_mux5_1;
    tmp := tmp or sel_mux6_1;
    tmp := tmp or sel_mux7_1;
    sel_mux1 <= tmp;

    tmpBit := validOut_0(1);
    tmpBit := tmpBit or validOut_1(1);
    tmpBit := tmpBit or validOut_2(1);
    tmpBit := tmpBit or validOut_3(1);
    tmpBit := tmpBit or validOut_4(1);
    tmpBit := tmpBit or validOut_5(1);
    tmpBit := tmpBit or validOut_6(1);
    tmpBit := tmpBit or validOut_7(1);
    validOut(1) <= tmpBit;
  end process;

  process(sel_mux0_2, sel_mux1_2, sel_mux2_2, sel_mux3_2,
          sel_mux4_2, sel_mux5_2, sel_mux6_2, sel_mux7_2,
          validOut_0, validOut_1, validOut_2, validOut_3,
          validOut_4, validOut_5, validOut_6, validOut_7)
    variable tmp : std_logic_vector(2 downto 0);
    variable tmpBit : std_logic;
  begin  -- process
    tmp := sel_mux0_2;
    tmp := tmp or sel_mux1_2;
    tmp := tmp or sel_mux2_2;
    tmp := tmp or sel_mux3_2;
    tmp := tmp or sel_mux4_2;
    tmp := tmp or sel_mux5_2;
    tmp := tmp or sel_mux6_2;
    tmp := tmp or sel_mux7_2;
    sel_mux2 <= tmp;
```

```
    tmpBit := validOut_0(2);
    tmpBit := tmpBit or validOut_1(2);
    tmpBit := tmpBit or validOut_2(2);
    tmpBit := tmpBit or validOut_3(2);
    tmpBit := tmpBit or validOut_4(2);
    tmpBit := tmpBit or validOut_5(2);
    tmpBit := tmpBit or validOut_6(2);
    tmpBit := tmpBit or validOut_7(2);
    validOut(2) <= tmpBit;
  end process;

  process(sel_mux0_3, sel_mux1_3, sel_mux2_3, sel_mux3_3,
          sel_mux4_3, sel_mux5_3, sel_mux6_3, sel_mux7_3,
          validOut_0, validOut_1, validOut_2, validOut_3,
          validOut_4, validOut_5, validOut_6, validOut_7)
    variable tmp : std_logic_vector(2 downto 0);
    variable tmpBit : std_logic;
  begin  -- process
    tmp := sel_mux0_3;
    tmp := tmp or sel_mux1_3;
    tmp := tmp or sel_mux2_3;
    tmp := tmp or sel_mux3_3;
    tmp := tmp or sel_mux4_3;
    tmp := tmp or sel_mux5_3;
    tmp := tmp or sel_mux6_3;
    tmp := tmp or sel_mux7_3;
    sel_mux3 <= tmp;

    tmpBit := validOut_0(3);
    tmpBit := tmpBit or validOut_1(3);
    tmpBit := tmpBit or validOut_2(3);
    tmpBit := tmpBit or validOut_3(3);
    tmpBit := tmpBit or validOut_4(3);
    tmpBit := tmpBit or validOut_5(3);
    tmpBit := tmpBit or validOut_6(3);
    tmpBit := tmpBit or validOut_7(3);
    validOut(3) <= tmpBit;
  end process;

  process(sel_mux0_4, sel_mux1_4, sel_mux2_4, sel_mux3_4,
          sel_mux4_4, sel_mux5_4, sel_mux6_4, sel_mux7_4,
          validOut_0, validOut_1, validOut_2, validOut_3,
          validOut_4, validOut_5, validOut_6, validOut_7)
    variable tmp : std_logic_vector(2 downto 0);
    variable tmpBit : std_logic;
  begin  -- process
    tmp := sel_mux0_4;
    tmp := tmp or sel_mux1_4;
    tmp := tmp or sel_mux2_4;
    tmp := tmp or sel_mux3_4;
    tmp := tmp or sel_mux4_4;
    tmp := tmp or sel_mux5_4;
    tmp := tmp or sel_mux6_4;
    tmp := tmp or sel_mux7_4;
    sel_mux4 <= tmp;

    tmpBit := validOut_0(4);
    tmpBit := tmpBit or validOut_1(4);
    tmpBit := tmpBit or validOut_2(4);
    tmpBit := tmpBit or validOut_3(4);
    tmpBit := tmpBit or validOut_4(4);
    tmpBit := tmpBit or validOut_5(4);
    tmpBit := tmpBit or validOut_6(4);
    tmpBit := tmpBit or validOut_7(4);
    validOut(4) <= tmpBit;
```

```
end process;

process(sel_mux0_5, sel_mux1_5, sel_mux2_5, sel_mux3_5,
        sel_mux4_5, sel_mux5_5, sel_mux6_5, sel_mux7_5,
        validOut_0, validOut_1, validOut_2, validOut_3,
        validOut_4, validOut_5, validOut_6, validOut_7)
  variable tmp : std_logic_vector(2 downto 0);
  variable tmpBit : std_logic;
begin  -- process
  tmp := sel_mux0_5;
  tmp := tmp or sel_mux1_5;
  tmp := tmp or sel_mux2_5;
  tmp := tmp or sel_mux3_5;
  tmp := tmp or sel_mux4_5;
  tmp := tmp or sel_mux5_5;
  tmp := tmp or sel_mux6_5;
  tmp := tmp or sel_mux7_5;
  sel_mux5 <= tmp;

  tmpBit := validOut_0(5);
  tmpBit := tmpBit or validOut_1(5);
  tmpBit := tmpBit or validOut_2(5);
  tmpBit := tmpBit or validOut_3(5);
  tmpBit := tmpBit or validOut_4(5);
  tmpBit := tmpBit or validOut_5(5);
  tmpBit := tmpBit or validOut_6(5);
  tmpBit := tmpBit or validOut_7(5);
  validOut(5) <= tmpBit;
end process;

process(sel_mux0_6, sel_mux1_6, sel_mux2_6, sel_mux3_6,
        sel_mux4_6, sel_mux5_6, sel_mux6_6, sel_mux7_6,
        validOut_0, validOut_1, validOut_2, validOut_3,
        validOut_4, validOut_5, validOut_6, validOut_7)
  variable tmp : std_logic_vector(2 downto 0);
  variable tmpBit : std_logic;
begin  -- process
  tmp := sel_mux0_6;
  tmp := tmp or sel_mux1_6;
  tmp := tmp or sel_mux2_6;
  tmp := tmp or sel_mux3_6;
  tmp := tmp or sel_mux4_6;
  tmp := tmp or sel_mux5_6;
  tmp := tmp or sel_mux6_6;
  tmp := tmp or sel_mux7_6;
  sel_mux6 <= tmp;

  tmpBit := validOut_0(6);
  tmpBit := tmpBit or validOut_1(6);
  tmpBit := tmpBit or validOut_2(6);
  tmpBit := tmpBit or validOut_3(6);
  tmpBit := tmpBit or validOut_4(6);
  tmpBit := tmpBit or validOut_5(6);
  tmpBit := tmpBit or validOut_6(6);
  tmpBit := tmpBit or validOut_7(6);
  validOut(6) <= tmpBit;
end process;

process(sel_mux0_7, sel_mux1_7, sel_mux2_7, sel_mux3_7,
        sel_mux4_7, sel_mux5_7, sel_mux6_7, sel_mux7_7,
        validOut_0, validOut_1, validOut_2, validOut_3,
        validOut_4, validOut_5, validOut_6, validOut_7)
  variable tmp : std_logic_vector(2 downto 0);
  variable tmpBit : std_logic;
begin  -- process
```

```
    tmp := sel_mux0_7;
    tmp := tmp or sel_mux1_7;
    tmp := tmp or sel_mux2_7;
    tmp := tmp or sel_mux3_7;
    tmp := tmp or sel_mux4_7;
    tmp := tmp or sel_mux5_7;
    tmp := tmp or sel_mux6_7;
    tmp := tmp or sel_mux7_7;
    sel_mux7 <= tmp;

    tmpBit := validOut_0(7);
    tmpBit := tmpBit or validOut_1(7);
    tmpBit := tmpBit or validOut_2(7);
    tmpBit := tmpBit or validOut_3(7);
    tmpBit := tmpBit or validOut_4(7);
    tmpBit := tmpBit or validOut_5(7);
    tmpBit := tmpBit or validOut_6(7);
    tmpBit := tmpBit or validOut_7(7);
    validOut(7) <= tmpBit;
  end process;

end behav;
```

VII. *switchReadMemory.vhd*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

-- ============================================================================
--                      SWITCH READ MEMORY
--
--      Gets the read data from memories and write them into the SISOs
--    according to interleaver sequence.
--
--
--  INPUTS:
--     Pe_in        -> number of parallels processes.
--     address_in   -> addresses to read for each parallel process, generated
--                     by the interleaver.
--     memRef_in    -> memories to read for each parallel process, generated
--                     by the interleaver.
--     input_valid  -> "address_in", "memRef_in" & "collision" are valids.
--     MEM_RDATA    -> data read from the memories.
--
--  OUTPUTS:
--     MEM_RADX     -> addresses to read in each memory.
--     MEM_REN      -> enable of each memory.
--     RDATA        -> data read and exchanged (according to interleaver).
--     RDATA_VALID  -> '1' when the RDATA is valid (all data at the same time).
--
-- ============================================================================

entity switchReadMemory is

  generic (
    DATA_SIZE    : positive := 32;
    ADDRESS_SIZE : positive := 12);

  port (
    clk          : in  std_logic;
    clear        : in  std_logic;
    rst_n        : in  std_logic;
    Pe_in        : in  std_logic_vector(3 downto 0);
    address_in   : in  std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
    memRef_in    : in  std_logic_vector(23 downto 0);
    input_valid  : in  std_logic;
    MEM_RDATA    : in  std_logic_vector(8*DATA_SIZE-1 downto 0);
    MEM_RADX     : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
    MEM_REN      : out std_logic_vector(7 downto 0);
    RDATA        : out std_logic_vector(8*DATA_SIZE-1 downto 0);
    RDATA_VALID  : out std_logic);

end switchReadMemory;

architecture behav_switchReadMemory of switchReadMemory is

  -- Auxiliar component: MATRIX SWITCH
  component switchMatrix
    generic (
      DSIZE : integer);
    port (
      data_in0  : in  std_logic_vector(DSIZE-1 downto 0);
      data_in1  : in  std_logic_vector(DSIZE-1 downto 0);
      data_in2  : in  std_logic_vector(DSIZE-1 downto 0);
      data_in3  : in  std_logic_vector(DSIZE-1 downto 0);
```

```
    data_in4  : in  std_logic_vector(DSIZE-1 downto 0);
    data_in5  : in  std_logic_vector(DSIZE-1 downto 0);
    data_in6  : in  std_logic_vector(DSIZE-1 downto 0);
    data_in7  : in  std_logic_vector(DSIZE-1 downto 0);
    adx0      : in  std_logic_vector(2 downto 0);
    adx1      : in  std_logic_vector(2 downto 0);
    adx2      : in  std_logic_vector(2 downto 0);
    adx3      : in  std_logic_vector(2 downto 0);
    adx4      : in  std_logic_vector(2 downto 0);
    adx5      : in  std_logic_vector(2 downto 0);
    adx6      : in  std_logic_vector(2 downto 0);
    adx7      : in  std_logic_vector(2 downto 0);
    enable0   : in  std_logic;
    enable1   : in  std_logic;
    enable2   : in  std_logic;
    enable3   : in  std_logic;
    enable4   : in  std_logic;
    enable5   : in  std_logic;
    enable6   : in  std_logic;
    enable7   : in  std_logic;
    validOut0 : out std_logic;
    validOut1 : out std_logic;
    validOut2 : out std_logic;
    validOut3 : out std_logic;
    validOut4 : out std_logic;
    validOut5 : out std_logic;
    validOut6 : out std_logic;
    validOut7 : out std_logic;
    data_out0 : out std_logic_vector(DSIZE-1 downto 0);
    data_out1 : out std_logic_vector(DSIZE-1 downto 0);
    data_out2 : out std_logic_vector(DSIZE-1 downto 0);
    data_out3 : out std_logic_vector(DSIZE-1 downto 0);
    data_out4 : out std_logic_vector(DSIZE-1 downto 0);
    data_out5 : out std_logic_vector(DSIZE-1 downto 0);
    data_out6 : out std_logic_vector(DSIZE-1 downto 0);
    data_out7 : out std_logic_vector(DSIZE-1 downto 0));
end component;

-- Auxiliar type & signal to the output MATRIX SWITCH
type arrayData is array (7 downto 0) of std_logic_vector(DATA_SIZE-1 downto 0);
signal dataOutSwitch : arrayData;


-- Auxiliar type & signals. Division of the input "memRef" & "address"
type arrayMemRef is array (7 downto 0) of std_logic_vector(2 downto 0);
type arrayAddress is array (7 downto 0) of std_logic_vector(ADDRESS_SIZE-1 downto 0);
signal memRef : arrayMemRef;
signal address : arrayAddress;


-- Auxiliar signal. Store the address of each memory to read
signal mem_address : arrayAddress := (others => (others => '0'));


-- Auxiliar signal to store the information during a collision, we must store
-- "MEM_REN" and "address" inputs for each memory:
--          MEM_REN --> 1 bits
--          address --> ADDRESS_SIZE bits
--
--              address               MEM_REN
--      ---------------------------------------
--      |                            |         |
--      ---------------------------------------
--  ADDRESS_SIZE                   1      0
--      MSB                              LSB
```

```
--
--
--        ------------------------------------------
--        |       ADX_MEM_0           | MEM_REN(0) |
--        ------------------------------------------
--        |       ADX_MEM_1           | MEM_REN(1) |
--        ------------------------------------------
--        |       ADX_MEM_2           | MEM_REN(2) |
--        ------------------------------------------
--        |       ADX_MEM_3           | MEM_REN(3) |
--        ------------------------------------------
--        |       ADX_MEM_4           | MEM_REN(4) |
--        ------------------------------------------
--        |       ADX_MEM_5           | MEM_REN(5) |
--        ------------------------------------------
--        |       ADX_MEM_6           | MEM_REN(6) |
--        ------------------------------------------
--        |       ADX_MEM_7           | MEM_REN(7) |
--        ------------------------------------------
--
  type arrayMEMORY is array (7 downto 0) of std_logic_vector(ADDRESS_SIZE downto 0);
  signal collMEM : arrayMEMORY := (others => (others => '0'));

  -- WE READ THE DATA FROM A EXTERNAL MEMORY, AND IT TAKES 1 CYCLE TO GIVE
  -- US THE INFORMATION REQUEST, SO WE MUST DELAY 1 CYCLE SIGNALS LIKE
  -- "ENABLE SWITCH" and "MEMORY REF":
  signal memRefSwitch : arrayMemRef := (others => (others => '0'));
  signal memRefSwitchDelay : arrayMemRef := (others => (others => '0'));
  signal memRefSwitchCollision : arrayMemRef := (others => (others => '0'));
  signal enableSwitch : std_logic_vector(7 downto 0) := (others => '0');
  signal enableSwitchDelay : std_logic_vector(7 downto 0) := (others => '0');

  -- EVERY OUTPUTS OF THE MATRIX SWITCH ARE NOT VALID, SO WE MUST READ ONLY
  -- THE SIGNALS THAT ARE VALID:
  signal validSwitch : std_logic_vector(7 downto 0);
  signal validOutput : std_logic := '0';
  signal validOutputDelay : std_logic := '0';

  -- IF THERE'S A READ MEMORY COLLISION WE MUST TO KNOW IT, SO WE USE A FLAG:
  signal FLAG_collision : std_logic := '0';


begin  -- behav_switchReadMemory


  -- ------------------------------------------------------------------------
  -- Creation of the SWITCH MATRIX
  switching :  switchMatrix
    generic map (
      DSIZE => DATA_SIZE)
    port map (
      data_in0  => MEM_RDATA(DATA_SIZE-1 downto 0),
      data_in1  => MEM_RDATA(2*DATA_SIZE-1 downto DATA_SIZE),
      data_in2  => MEM_RDATA(3*DATA_SIZE-1 downto 2*DATA_SIZE),
      data_in3  => MEM_RDATA(4*DATA_SIZE-1 downto 3*DATA_SIZE),
      data_in4  => MEM_RDATA(5*DATA_SIZE-1 downto 4*DATA_SIZE),
      data_in5  => MEM_RDATA(6*DATA_SIZE-1 downto 5*DATA_SIZE),
      data_in6  => MEM_RDATA(7*DATA_SIZE-1 downto 6*DATA_SIZE),
      data_in7  => MEM_RDATA(8*DATA_SIZE-1 downto 7*DATA_SIZE),
      adx0      => memRefSwitchDelay(0),
      adx1      => memRefSwitchDelay(1),
      adx2      => memRefSwitchDelay(2),
      adx3      => memRefSwitchDelay(3),
      adx4      => memRefSwitchDelay(4),
      adx5      => memRefSwitchDelay(5),
```

```
    adx6      => memRefSwitchDelay(6),
    adx7      => memRefSwitchDelay(7),
    enable0   => enableSwitchDelay(0),
    enable1   => enableSwitchDelay(1),
    enable2   => enableSwitchDelay(2),
    enable3   => enableSwitchDelay(3),
    enable4   => enableSwitchDelay(4),
    enable5   => enableSwitchDelay(5),
    enable6   => enableSwitchDelay(6),
    enable7   => enableSwitchDelay(7),
    validOut0 => validSwitch(0),
    validOut1 => validSwitch(1),
    validOut2 => validSwitch(2),
    validOut3 => validSwitch(3),
    validOut4 => validSwitch(4),
    validOut5 => validSwitch(5),
    validOut6 => validSwitch(6),
    validOut7 => validSwitch(7),
    data_out0 => dataOutSwitch(0),
    data_out1 => dataOutSwitch(1),
    data_out2 => dataOutSwitch(2),
    data_out3 => dataOutSwitch(3),
    data_out4 => dataOutSwitch(4),
    data_out5 => dataOutSwitch(5),
    data_out6 => dataOutSwitch(6),
    data_out7 => dataOutSwitch(7));
-- --------------------------------------------------------------------------


-- Bus input to array
busToArray: for i in 7 downto 0 generate
  memRef(i) <= memRef_in((3*i)+2 downto 3*i);
  address(i) <= address_in(ADDRESS_SIZE*(i+1)-1 downto ADDRESS_SIZE*i);
end generate busToArray;

-- Array to bus
arrayToBus: for i in 7 downto 0 generate
  MEM_RADX(ADDRESS_SIZE*(i+1)-1 downto ADDRESS_SIZE*i) <= mem_address(i);
end generate arrayToBus;


-- The unique process of the entity:
procesing: process (clk, rst_n)

  -- Internal variables to handle the switching
  variable var_1 : std_logic_vector(7 downto 0) := (others => '0');
  variable var_2 : std_logic_vector(7 downto 0) := (others => '0');
  variable varMR_1 : arrayMemRef := (others => (others => '0'));
  variable varMR_2 : arrayMemRef := (others => (others => '0'));
  variable flagCollision : std_logic := '0';
  variable indxMR : integer := 0;

begin  -- process procesing

  if (rst_n = '0') then
    -- Outputs to zero:
    mem_address <= (others => (others => '0'));
    MEM_REN <= (others => '0');
    RDATA <= (others => '0');
    RDATA_VALID <= '0';
    -- Reset internal signal:
    collMEM <= (others => (others => '0'));
    memRefSwitch <= (others => (others => '0'));
    enableSwitch <= (others => '0');
    memRefSwitchDelay <= (others => (others => '0'));
```

```
    enableSwitchDelay <= (others => '0');
    memRefSwitchCollision <= (others => (others => '0'));
    validOutput <= '0';
    validOutputDelay <= '0';
    FLAG_collision <= '0';
    -- Reset internal variables:
    varMR_1 := (others => (others => '0'));
    varMR_2 := (others => (others => '0'));

  elsif (clk = '1' and clk'event) then
    if (clear = '1') then
      -- Outputs to zero:
      mem_address <= (others => (others => '0'));
      MEM_REN <= (others => '0');
      RDATA <= (others => '0');
      RDATA_VALID <= '0';
      -- Reset internal signal:
      collMEM <= (others => (others => '0'));
      memRefSwitch <= (others => (others => '0'));
      enableSwitch <= (others => '0');
      memRefSwitchDelay <= (others => (others => '0'));
      enableSwitchDelay <= (others => '0');
      memRefSwitchCollision <= (others => (others => '0'));
      validOutput <= '0';
      validOutputDelay <= '0';
      FLAG_collision <= '0';
      -- Reset internal variables:
      varMR_1 := (others => (others => '0'));
      varMR_2 := (others => (others => '0'));

    else

      -- --------------------------------------------------------------------
      -- WHEN THE INPUT IS NOT VALID AND THERE IS NO COLLISION IN HANDLING
      -- IT'S LIKE A "CLEAR" BUT "SOFT"
      if (input_valid = '0' and FLAG_collision = '0') then
        -- Outputs to zero:
        MEM_REN <= (others => '0');
        RDATA_VALID <= '0';
        -- Reset some internal signals:
        collMEM <= (others => (others => '0'));
        memRefSwitch <= (others => (others => '0'));
        enableSwitch <= (others => '0');
        validOutput <= '0';
      end if;  -- INPUT_VALID = 0 and FLAG_COLLISION = 0
      -- --------------------------------------------------------------------

      -- --------------------------------------------------------------------
      -- WORK WHEN THE INPUT IS VALID AND NO PREVIOUS COLLISION
      if (input_valid = '1' and FLAG_collision = '0') then

        -- Filling variables with zeros
        var_1 := "00000000";
        var_2 := "00000000";
        flagCollision := '0';
        indxMR := 0;

        for i in 0 to 7 loop
          if (i < conv_integer(Pe_in)) then
            indxMR := conv_integer(memRef(i));

            if (var_1(indxMR) = '0') then
              var_1(indxMR) := '1';
              varMR_1(indxMR) := conv_std_logic_vector(i,3);
              -- Memory addressing:
```

```
        mem_address(indxMR) <= address(i);

      else
        -- collision!!!
        flagCollision := '1';
        var_2(indxMR) := '1';
        varMR_2(indxMR) := conv_std_logic_vector(i,3);
        -- Storing memory address:
        collMEM(indxMR)(ADDRESS_SIZE downto 1) <= address(i);
      end if;

    end if;
  end loop;  -- i

  -- "memRef" activation (collision and not collision):
  memRefSwitch <= varMR_1;           -- not collision
  memRefSwitchCollision <= varMR_2;  -- collision

  -- Memory enable for handling the collision on 2 cycles:
  MEM_REN <= var_1;
  enableSwitch <= var_1;
  for i in 0 to 7 loop
    collMEM(i)(0) <= var_2(i);
  end loop;  -- i

  if (flagCollision = '0') then  -- NO COLLISION
    -- Next data output will be valid
    validOutput <= '1';

  else                           -- COLLISION
    -- We indicate that it's a collision
    FLAG_collision <= '1';
    -- The next data output will be NOT valid
    validOutput <= '0';

  end if;  -- COLLISION

end if;  -- INPUT_VALID = 1 and FLAG_COLLISION = 0
-- -------------------------------------------------------------------

-- -------------------------------------------------------------------
-- WORK WHEN IS A PREVIOUS COLLISION
if (FLAG_collision = '1') then

  -- No more collision
  FLAG_collision <= '0';

  -- "memRef" allocation for a collision
  memRefSwitch <= memRefSwitchCollision;

  for i in 0 to 7 loop
    MEM_REN(i) <= collMEM(i)(0);
    enableSwitch(i) <= collMEM(i)(0);
    mem_address(i) <= collMEM(i)(ADDRESS_SIZE downto 1);
  end loop;  -- i

  -- Next data output will be valid
  validOutput <= '1';

end if;
-- -------------------------------------------------------------------

-- -------------------------------------------------------------------
-- PROPAGATION OF THE SIGNALS "enableSwitch" TO "enableSwitchDelay",
-- "memRefSwitch" TO "memRefSwitchDelay" AND "validOutput" TO
```

```
        -- "validOutputDelay"
        memRefSwitchDelay <= memRefSwitch;
        enableSwitchDelay <= enableSwitch;
        validOutputDelay <= validOutput;
        -- -------------------------------------------------------------------

        -- -------------------------------------------------------------------
        -- GOING AWAY DATA

        -- We only consider the results when the MATRIX SWITCH tell us that the
        -- data in its outputs are valid -> "validSwitch"
        for i in 0 to 7 loop
          if (validSwitch(i) = '1') then
            RDATA((i+1)*DATA_SIZE-1 downto DATA_SIZE*i) <= dataOutSwitch(i);
          end if;
        end loop;  -- i

        -- The output will be valid when:
        if (validOutputDelay = '1') then
          RDATA_VALID <= '1';
        else
          RDATA_VALID <= '0';
        end if;
        -- -------------------------------------------------------------------

      end if;  -- CLEAR

    end if;  -- RST_N & CLK

  end process procesing;


end behav_switchReadMemory;
```

VIII. *switchWriteMemory.vhd*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


-- ============================================================================
--                      SWITCH WRITE MEMORY
--
--      Gets the data process by the SISOs and write them into MEMORY according
--   to interleaver sequence.
--
--
--  INPUTS:
--     Pe_in        -> number of parallels processes.
--     addressing   -> address from interleaver operation but invested.
--     WDATA        -> data read from SISO
--     WDATA_VALID  -> '1' when the WDATA is valid (all data at the same time).
--
--  OUTPUTS:
--     MEM_WADX     -> addresses to write in each memory.
--     MEM_WDATA    -> data to write.
--     MEM_WEN      -> write enable of each memory.
--
-- ============================================================================


entity switchWriteMemory is

  generic (
    DATA_SIZE    : positive := 8;
    ADDRESS_SIZE : positive := 12;
    FIFO_LENGTH  : positive := 6);      -- MINIMUM FIFO SIZE: 6

  port (
    clk         : in  std_logic;
    clear       : in  std_logic;
    rst_n       : in  std_logic;
    Pe_in       : in  std_logic_vector(3 downto 0);
    addressing  : in  std_logic_vector(8*ADDRESS_SIZE+23 downto 0);
    WDATA       : in  std_logic_vector(8*DATA_SIZE-1 downto 0);
    WDATA_VALID : in  std_logic;
    MEM_WADX    : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
    MEM_WDATA   : out std_logic_vector(8*DATA_SIZE-1 downto 0);
    MEM_WEN     : out std_logic_vector(7 downto 0));

end switchWriteMemory;


architecture behav_switchWriteMemory of switchWriteMemory is


  -- Auxiliar type & signals. Division of the input "addressing" into two
  -- signals: "memRef" & "address"
  type arrayMemRef is array (7 downto 0) of std_logic_vector(2 downto 0);
  type arrayAddress is array (7 downto 0) of std_logic_vector(ADDRESS_SIZE-1 downto 0);
  signal memRef : arrayMemRef := (others => (others => '0'));
  signal address : arrayAddress := (others => (others => '0'));


  -- Auxiliar type & signal to DELAY 1 cycles the signals "WDATA" and
  -- "WDATA_VALID", because the LIFO has 1 cycle of delay for read/write
```

```
  --
  --  WDATA        --D-->  WDATA_store
  --  WDATA_VALID  --D-->  FLAG_WDATA
  --
  type arrayDATA is array (7 downto 0) of std_logic_vector(DATA_SIZE-1 downto 0);
  signal WDATA_store : arrayDATA := (others => (others => '0'));
  signal FLAG_WDATA : std_logic := '0';


  -- Auxiliar type & signal for FIFO:
  --
  --   <------------------ 8*(DATA_SIZE+ADDRESS_SIZE) ----------------->
  --
  --     MEM 7   MEM 6   MEM 5   MEM 4   MEM 3   MEM 2   MEM 1   MEM 0
  --   |       |       |       |       |       |       |       |       |
  --   | D + A |       |       |       |       |       |       |       | POS 0
  --   | - - - | - - - | - - - | - - - | - - - | - - - | - - - | - - - |
  --   |       |       |       |       |       |       |       |       |
  --   |       |       |       |       |       |       |       |       | POS 1
  --   | - - - | - - - | - - - | - - - | - - - | - - - | - - - | - - - |
  --   |       |       |       |       |       |       |       |       |
  --   |       |       |       |       |       |       |       |       | POS 2
  --   | - - - | - - - | - - - | - - - | - - - | - - - | - - - | - - - |
  --   |       |       |       |       |       |       |       |       |
  --   .       .       .       .       .       .       .       .       .
  --   .       .       .       .       .       .       .       .       .
  --   .       .       .       .       .       .       .       .       .
  --   |       |       |       |       |       |       |       |       |
  --   | - - - | - - - | - - - | - - - | - - - | - - - | - - - | - - - |
  --   |       |       |       |       |       |       |       |       |
  --   |   _   |   _   |   _   |   _   |   _   |   _   |   _   |   _   | POS n
  --      /|\     /|\     /|\     /|\     /|\     /|\     /|\     /|\
  --       |       |       |       |       |       |       |       |
  --       ----------------------------------------------------------
  --                                ||
  --      ____________________________/\____________________________
  --     /                                                          \
  --     <----------------- DATA_SIZE+ADDRESS_SIZE ----------------->
  --     ------------------------------------------------------------
  --     |            DATA             |           ADDRESS          |
  --     ------------------------------------------------------------
  --     /|\                        /|\ /|\                      /|\
  --      |                          |   |                        |
  --      |               ADDRESS_SIZE ADDRESS_SIZE-1             |
  -- (DATA_SIZE+ADDRESS_SIZE)-1                                   0
  --      [MSB]                                                 [LSB]
  --
  type arrayFIFO is array (FIFO_LENGTH-1 downto 0)
               of std_logic_vector(8*(DATA_SIZE+ADDRESS_SIZE)-1 downto 0);
  type arrayPOINTER is array (7 downto 0) of integer;
  signal FIFO : arrayFIFO := (others => (others => '0'));
  signal pointerFIFO : arrayPOINTER := (others => 0);

  -- Auxiliar signal "FIFOread" tell us if some FIFO is empty or not:
  --           '0' empty
  --           '1' with some element, so we can read it
  signal FIFOread : std_logic_vector(7 downto 0) := (others => '0');


begin  -- behav_switchWriteMemory

  -- Bus input to arrays
  busToArray: for i in 7 downto 0 generate
    memRef(i) <= addressing((3*i)+2 downto 3*i);
    address(i) <= addressing((i+1)*ADDRESS_SIZE+23 downto i*ADDRESS_SIZE+24);
```

```
end generate busToArray;


-- READING FIFO
readingFIFO: for i in 0 to 7 generate
  FIFOread(i) <= '1' when (pointerFIFO(i) > 0) else
                 '0';
end generate readingFIFO;


-- Synchronous process
sync: process (clk, rst_n)

  -- Internal variables to handle the FIFO
  type arrayFIFOaux is array (7 downto 0) of std_logic_vector((DATA_SIZE+ADDRESS_SIZE)-1 downto 0);
  variable FIFOaux_A : arrayFIFOaux := (others => (others => '0'));
  variable FIFOaux_B : arrayFIFOaux := (others => (others => '0'));
  variable pointer : arrayPOINTER := (others => 0);
  variable pointerAUX : arrayPOINTER := (others => 0);
  variable indx : integer := 0;
  variable data_adxTEMP : std_logic_vector((DATA_SIZE+ADDRESS_SIZE)-1 downto 0) := (others => '0');
  type arraySubFIFO is array (FIFO_LENGTH-2 downto 0)
                    of std_logic_vector((DATA_SIZE+ADDRESS_SIZE)-1 downto 0);
  variable subFIFO : arraySubFIFO := (others => (others => '0'));
  variable pointerINT : integer := 0;

begin  -- process sync

  if (rst_n = '0') then
    -- Outputs to zero:
    MEM_WEN <= (others => '0');
    MEM_WDATA <= (others => '0');
    MEM_WADX <= (others => '0');
    -- Reset internal signals:
    WDATA_store <= (others => (others => '0'));
    FLAG_WDATA <= '0';
    FIFO <= (others => (others => '0'));
    pointerFIFO <= (others => 0);


  elsif (clk = '1' and clk'event) then
    if (clear = '1') then
      -- Outputs to zero:
      MEM_WEN <= (others => '0');
      MEM_WDATA <= (others => '0');
      MEM_WADX <= (others => '0');
      -- Reset internal signals:
      WDATA_store <= (others => (others => '0'));
      FLAG_WDATA <= '0';
      FIFO <= (others => (others => '0'));
      pointerFIFO <= (others => 0);


    else

      -- "pointerFIFO" signal to variable "pointer":
      pointer := pointerFIFO;

      -- ----------------------------------------------------------------------
      -- IF THERE'S SOME ELEMENTS INTO THE FIFO, WE READ ONE OF EACH IN
      -- A CYCLE:
      if (FIFOread /= "00000000") then

        data_adxTEMP := (others => '0');
        subFIFO := (others => (others => '0'));
```

```
    for i in 7 downto 0 loop
      if (FIFOread(i) = '1') then   -- READING FIFO
        data_adxTEMP :=
          FIFO(0)((i+1)*(DATA_SIZE+ADDRESS_SIZE)-1 downto (DATA_SIZE+ADDRESS_SIZE)*i);

        -- Up one position the correspond FIFO:
        for k in FIFO_LENGTH-2 downto 0 loop
          subFIFO(k) := FIFO(k+1)((i+1)*(DATA_SIZE+ADDRESS_SIZE)-1 downto
                                  (DATA_SIZE+ADDRESS_SIZE)*i);
          FIFO(k)((i+1)*(DATA_SIZE+ADDRESS_SIZE)-1 downto
                  (DATA_SIZE+ADDRESS_SIZE)*i) <= subFIFO(k);
        end loop;  -- k

        -- Pointer - 1:
        pointer(i) := pointer(i) - 1;

        MEM_WEN(i) <= '1';
        MEM_WADX((i+1)*ADDRESS_SIZE-1 downto
                 ADDRESS_SIZE*i) <= data_adxTEMP(ADDRESS_SIZE-1 downto 0);
        MEM_WDATA((i+1)*DATA_SIZE-1 downto
                  DATA_SIZE*i) <= data_adxTEMP((DATA_SIZE+ADDRESS_SIZE)-1 downto
                                              ADDRESS_SIZE);
      else
        MEM_WEN(i) <= '0';
      end if;
    end loop;  -- i

  else
    MEM_WEN <= (others => '0');
  end if;
  -- --------------------------------------------------------------------

  -- --------------------------------------------------------------------
  -- WHEN THE "FLAG_WDATA" IS '1' WE MUST TO PUT "WDATA_store" INTO ITS
  -- CORRESPONDING FIFOs:
  if (FLAG_WDATA = '1') then

    pointerAUX := (others => 0);
    FIFOaux_A := (others => (others => '0'));
    FIFOaux_B := (others => (others => '0'));

    for i in 7 downto 0 loop
      if (i < conv_integer(Pe_in)) then
        indx := conv_integer(memRef(i));
        if (pointerAUX(indx) = 0) then
          pointerAUX(indx) := 1;
          FIFOaux_A(indx)(ADDRESS_SIZE-1 downto 0) := address(i);
          FIFOaux_A(indx)((DATA_SIZE+ADDRESS_SIZE)-1 downto ADDRESS_SIZE) := WDATA_store(i);
        else
          pointerAUX(indx) := 2;
          FIFOaux_B(indx)(ADDRESS_SIZE-1 downto 0) := address(i);
          FIFOaux_B(indx)((DATA_SIZE+ADDRESS_SIZE)-1 downto ADDRESS_SIZE) := WDATA_store(i);
        end if;
      end if;
    end loop;  -- i

    for i in 7 downto 0 loop
      if (pointerAUX(i) = 1) then
        FIFO(pointer(i))((i+1)*(DATA_SIZE+ADDRESS_SIZE)-1 downto
                         (DATA_SIZE+ADDRESS_SIZE)*i) <= FIFOaux_A(i);
      elsif (pointerAUX(i) = 2) then
        FIFO(pointer(i))((i+1)*(DATA_SIZE+ADDRESS_SIZE)-1 downto
                         (DATA_SIZE+ADDRESS_SIZE)*i) <= FIFOaux_A(i);
        FIFO(pointer(i)+1)((i+1)*(DATA_SIZE+ADDRESS_SIZE)-1 downto
```

```
                                   (DATA_SIZE+ADDRESS_SIZE)*i) <= FIFOaux_B(i);
          end if;
          pointer(i) := pointer(i) + pointerAUX(i);
        end loop;  -- i
      end if;
      -- -------------------------------------------------------------------

      -- -------------------------------------------------------------------
      -- LOADING TO THE FIRST DELAY SIGNAL THE INPUTS WHEN THE IT'S VALID
      if (WDATA_VALID = '1') then
        -- Flag to '1'
        FLAG_WDATA <= '1';
        for i in 0 to 7 loop
          WDATA_store(i) <= WDATA((i+1)*DATA_SIZE-1 downto DATA_SIZE*i);
        end loop;  -- i
      else
        -- No DATA ready
        FLAG_WDATA <= '0';
      end if;
      -- -------------------------------------------------------------------

      -- Variable "pointer" to signal "pointerFIFO" after calculations:
      pointerFIFO <= pointer;

    end if;  -- CLEAR
  end if;  -- CLK and RST_N


end process sync;


end behav_switchWriteMemory;
```

IX. *interleaver.vhd*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


-- ============================================================================
--                      TURBO-CODE INTERLEAVER     [IEEE 802.16]
--
--  INPUTS:
--     clk          -> clock
--     clear        -> clear signal (synchronous)
--     rst_n        -> reset signal (asynchronous)
--     N            -> dimension of the block
--     PE           -> number of parallels processes
--     SCRAMBLED    -> order(0) or scrambled(1) sequence
--     BURST_LENGTH -> burst decoder length
--     START_READ   -> start to read the block
--     CLEAR_ADDGEN -> clear the "addressGenerator" to start a new block
--     WDATA        -> data read from SISO
--     WDATA_VALID  -> '1' when the WDATA is valid (all data at the same time)
--     MEM_RDATA    -> read data from memory
--
--  OUTPUTS:
--     RDATA_VALID  -> '1' when the RDATA is valid (all data at the same time)
--     RDATA        -> data reading from memory (and goes to SISOs)
--     MEM_RADX     -> addresses to read of each memory
--     MEM_REN      -> read enable of each memory
--     MEM_WADX     -> addresses to write of each memory
--     MEM_WDATA    -> data to write in memory
--     MEM_WEN      -> write enable of each memory
--
-- ============================================================================


entity interleaver is

  generic (
    DATA_SIZE    : positive := 8;
    ADDRESS_SIZE : positive := 12);

  port (
    clk          : in  std_logic;
    clear        : in  std_logic;
    rst_n        : in  std_logic;
    N            : in  std_logic_vector(11 downto 0);
    PE           : in  std_logic_vector(3 downto 0);
    SCRAMBLED    : in  std_logic;
    BURST_LENGTH : in  std_logic_vector(11 downto 0);
    START_READ   : in  std_logic;
    CLEAR_ADDGEN : in  std_logic;
    WDATA        : in  std_logic_vector(8*DATA_SIZE-1 downto 0);
    WDATA_VALID  : in  std_logic;
    MEM_RDATA    : in  std_logic_vector(8*DATA_SIZE-1 downto 0);
    RDATA_VALID  : out std_logic;
    RDATA        : out std_logic_vector(8*DATA_SIZE-1 downto 0);
    MEM_RADX     : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
    MEM_REN      : out std_logic_vector(7 downto 0);
    MEM_WADX     : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
    MEM_WDATA    : out std_logic_vector(8*DATA_SIZE-1 downto 0);
    MEM_WEN      : out std_logic_vector(7 downto 0));
```

```
end interleaver;


architecture behav_interleaver of interleaver is


  -- -------------------------------------------------------------------------
  -- -------------------------------------------------------------------------
  --            COMPONENTS:     addressGenerator
  --                            LIFOmultiple
  --                            switchReadMemory
  --                            switchWriteMemory

  component addressGenerator
    generic (
      ADDRESS_SIZE : positive);
    port (
      clk       : in  std_logic;
      enable    : in  std_logic;
      clear     : in  std_logic;
      rst_n     : in  std_logic;
      scrambled : in  std_logic;
      N_in      : in  std_logic_vector(11 downto 0);
      Pe_in     : in  std_logic_vector(3 downto 0);
      validOut  : out std_logic;
      mem_ref   : out std_logic_vector(23 downto 0);
      address   : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0));
  end component;

  component LIFOmultiple
    generic (
      ADDRESS_SIZE : positive);
    port (
      clk         : in  std_logic;
      clear       : in  std_logic;
      rst_n       : in  std_logic;
      write_en    : in  std_logic;
      read_en     : in  std_logic;
      LIFO_change : in  std_logic;
      data_in     : in  std_logic_vector(8*ADDRESS_SIZE+23 downto 0);
      data_out    : out std_logic_vector(8*ADDRESS_SIZE+23 downto 0));
  end component;

  component switchReadMemory
    generic (
      DATA_SIZE    : positive;
      ADDRESS_SIZE : positive);
    port (
      clk         : in  std_logic;
      clear       : in  std_logic;
      rst_n       : in  std_logic;
      Pe_in       : in  std_logic_vector(3 downto 0);
      address_in  : in  std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
      memRef_in   : in  std_logic_vector(23 downto 0);
      input_valid : in  std_logic;
      MEM_RDATA   : in  std_logic_vector(8*DATA_SIZE-1 downto 0);
      MEM_RADX    : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
      MEM_REN     : out std_logic_vector(7 downto 0);
      RDATA       : out std_logic_vector(8*DATA_SIZE-1 downto 0);
      RDATA_VALID : out std_logic);
  end component;

  component switchWriteMemory
    generic (
      DATA_SIZE    : positive;
```

```
      ADDRESS_SIZE : positive;
      FIFO_LENGTH  : positive);
    port (
      clk          : in  std_logic;
      clear        : in  std_logic;
      rst_n        : in  std_logic;
      Pe_in        : in  std_logic_vector(3 downto 0);
      addressing   : in  std_logic_vector(8*ADDRESS_SIZE+23 downto 0);
      WDATA        : in  std_logic_vector(8*DATA_SIZE-1 downto 0);
      WDATA_VALID  : in  std_logic;
      MEM_WADX     : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
      MEM_WDATA    : out std_logic_vector(8*DATA_SIZE-1 downto 0);
      MEM_WEN      : out std_logic_vector(7 downto 0));
  end component;

  -- -------------------------------------------------------------------------
  -- -------------------------------------------------------------------------


  -- Signals to interconnect the entities
  signal clearAddGen : std_logic := '0';
  signal addressValid_GEN : std_logic := '0';
  signal memRef_GEN : std_logic_vector(23 downto 0) := (others => '0');
  signal address_GEN : std_logic_vector(8*ADDRESS_SIZE-1 downto 0) := (others => '0');
  signal dataLIFO : std_logic_vector(8*ADDRESS_SIZE+23 downto 0) := (others => '0');
  signal addressing_LIFO : std_logic_vector(8*ADDRESS_SIZE+23 downto 0) := (others => '0');


  -- CONTROL UNIT signals:
  signal counter : std_logic_vector(11 downto 0) := (others => '0');
  signal collision : std_logic := '0';
  signal BURST_END : std_logic := '0';
  signal enableAddGen : std_logic := '0';
  type states is (STATE0, STATE1, STATE2, STATE3, STATE4, STATE5);
  signal nextSTATE : states;


begin  -- behav_interleaver


  -- Making "dataLIFO" signal:
  dataLIFO(23 downto 0) <= memRef_GEN;
  dataLIFO(8*ADDRESS_SIZE+23 downto 24) <= address_GEN;




  -- -------------------------------------------------------------------------
  --            CONTROL UNIT:
  --
  --    We have a block of N units. This block we divided by windows of length
  --   "BURST_LENGTH":
  --
  --    <-------------------------------- N -------------------------------->
  --    <------------> <------------> <---- - - - - - - - ----> <------------>
  --   ----------------------------------------------------------------------
  --   |   window_0   |   window_1   |          . . . .         |   window_j   |
  --   ----------------------------------------------------------------------
  --    \_____  _____/ \_____  _____/                          \_____  _____/
  --          \/             \/                                       \/
  --     BURST_LENGTH   BURST_LENGTH                             BURST_LENGTH
  --       elements       elements                                 elements
  --
  --
  --                              BURST_LENGTH <= N
```

```
--
--    The signals that control the process are:       START_READ
--                                                    CLEAR_ADDGEN
--
--    With "START_READ" we began to read from memory so many words (index)
-- as "BURST_LENGTH", then, we stop and waiting for another "START_READ".
--    On the other hand, "CLEAR_ADDGEN" saids when a new block of N elements
-- is incoming, and we must clean the "addressGenerator".
--    So, if arrives a "clear" or "CLEAR_ADDGEN" we clean "addressGenerator":

clearAddGen <= clear or CLEAR_ADDGEN;


--    In each arrival of a "START_READ" we must to change the LIFO on wich
-- we are reading/writing. Let's to imagine a block of N elements that has
-- been divided in 3 windows, so we have 4 times:
--
--        ----------------------------------------------
--        |   window_1    |    window_2    |    window_3    |
--        ----------------------------------------------
--              ALPHA
--    1)   ------------->
--              BETA           ALPHA
--    2)   <-------------.-------------->
--                             BETA           ALPHA
--    3)                  <--------------.------------->
--                                            BETA
--    4)                                 <-------------
--
--
--
-- DATA_IN   _____________ WIN_1 ____ WIN_2 ____ WIN_3 ______________
--
--                _                                            _
-- CLEAR_ADDGEN  ___| |________________________________________| |__
--
--                   _           _           _
-- START_READ _________| |________| |________| |_____________________
--
--
--    So, "LIFO_change" at "LIFOmultiple" is connected with "START_READ":
--
--    LIFO : LIFOmultiple
--      generic map (
--        ADDRESS_SIZE => ADDRESS_SIZE)
--      port map (
--        clk         => clk,
--        clear       => clear,
--        rst_n       => rst_n,
--        write_en    => addressValid_GEN,
--        read_en     => WDATA_VALID,
--  ===>  LIFO_change => START_READ,   <========
--        data_in     => dataLIFO,
--        data_out    => addressing_LIFO);
--
--


--
--
--
--            ADDRESS GENERATOR ENABLE:
--
--    In the beginning, "addressGenerator" needs one "enable" more to
-- initiate the work.
--    For example, if we have a "BURST_LENGTH" = 9, the "enable" signal of
```

```
-- the "addressGenerator" must be 10 cycles high after the first "START_READ"
--
-- ENABLE          _________  10   ____   9   ____   9   ______________
-- (cycles high)
--                     _                                             _
-- CLEAR_ADDGEN   ___| |_________________________________________| |__
--
--
--                        _          _          _
-- START_READ _________| |________| |________| |_____________________
--
--
--    We have another asynchronous signal that says us if we  have finished of
-- reading the window:

BURST_END <= '1' when ( counter >= BURST_LENGTH ) else
             '0';
--
--
--            COLLISIONS:
--
--    The collisions only appears when:
--            N = 108   Pe = 2
--            N = 108   Pe = 4
--            N = 108   Pe = 6
--            N = 72    Pe = 8
--    and SCRAMBLED = '1'
--
--    So, we make an asynchronous signal to verify if there's a collision or
-- not, "collision":


collision <= '1' when ( SCRAMBLED = '1' and
                        (
                          ( N = conv_std_logic_vector(108,12) and
                            PE = conv_std_logic_vector(2,4) ) or

                          ( N = conv_std_logic_vector(108,12) and
                            PE = conv_std_logic_vector(4,4) ) or

                          ( N = conv_std_logic_vector(108,12) and
                            PE = conv_std_logic_vector(6,4) ) or

                          ( N = conv_std_logic_vector(72,12) and
                            PE = conv_std_logic_vector(8,4) )

                        )) else
             '0';

--    When we have a collision we must to make an intermittent enable,
-- because all collision take place by pairs and with this form we solved
-- the problem of collisions:
--
--             _   _   _   _   _   _   _   _   _   _   _   _
--    CLK   _| |_| |_| |_| |_| |_| |_| |_| |_| |_| |_| |_| |_| |_
--             ___     ___     ___     ___     ___     ___
-- ENABLE __|   |___|   |___|   |___|   |___|   |___|   |___
--
--
--
--
--    With all these considerations we can make an algorithm:
--
--                               MEALY STATE MACHINE:       input/outputs
--
--                    CLEAR_ADDGEN = 1/
```

```
--                    or  CLEAR = 1 / enable = 0
--                    or  RST_N = 0/
--                                  |
--                                  |
--                                  |
--                                 \|/
--                             ------------
--                             |  STATE0  |
--                             ------------
--                                  |
--   BURST_END / enable = 1         |  START_READ = 1 / enable = 1
--      = 0   / count++             |
--        -----                    \|/
--       |     |               ------------
--       |     |       --------|  STATE1  |-----------
--       |     |      |        ------------          |
--       |     |      |                              | COLLISION / enable = 1
--       |     |      | COLLISON / enable = 1        |     = 1  / count = 1
--       |    \|/     |     = 0  / count = 1         |
--     ------------   |                             \|/
--     |  STATE2  |<---                         ------------
--     ------------                             |  STATE3  |------>---------
--       |     /|\                              ------------               |
--       |      |                                |  /|\ /|\                |
--       |      |  START_READ = 1 / count = 1    |   |   |                 |
--       |      | & COLLISON = 0 / enable = 1    |   |   |                 |
--       |      |                                |   |   |                 |
--       |      |                                |   |   | / enable = 1    |
--       |      |                                |   |   | / count++       |
--       |      |                                |   |   |                 |
--       |      |     BURST_END = 1 / enable = 0 |   |   |                 |
--       |      |                    ----<-------|   |   |                 |
--       |      |                    |               |   |                \|/
--       |      |                    |     ----->-----   |                 |
--       |      |                    |     |             |                 |
--       |      |                    |     |             |                 |
--       |      |                    |     |   ------------                |
--       |      |                    |     |   |  STATE4  |                |
--       |      |                    |     |   ------------                |
--       |      |                    |     |   /|\                         |
--       |      |                    |     |    |                          |
--       |      |                    |     |    |BURST_END = 0 / enable = 0|
--       |      |                    |     |    |                          |
--      \|/     |                   \|/   /|\   |---------------------------
--       |      |                    |     |
--       |      |                    |     |
--       |      |                    |     |  START_READ = 1 / count = 1
--       |      |                    |     | & COLLISON = 1 / enable = 1
--       |      |                    |     |
--       |      |                    |     |
--       |      |                 --------------
--       |      |-----<-----------|            |
--       |                        |   STATE5   |
--       |----------------------->|            |
--  BURST_END = 1 / enable = 0    --------------
--
--


controlUnit: process (clk, rst_n)
begin  -- process controlUnit

  if (rst_n = '0') then
    nextSTATE <= STATE0;
    enableAddGen <= '0';
```

```
    counter <= (others => '0');

  elsif (clk = '1' and clk'event) then
    if (clear = '1' or CLEAR_ADDGEN = '1') then
      nextSTATE <= STATE0;
      enableAddGen <= '0';
      counter <= (others => '0');

    else

      case nextSTATE IS

        when STATE0 =>
          if (START_READ = '1') then
            nextSTATE <= STATE1;
            enableAddGen <= '1';
          end if;

        when STATE1 =>
          if (collision = '0') then
            nextSTATE <= STATE2;
            enableAddGen <= '1';
            counter <= conv_std_logic_vector(1,12);
          else
            nextSTATE <= STATE3;
            enableAddGen <= '1';
            counter <= conv_std_logic_vector(1,12);
          end if;

        when STATE2 =>
          if (BURST_END = '0') then
            nextSTATE <= STATE2;
            enableAddGen <= '1';
            counter <= counter + 1;
          else
            nextSTATE <= STATE5;
            enableAddGen <= '0';
          end if;

        when STATE3 =>
          if (BURST_END = '0') then
            nextSTATE <= STATE4;
            enableAddGen <= '0';
          else
            nextSTATE <= STATE5;
            enableAddGen <= '0';
          end if;

        when STATE4 =>
          nextSTATE <= STATE3;
          enableAddGen <= '1';
          counter <= counter + 1;

        when STATE5 =>
          if (START_READ = '1') then
            enableAddGen <= '1';
            counter <= conv_std_logic_vector(1,12);
            if (collision = '0') then
              nextSTATE <= STATE2;
            else
              nextSTATE <= STATE3;
            end if;
          end if;

      end case;
```

```
    end if;  -- CLEAR & CLEAR_ADDGEN
  end if;  -- CLK & RST_N

end process controlUnit;
-- -------------------------------------------------------------------------



-- -------------------------------------------------------------------------
-- -------------------------------------------------------------------------
--    CREATION AND CONNECTION OF THE COMPONENTS

addGen : addressGenerator
  generic map (
    ADDRESS_SIZE => ADDRESS_SIZE)
  port map (
    clk       => clk,
    enable    => enableAddGen,
    clear     => clearAddGen,
    rst_n     => rst_n,
    scrambled => SCRAMBLED,
    N_in      => N,
    Pe_in     => PE,
    validOut  => addressValid_GEN,
    mem_ref   => memRef_GEN,
    address   => address_GEN);

LIFO : LIFOmultiple
  generic map (
    ADDRESS_SIZE => ADDRESS_SIZE)
  port map (
    clk         => clk,
    clear       => clear,
    rst_n       => rst_n,
    write_en    => addressValid_GEN,
    read_en     => WDATA_VALID,
    LIFO_change => START_READ,        -- On each "START_READ", change LIFO
    data_in     => dataLIFO,
    data_out    => addressing_LIFO);

SRM : switchReadMemory
  generic map (
    DATA_SIZE    => DATA_SIZE,
    ADDRESS_SIZE => ADDRESS_SIZE)
  port map (
    clk         => clk,
    clear       => clear,
    rst_n       => rst_n,
    Pe_in       => PE,
    address_in  => address_GEN,
    memRef_in   => memRef_GEN,
    input_valid => addressValid_GEN,
    MEM_RDATA   => MEM_RDATA,
    MEM_RADX    => MEM_RADX,
    MEM_REN     => MEM_REN,
    RDATA       => RDATA,
    RDATA_VALID => RDATA_VALID);

SWM : switchWriteMemory
  generic map (
    DATA_SIZE    => DATA_SIZE,
    ADDRESS_SIZE => ADDRESS_SIZE,
    FIFO_LENGTH  => 6)
  port map (
    clk         => clk,
```

```
      clear       => clear,
      rst_n       => rst_n,
      Pe_in       => PE,
      addressing  => addressing_LIFO,
      WDATA       => WDATA,
      WDATA_VALID => WDATA_VALID,
      MEM_WADX    => MEM_WADX,
      MEM_WDATA   => MEM_WDATA,
      MEM_WEN     => MEM_WEN);

  -- ---------------------------------------------------------------------------
  -- ---------------------------------------------------------------------------



end behav_interleaver;
```

X. *tb_indexGenerator_report_ORDER.vhd*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library std;
use std.textio.all;

ENTITY tb_indexGenerator_report_ORDER_vhd IS
END tb_indexGenerator_report_ORDER_vhd;

ARCHITECTURE behavior OF tb_indexGenerator_report_ORDER_vhd IS

  -- Component Declaration for the Unit Under Test (UUT)
  COMPONENT indexgenerator
    PORT(
      clk : IN std_logic;
      enable : IN std_logic;
      clear : IN std_logic;
      rst_n : IN std_logic;
      scrambled : IN std_logic;
      N_in : IN std_logic_vector(11 downto 0);
      Pe_in : IN std_logic_vector(3 downto 0);
      validOut : OUT std_logic;
      indexOut : OUT std_logic_vector(95 downto 0);
      mult_nBus : OUT std_logic_vector(83 downto 0);
      indexBus0 : out std_logic_vector(11 downto 0);
      indexBus1 : out std_logic_vector(11 downto 0);
      indexBus2 : out std_logic_vector(11 downto 0);
      indexBus3 : out std_logic_vector(11 downto 0);
      indexBus4 : out std_logic_vector(11 downto 0);
      indexBus5 : out std_logic_vector(11 downto 0);
      indexBus6 : out std_logic_vector(11 downto 0);
      indexBus7 : out std_logic_vector(11 downto 0);
      s_n  : out std_logic_vector(11 downto 0);
      s_2n : out std_logic_vector(11 downto 0);
      s_3n : out std_logic_vector(11 downto 0);
      s_4n : out std_logic_vector(11 downto 0);
      s_5n : out std_logic_vector(11 downto 0);
      s_6n : out std_logic_vector(11 downto 0);
      s_7n : out std_logic_vector(11 downto 0));
  END COMPONENT;

  --Inputs
  SIGNAL clk :  std_logic := '1';
  SIGNAL enable :  std_logic := '0';
  SIGNAL clear :  std_logic := '1';
  SIGNAL rst_n :  std_logic := '0';
  SIGNAL scrambled :  std_logic := '0';
  SIGNAL N_in :  std_logic_vector(11 downto 0) := (others => '0');
  SIGNAL Pe_in :  std_logic_vector(3 downto 0) := (others => '0');

  --Outputs
  SIGNAL validOut  : std_logic;
  SIGNAL indexOut  : std_logic_vector(95 downto 0);
  SIGNAL mult_nBus : std_logic_vector(83 downto 0);
  SIGNAL indexBus0 : std_logic_vector(11 downto 0);
  SIGNAL indexBus1 : std_logic_vector(11 downto 0);
  SIGNAL indexBus2 : std_logic_vector(11 downto 0);
  SIGNAL indexBus3 : std_logic_vector(11 downto 0);
  SIGNAL indexBus4 : std_logic_vector(11 downto 0);
  SIGNAL indexBus5 : std_logic_vector(11 downto 0);
  SIGNAL indexBus6 : std_logic_vector(11 downto 0);
```

```
  SIGNAL indexBus7 : std_logic_vector(11 downto 0);
  SIGNAL s_n : std_logic_vector(11 downto 0);
  SIGNAL s_2n : std_logic_vector(11 downto 0);
  SIGNAL s_3n : std_logic_vector(11 downto 0);
  SIGNAL s_4n : std_logic_vector(11 downto 0);
  SIGNAL s_5n : std_logic_vector(11 downto 0);
  SIGNAL s_6n : std_logic_vector(11 downto 0);
  SIGNAL s_7n : std_logic_vector(11 downto 0);

  --Constant CLK
  constant period : time := 100 ns;

  --String constants
  constant N_txt : string := "N = ";
  constant Pe_txt : string := "Pe = ";
  constant blank_txt : string := " ";
  constant empty_txt : string := "";
  constant change_txt : string := "***";
  constant space_txt : string := " - ";
  constant C_char : string := " C";
  constant end_txt : string := "END";


  --Constant for using at the simulator
  type arrayPe is array (4 downto 0) of std_logic_vector(3 downto 0);
  type arrayN is array (16 downto 0) of std_logic_vector(11 downto 0);
  constant Pe : arrayPe := ("1000", "0110", "0100", "0010", "0001");
  constant N : arrayN := (conv_std_logic_vector(2400,12),
                          conv_std_logic_vector(1920,12),
                          conv_std_logic_vector(1440,12),
                          conv_std_logic_vector(960,12),
                          conv_std_logic_vector(480,12),
                          conv_std_logic_vector(240,12),
                          conv_std_logic_vector(216,12),
                          conv_std_logic_vector(192,12),
                          conv_std_logic_vector(180,12),
                          conv_std_logic_vector(144,12),
                          conv_std_logic_vector(120,12),
                          conv_std_logic_vector(108,12),
                          conv_std_logic_vector(96,12),
                          conv_std_logic_vector(72,12),
                          conv_std_logic_vector(48,12),
                          conv_std_logic_vector(36,12),
                          conv_std_logic_vector(24,12));

BEGIN

  -- Instantiate the Unit Under Test (UUT)
  uut: indexgenerator PORT MAP(
    clk => clk,
    enable => enable,
    clear => clear,
    rst_n => rst_n,
    scrambled => scrambled,
    N_in => N_in,
    Pe_in => Pe_in,
    validOut => validOut,
    indexOut => indexOut,
    mult_nBus => mult_nBus);

  clk <= not clk after period/2;

  -- Separating output bus:
  indexBus0 <= indexOut(11 downto 0);
  indexBus1 <= indexOut(23 downto 12);
```

```
indexBus2 <= indexOut(35 downto 24);
indexBus3 <= indexOut(47 downto 36);
indexBus4 <= indexOut(59 downto 48);
indexBus5 <= indexOut(71 downto 60);
indexBus6 <= indexOut(83 downto 72);
indexBus7 <= indexOut(95 downto 84);
s_n <= mult_nBus(11 downto 0);
s_2n <= mult_nBus(23 downto 12);
s_3n <= mult_nBus(35 downto 24);
s_4n <= mult_nBus(47 downto 36);
s_5n <= mult_nBus(59 downto 48);
s_6n <= mult_nBus(71 downto 60);
s_7n <= mult_nBus(83 downto 72);


tb : PROCESS

  file fileTXT : text open WRITE_MODE is "report_indexGenerator_ORDER_VHDL.txt";
  variable tmp : integer;
  variable ptr  : line;
  variable counter : integer := 0;

BEGIN

  for i_Pe in 0 to 4 loop
    Pe_in <= Pe(i_Pe);

    for i_N in 0 to 16 loop
      N_in <= N(i_N);
      wait for period;

      -- Excluding cases:
      if ( not( Pe_in = "1000" and (N_in = conv_std_logic_vector(36,12) or
                                    N_in = conv_std_logic_vector(108,12) or
                                    N_in = conv_std_logic_vector(180,12)) ) ) then
        -------------------------------------------------------------------
        --  WRITE TO FILE THE INFORMATION OF THE ACTUAL OPERATION
        tmp := conv_integer(N_in);
        write(ptr,N_txt);
        write(ptr,tmp);
        writeline(fileTXT,ptr);
        tmp := conv_integer(Pe_in);
        write(ptr,Pe_txt);
        write(ptr,tmp);
        writeline(fileTXT,ptr);
        -------------------------------------------------------------------

        -- Counter -> 0
        counter := 0;

        -- SIMULATION
        enable <= '0';
        clear<= '1';
        rst_n <= '0';
        wait for 2*period;
        clear <= '0';
        rst_n <= '1';
        wait for 2*period;

        while (counter < conv_integer(N_in)/conv_integer(Pe_in)) loop
          enable <= '1';

          if (validOut = '1') then

            counter := counter + 1;
```

```
-- Output valid, so, WRITE TO FILE!!!!
----------------------------------------------------------------
--                              Pe = 1
if (Pe_in = "0001") then
  tmp := conv_integer(indexBus0);
  write(ptr,tmp);

--                              Pe = 2
elsif (Pe_in = "0010") then
  tmp := conv_integer(indexBus0);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus1);
  write(ptr,tmp);

--                              Pe = 4
elsif (Pe_in = "0100") then
  tmp := conv_integer(indexBus0);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus1);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus2);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus3);
  write(ptr,tmp);

--                              Pe = 6
elsif (Pe_in = "0110") then
  tmp := conv_integer(indexBus0);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus1);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus2);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus3);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus4);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus5);
  write(ptr,tmp);

--                              Pe = 8
elsif (Pe_in = "1000") then
  tmp := conv_integer(indexBus0);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus1);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus2);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus3);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus4);
```

```
              write(ptr,tmp);
              write(ptr,blank_txt);
              tmp := conv_integer(indexBus5);
              write(ptr,tmp);
              write(ptr,blank_txt);
              tmp := conv_integer(indexBus6);
              write(ptr,tmp);
              write(ptr,blank_txt);
              tmp := conv_integer(indexBus7);
              write(ptr,tmp);
            end if;                     -- Pe's

            writeline(fileTXT,ptr);

          end if;                       -- DATA OUT

          wait for period;

        end loop;                       -- WHILE

        enable <= '0';
        clear <= '1';

        -------------------------------------------------------------------
        --  WRITE TO FILE INDICATOR CHANGE OF PARAMETERS: "***"
        write(ptr,change_txt);
        writeline(fileTXT,ptr);
        -------------------------------------------------------------------
        wait for 3*period;

      end if;                           -- IMPOSSIBLES CASES

    end loop;  -- i_N
  end loop;  -- i_Pe

  -- WRITING A FLAG TO END:
  write(ptr,end_txt);
  writeline(fileTXT,ptr);

  wait;


END PROCESS;

END;
```

XI. *tb_indexGenerator_report_SCRAMB.vhd*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library std;
use std.textio.all;

ENTITY tb_indexGenerator_report_SCRAMB_vhd IS
END tb_indexGenerator_report_SCRAMB_vhd;

ARCHITECTURE behavior OF tb_indexGenerator_report_SCRAMB_vhd IS

  -- Component Declaration for the Unit Under Test (UUT)
  COMPONENT indexgenerator
    PORT(
      clk : IN std_logic;
      enable : IN std_logic;
      clear : IN std_logic;
      rst_n : IN std_logic;
      scrambled : IN std_logic;
      N_in : IN std_logic_vector(11 downto 0);
      Pe_in : IN std_logic_vector(3 downto 0);
      validOut : OUT std_logic;
      indexOut : OUT std_logic_vector(95 downto 0);
      mult_nBus : OUT std_logic_vector(83 downto 0);
      indexBus0 : out std_logic_vector(11 downto 0);
      indexBus1 : out std_logic_vector(11 downto 0);
      indexBus2 : out std_logic_vector(11 downto 0);
      indexBus3 : out std_logic_vector(11 downto 0);
      indexBus4 : out std_logic_vector(11 downto 0);
      indexBus5 : out std_logic_vector(11 downto 0);
      indexBus6 : out std_logic_vector(11 downto 0);
      indexBus7 : out std_logic_vector(11 downto 0);
      s_n  : out std_logic_vector(11 downto 0);
      s_2n : out std_logic_vector(11 downto 0);
      s_3n : out std_logic_vector(11 downto 0);
      s_4n : out std_logic_vector(11 downto 0);
      s_5n : out std_logic_vector(11 downto 0);
      s_6n : out std_logic_vector(11 downto 0);
      s_7n : out std_logic_vector(11 downto 0));
  END COMPONENT;

  --Inputs
  SIGNAL clk :  std_logic := '1';
  SIGNAL enable :  std_logic := '0';
  SIGNAL clear :  std_logic := '1';
  SIGNAL rst_n :  std_logic := '0';
  SIGNAL scrambled :  std_logic := '1';
  SIGNAL N_in :  std_logic_vector(11 downto 0) := (others => '0');
  SIGNAL Pe_in :  std_logic_vector(3 downto 0) := (others => '0');

  --Outputs
  SIGNAL validOut  : std_logic;
  SIGNAL indexOut  : std_logic_vector(95 downto 0);
  SIGNAL mult_nBus : std_logic_vector(83 downto 0);
  SIGNAL indexBus0 : std_logic_vector(11 downto 0);
  SIGNAL indexBus1 : std_logic_vector(11 downto 0);
  SIGNAL indexBus2 : std_logic_vector(11 downto 0);
  SIGNAL indexBus3 : std_logic_vector(11 downto 0);
  SIGNAL indexBus4 : std_logic_vector(11 downto 0);
  SIGNAL indexBus5 : std_logic_vector(11 downto 0);
  SIGNAL indexBus6 : std_logic_vector(11 downto 0);
```

```
  SIGNAL indexBus7 : std_logic_vector(11 downto 0);
  SIGNAL s_n : std_logic_vector(11 downto 0);
  SIGNAL s_2n : std_logic_vector(11 downto 0);
  SIGNAL s_3n : std_logic_vector(11 downto 0);
  SIGNAL s_4n : std_logic_vector(11 downto 0);
  SIGNAL s_5n : std_logic_vector(11 downto 0);
  SIGNAL s_6n : std_logic_vector(11 downto 0);
  SIGNAL s_7n : std_logic_vector(11 downto 0);

  --Constant CLK
  constant period : time := 100 ns;

  --String constants
  constant N_txt : string := "N = ";
  constant Pe_txt : string := "Pe = ";
  constant blank_txt : string := " ";
  constant empty_txt : string := "";
  constant change_txt : string := "***";
  constant space_txt : string := " - ";
  constant C_char : string := " C";
  constant end_txt : string := "END";

  --Constant for using at the simulator
  type arrayPe is array (4 downto 0) of std_logic_vector(3 downto 0);
  type arrayN is array (16 downto 0) of std_logic_vector(11 downto 0);
  constant Pe : arrayPe := ("1000", "0110", "0100", "0010", "0001");
  constant N : arrayN := (conv_std_logic_vector(2400,12),
                          conv_std_logic_vector(1920,12),
                          conv_std_logic_vector(1440,12),
                          conv_std_logic_vector(960,12),
                          conv_std_logic_vector(480,12),
                          conv_std_logic_vector(240,12),
                          conv_std_logic_vector(216,12),
                          conv_std_logic_vector(192,12),
                          conv_std_logic_vector(180,12),
                          conv_std_logic_vector(144,12),
                          conv_std_logic_vector(120,12),
                          conv_std_logic_vector(108,12),
                          conv_std_logic_vector(96,12),
                          conv_std_logic_vector(72,12),
                          conv_std_logic_vector(48,12),
                          conv_std_logic_vector(36,12),
                          conv_std_logic_vector(24,12));

BEGIN

  -- Instantiate the Unit Under Test (UUT)
  uut: indexgenerator PORT MAP(
    clk => clk,
    enable => enable,
    clear => clear,
    rst_n => rst_n,
    scrambled => scrambled,
    N_in => N_in,
    Pe_in => Pe_in,
    validOut => validOut,
    indexOut => indexOut,
    mult_nBus => mult_nBus);

  clk <= not clk after period/2;

  -- Separating output bus:
  indexBus0 <= indexOut(11 downto 0);
  indexBus1 <= indexOut(23 downto 12);
  indexBus2 <= indexOut(35 downto 24);
```

```
indexBus3 <= indexOut(47 downto 36);
indexBus4 <= indexOut(59 downto 48);
indexBus5 <= indexOut(71 downto 60);
indexBus6 <= indexOut(83 downto 72);
indexBus7 <= indexOut(95 downto 84);
s_n <= mult_nBus(11 downto 0);
s_2n <= mult_nBus(23 downto 12);
s_3n <= mult_nBus(35 downto 24);
s_4n <= mult_nBus(47 downto 36);
s_5n <= mult_nBus(59 downto 48);
s_6n <= mult_nBus(71 downto 60);
s_7n <= mult_nBus(83 downto 72);

tb : PROCESS

  file fileTXT : text open WRITE_MODE is "report_indexGenerator_SCRAMB_VHDL.txt";
  variable tmp : integer;
  variable ptr  : line;
  variable counter : integer := 0;

BEGIN

  for i_Pe in 0 to 4 loop
    Pe_in <= Pe(i_Pe);

    for i_N in 0 to 16 loop
      N_in <= N(i_N);
      wait for period;

      -- Excluding cases:
      if ( not( Pe_in = "1000" and (N_in = conv_std_logic_vector(36,12) or
                                    N_in = conv_std_logic_vector(108,12) or
                                    N_in = conv_std_logic_vector(180,12)) ) ) then
        ------------------------------------------------------------------
        --  WRITE TO FILE THE INFORMATION OF THE ACTUAL OPERATION
        tmp := conv_integer(N_in);
        write(ptr,N_txt);
        write(ptr,tmp);
        writeline(fileTXT,ptr);
        tmp := conv_integer(Pe_in);
        write(ptr,Pe_txt);
        write(ptr,tmp);
        writeline(fileTXT,ptr);
        ------------------------------------------------------------------

        -- Counter -> 0
        counter := 0;

        -- SIMULATION
        enable <= '0';
        clear<= '1';
        rst_n <= '0';
        wait for 2*period;
        clear <= '0';
        rst_n <= '1';
        wait for 2*period;

        while (counter < conv_integer(N_in)/conv_integer(Pe_in)) loop
          enable <= '1';

          if (validOut = '1') then

            counter := counter + 1;

            -- Output valid, so, WRITE TO FILE!!!!
```

```
---------------------------------------------------------------
--                              Pe = 1
if (Pe_in = "0001") then
  tmp := conv_integer(indexBus0);
  write(ptr,tmp);

--                              Pe = 2
elsif (Pe_in = "0010") then
  tmp := conv_integer(indexBus0);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus1);
  write(ptr,tmp);

--                              Pe = 4
elsif (Pe_in = "0100") then
  tmp := conv_integer(indexBus0);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus1);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus2);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus3);
  write(ptr,tmp);

--                              Pe = 6
elsif (Pe_in = "0110") then
  tmp := conv_integer(indexBus0);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus1);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus2);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus3);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus4);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus5);
  write(ptr,tmp);

--                              Pe = 8
elsif (Pe_in = "1000") then
  tmp := conv_integer(indexBus0);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus1);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus2);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus3);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(indexBus4);
  write(ptr,tmp);
  write(ptr,blank_txt);
```

```
              tmp := conv_integer(indexBus5);
              write(ptr,tmp);
              write(ptr,blank_txt);
              tmp := conv_integer(indexBus6);
              write(ptr,tmp);
              write(ptr,blank_txt);
              tmp := conv_integer(indexBus7);
              write(ptr,tmp);
            end if;                    -- Pe's

            writeline(fileTXT,ptr);

          end if;                      -- DATA OUT

          wait for period;

        end loop;                      -- WHILE

        enable <= '0';
        clear <= '1';

        -------------------------------------------------------------------
        --  WRITE TO FILE INDICATOR CHANGE OF PARAMETERS: "***"
        write(ptr,change_txt);
        writeline(fileTXT,ptr);
        -------------------------------------------------------------------
        wait for 3*period;

      end if;                          -- IMPOSSIBLES CASES

    end loop;  -- i_N
  end loop;  -- i_Pe

  -- WRITING A FLAG TO END:
  write(ptr,end_txt);
  writeline(fileTXT,ptr);

  wait;

 END PROCESS;

END;
```

XII. *tb_adxGen_ORDER.vhd*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library std;
use std.textio.all;

ENTITY tb_adxGen_ORDER_vhd IS
END tb_adxGen_ORDER_vhd;

ARCHITECTURE behavior OF tb_adxGen_ORDER_vhd IS

  -- Component Declaration for the Unit Under Test (UUT)
  COMPONENT addressGenerator
    PORT(
      clk : IN std_logic;
      enable : IN std_logic;
      clear : IN std_logic;
      rst_n : IN std_logic;
      scrambled : IN std_logic;
      N_in : IN std_logic_vector(11 downto 0);
      Pe_in : IN std_logic_vector(3 downto 0);
      validOut : OUT std_logic;
      mem_ref : OUT std_logic_vector(23 downto 0);
      address : OUT std_logic_vector(119 downto 0));
  END COMPONENT;

  --Inputs
  SIGNAL clk :  std_logic := '1';
  SIGNAL enable :  std_logic := '0';
  SIGNAL clear :  std_logic := '1';
  SIGNAL rst_n :  std_logic := '0';
  SIGNAL scrambled :  std_logic := '0';
  SIGNAL N_in :  std_logic_vector(11 downto 0) := (others=>'0');
  SIGNAL Pe_in :  std_logic_vector(3 downto 0) := (others=>'0');

  --Outputs
  SIGNAL validOut :  std_logic;
  SIGNAL mem_ref :  std_logic_vector(23 downto 0);
  SIGNAL address :  std_logic_vector(119 downto 0);

  --Auxiliar signals
  signal mem_ref0 :  std_logic_vector(2 downto 0);
  signal mem_ref1 :  std_logic_vector(2 downto 0);
  signal mem_ref2 :  std_logic_vector(2 downto 0);
  signal mem_ref3 :  std_logic_vector(2 downto 0);
  signal mem_ref4 :  std_logic_vector(2 downto 0);
  signal mem_ref5 :  std_logic_vector(2 downto 0);
  signal mem_ref6 :  std_logic_vector(2 downto 0);
  signal mem_ref7 :  std_logic_vector(2 downto 0);
  signal address0 :  std_logic_vector(14 downto 0);
  signal address1 :  std_logic_vector(14 downto 0);
  signal address2 :  std_logic_vector(14 downto 0);
  signal address3 :  std_logic_vector(14 downto 0);
  signal address4 :  std_logic_vector(14 downto 0);
  signal address5 :  std_logic_vector(14 downto 0);
  signal address6 :  std_logic_vector(14 downto 0);
  signal address7 :  std_logic_vector(14 downto 0);

  --Time constants
  constant period : time := 100 ns;
```

```
  --String constants
  constant N_txt : string := "N = ";
  constant Pe_txt : string := "Pe = ";
  constant blank_txt : string := " ";
  constant empty_txt : string := "";
  constant change_txt : string := "***";
  constant space_txt : string := " - ";
  constant C_char : string := " C";
  constant end_txt : string := "END";


  --Constant for using at the simulator
  type arrayPe is array (4 downto 0) of std_logic_vector(3 downto 0);
  type arrayN is array (16 downto 0) of std_logic_vector(11 downto 0);
  constant Pe : arrayPe := ("1000", "0110", "0100", "0010", "0001");
  constant N : arrayN := (conv_std_logic_vector(2400,12),
                          conv_std_logic_vector(1920,12),
                          conv_std_logic_vector(1440,12),
                          conv_std_logic_vector(960,12),
                          conv_std_logic_vector(480,12),
                          conv_std_logic_vector(240,12),
                          conv_std_logic_vector(216,12),
                          conv_std_logic_vector(192,12),
                          conv_std_logic_vector(180,12),
                          conv_std_logic_vector(144,12),
                          conv_std_logic_vector(120,12),
                          conv_std_logic_vector(108,12),
                          conv_std_logic_vector(96,12),
                          conv_std_logic_vector(72,12),
                          conv_std_logic_vector(48,12),
                          conv_std_logic_vector(36,12),
                          conv_std_logic_vector(24,12));

BEGIN

  -- Instantiate the Unit Under Test (UUT)
  uut: addressGenerator PORT MAP(
    clk => clk,
    enable => enable,
    clear => clear,
    rst_n => rst_n,
    scrambled => scrambled,
    N_in => N_in,
    Pe_in => Pe_in,
    validOut => validOut,
    mem_ref => mem_ref,
    address => address);

  -- cycling the clk
  clk <= NOT clk AFTER period/2;

  -- Separating "mem_ref" & "address"
  mem_ref0 <= mem_ref(2 downto 0);
  mem_ref1 <= mem_ref(5 downto 3);
  mem_ref2 <= mem_ref(8 downto 6);
  mem_ref3 <= mem_ref(11 downto 9);
  mem_ref4 <= mem_ref(14 downto 12);
  mem_ref5 <= mem_ref(17 downto 15);
  mem_ref6 <= mem_ref(20 downto 18);
  mem_ref7 <= mem_ref(23 downto 21);
  address0 <= address(14 downto 0);
  address1 <= address(29 downto 15);
  address2 <= address(44 downto 30);
  address3 <= address(59 downto 45);
```

```
address4 <= address(74 downto 60);
address5 <= address(89 downto 75);
address6 <= address(104 downto 90);
address7 <= address(119 downto 105);

tb : PROCESS

  file fileTXT : text open WRITE_MODE is "report_addressGen_ORDER_VHDL.txt";
  variable tmp : integer;
  variable ptr  : line;
  variable counter : integer := 0;

BEGIN

  for i_Pe in 0 to 4 loop
    Pe_in <= Pe(i_Pe);

    for i_N in 0 to 16 loop
      N_in <= N(i_N);
      wait for period;

      -- Excluding cases:
      if ( not( Pe_in = "1000" and (N_in = conv_std_logic_vector(36,12) or
                                    N_in = conv_std_logic_vector(108,12) or
                                    N_in = conv_std_logic_vector(180,12)) ) ) then
        -------------------------------------------------------------------
        --  WRITE TO FILE THE INFORMATION OF THE ACTUAL OPERATION
        tmp := conv_integer(N_in);
        write(ptr,N_txt);
        write(ptr,tmp);
        writeline(fileTXT,ptr);
        tmp := conv_integer(Pe_in);
        write(ptr,Pe_txt);
        write(ptr,tmp);
        writeline(fileTXT,ptr);
        -------------------------------------------------------------------

        -- Counter -> 0
        counter := 0;

        -- SIMULATION
        enable <= '0';
        clear<= '1';
        rst_n <= '0';
        wait for 2*period;
        clear <= '0';
        rst_n <= '1';
        wait for 2*period;

        while (counter < conv_integer(N_in)/conv_integer(Pe_in)) loop
          enable <= '1';

          if (validOut = '1') then

            counter := counter + 1;

            -- Output valid, so, WRITE TO FILE!!!!
            ---------------------------------------------------------------
            --                              Pe = 1
            if (Pe_in = "0001") then
                tmp := conv_integer(address0);
                write(ptr,tmp);
                write(ptr,space_txt);

                tmp := conv_integer(mem_ref0);
```

```
        write(ptr,tmp);

    --                              Pe = 2
    elsif (Pe_in = "0010") then
        tmp := conv_integer(address0);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address1);
        write(ptr,tmp);
        write(ptr,space_txt);

        tmp := conv_integer(mem_ref0);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref1);
        write(ptr,tmp);

    --                              Pe = 4
    elsif (Pe_in = "0100") then
        tmp := conv_integer(address0);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address1);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address2);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address3);
        write(ptr,tmp);
        write(ptr,space_txt);

        tmp := conv_integer(mem_ref0);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref1);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref2);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref3);
        write(ptr,tmp);

    --                              Pe = 6
    elsif (Pe_in = "0110") then
        tmp := conv_integer(address0);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address1);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address2);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address3);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address4);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address5);
        write(ptr,tmp);
        write(ptr,space_txt);
```

```
        tmp := conv_integer(mem_ref0);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref1);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref2);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref3);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref4);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref5);
        write(ptr,tmp);

    --                              Pe = 8
    elsif (Pe_in = "1000") then
        tmp := conv_integer(address0);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address1);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address2);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address3);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address4);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address5);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address6);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address7);
        write(ptr,tmp);
        write(ptr,space_txt);

        tmp := conv_integer(mem_ref0);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref1);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref2);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref3);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref4);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref5);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref6);
        write(ptr,tmp);
```

```
                  write(ptr,blank_txt);
                  tmp := conv_integer(mem_ref7);
                  write(ptr,tmp);

              end if;                   -- Pe's

              writeline(fileTXT,ptr);

            end if;                     -- DATA OUT

            wait for period;

          end loop;                     -- WHILE

          enable <= '0';
          clear <= '1';

          -------------------------------------------------------------------
          --  WRITE TO FILE INDICATOR CHANGE OF PARAMETERS: "***"
          write(ptr,change_txt);
          writeline(fileTXT,ptr);
          -------------------------------------------------------------------
          wait for 3*period;

        end if;                         -- IMPOSSIBLES CASES

      end loop;  -- i_N
    end loop;  -- i_Pe

    -- WRITING A FLAG TO END:
    write(ptr,end_txt);
    writeline(fileTXT,ptr);

    wait;

  END PROCESS;

END;
```

XIII. *tb_adxGen_SCRAMB.vhd*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library std;
use std.textio.all;

ENTITY tb_adxGen_SCRAMB_vhd IS
END tb_adxGen_SCRAMB_vhd;

ARCHITECTURE behavior OF tb_adxGen_SCRAMB_vhd IS

  -- Component Declaration for the Unit Under Test (UUT)
  COMPONENT addressGenerator
    PORT(
      clk : IN std_logic;
      enable : IN std_logic;
      clear : IN std_logic;
      rst_n : IN std_logic;
      scrambled : IN std_logic;
      N_in : IN std_logic_vector(11 downto 0);
      Pe_in : IN std_logic_vector(3 downto 0);
      validOut : OUT std_logic;
      mem_ref : OUT std_logic_vector(23 downto 0);
      address : OUT std_logic_vector(119 downto 0));
  END COMPONENT;

  --Inputs
  SIGNAL clk :  std_logic := '1';
  SIGNAL enable :  std_logic := '0';
  SIGNAL clear :  std_logic := '1';
  SIGNAL rst_n :  std_logic := '0';
  SIGNAL scrambled :  std_logic := '1';
  SIGNAL N_in :  std_logic_vector(11 downto 0) := (others=>'0');
  SIGNAL Pe_in :  std_logic_vector(3 downto 0) := (others=>'0');

  --Outputs
  SIGNAL validOut :  std_logic;
  SIGNAL mem_ref :  std_logic_vector(23 downto 0);
  SIGNAL address :  std_logic_vector(119 downto 0);

  --Auxiliar signals
  signal mem_ref0 :  std_logic_vector(2 downto 0);
  signal mem_ref1 :  std_logic_vector(2 downto 0);
  signal mem_ref2 :  std_logic_vector(2 downto 0);
  signal mem_ref3 :  std_logic_vector(2 downto 0);
  signal mem_ref4 :  std_logic_vector(2 downto 0);
  signal mem_ref5 :  std_logic_vector(2 downto 0);
  signal mem_ref6 :  std_logic_vector(2 downto 0);
  signal mem_ref7 :  std_logic_vector(2 downto 0);
  signal address0 :  std_logic_vector(14 downto 0);
  signal address1 :  std_logic_vector(14 downto 0);
  signal address2 :  std_logic_vector(14 downto 0);
  signal address3 :  std_logic_vector(14 downto 0);
  signal address4 :  std_logic_vector(14 downto 0);
  signal address5 :  std_logic_vector(14 downto 0);
  signal address6 :  std_logic_vector(14 downto 0);
  signal address7 :  std_logic_vector(14 downto 0);

  --Time constants
  constant period : time := 100 ns;
```

```
  --String constants
  constant N_txt : string := "N = ";
  constant Pe_txt : string := "Pe = ";
  constant blank_txt : string := " ";
  constant empty_txt : string := "";
  constant change_txt : string := "***";
  constant space_txt : string := " - ";
  constant C_char : string := " C";
  constant end_txt : string := "END";


  --Constant for using at the simulator
  type arrayPe is array (4 downto 0) of std_logic_vector(3 downto 0);
  type arrayN is array (16 downto 0) of std_logic_vector(11 downto 0);
  constant Pe : arrayPe := ("1000", "0110", "0100", "0010", "0001");
  constant N : arrayN := (conv_std_logic_vector(2400,12),
                         conv_std_logic_vector(1920,12),
                         conv_std_logic_vector(1440,12),
                         conv_std_logic_vector(960,12),
                         conv_std_logic_vector(480,12),
                         conv_std_logic_vector(240,12),
                         conv_std_logic_vector(216,12),
                         conv_std_logic_vector(192,12),
                         conv_std_logic_vector(180,12),
                         conv_std_logic_vector(144,12),
                         conv_std_logic_vector(120,12),
                         conv_std_logic_vector(108,12),
                         conv_std_logic_vector(96,12),
                         conv_std_logic_vector(72,12),
                         conv_std_logic_vector(48,12),
                         conv_std_logic_vector(36,12),
                         conv_std_logic_vector(24,12));

BEGIN

  -- Instantiate the Unit Under Test (UUT)
  uut: addressGenerator PORT MAP(
    clk => clk,
    enable => enable,
    clear => clear,
    rst_n => rst_n,
    scrambled => scrambled,
    N_in => N_in,
    Pe_in => Pe_in,
    validOut => validOut,
    mem_ref => mem_ref,
    address => address);

  -- cycling the clk
  clk <= NOT clk AFTER period/2;

  -- Separating "mem_ref" & "address"
  mem_ref0 <= mem_ref(2 downto 0);
  mem_ref1 <= mem_ref(5 downto 3);
  mem_ref2 <= mem_ref(8 downto 6);
  mem_ref3 <= mem_ref(11 downto 9);
  mem_ref4 <= mem_ref(14 downto 12);
  mem_ref5 <= mem_ref(17 downto 15);
  mem_ref6 <= mem_ref(20 downto 18);
  mem_ref7 <= mem_ref(23 downto 21);
  address0 <= address(14 downto 0);
  address1 <= address(29 downto 15);
  address2 <= address(44 downto 30);
  address3 <= address(59 downto 45);
```

```
address4 <= address(74 downto 60);
address5 <= address(89 downto 75);
address6 <= address(104 downto 90);
address7 <= address(119 downto 105);

tb : PROCESS

  file fileTXT : text open WRITE_MODE is "report_addressGen_SCRAMB_VHDL.txt";
  variable tmp : integer;
  variable ptr  : line;
  variable counter : integer := 0;

BEGIN

  for i_Pe in 0 to 4 loop
    Pe_in <= Pe(i_Pe);

    for i_N in 0 to 16 loop
      N_in <= N(i_N);
      wait for period;

      -- Excluding cases:
      if ( not( Pe_in = "1000" and (N_in = conv_std_logic_vector(36,12) or
                                    N_in = conv_std_logic_vector(108,12) or
                                    N_in = conv_std_logic_vector(180,12)) ) ) then
        -------------------------------------------------------------------
        --  WRITE TO FILE THE INFORMATION OF THE ACTUAL OPERATION
        tmp := conv_integer(N_in);
        write(ptr,N_txt);
        write(ptr,tmp);
        writeline(fileTXT,ptr);
        tmp := conv_integer(Pe_in);
        write(ptr,Pe_txt);
        write(ptr,tmp);
        writeline(fileTXT,ptr);
        -------------------------------------------------------------------

        -- Counter -> 0
        counter := 0;

        -- SIMULATION
        enable <= '0';
        clear<= '1';
        rst_n <= '0';
        wait for 2*period;
        clear <= '0';
        rst_n <= '1';
        wait for 2*period;

        while (counter < conv_integer(N_in)/conv_integer(Pe_in)) loop
          enable <= '1';

          if (validOut = '1') then

            counter := counter + 1;

            -- Output valid, so, WRITE TO FILE!!!!
            -------------------------------------------------------------
            --                              Pe = 1
            if (Pe_in = "0001") then
                tmp := conv_integer(address0);
                write(ptr,tmp);
                write(ptr,space_txt);

                tmp := conv_integer(mem_ref0);
```

```
    write(ptr,tmp);

--                              Pe = 2
elsif (Pe_in = "0010") then
    tmp := conv_integer(address0);
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(address1);
    write(ptr,tmp);
    write(ptr,space_txt);

    tmp := conv_integer(mem_ref0);
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(mem_ref1);
    write(ptr,tmp);

--                              Pe = 4
elsif (Pe_in = "0100") then
    tmp := conv_integer(address0);
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(address1);
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(address2);
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(address3);
    write(ptr,tmp);
    write(ptr,space_txt);

    tmp := conv_integer(mem_ref0);
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(mem_ref1);
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(mem_ref2);
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(mem_ref3);
    write(ptr,tmp);

--                              Pe = 6
elsif (Pe_in = "0110") then
    tmp := conv_integer(address0);
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(address1);
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(address2);
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(address3);
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(address4);
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(address5);
    write(ptr,tmp);
    write(ptr,space_txt);
```

```
        tmp := conv_integer(mem_ref0);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref1);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref2);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref3);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref4);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref5);
        write(ptr,tmp);

    --                              Pe = 8
    elsif (Pe_in = "1000") then
        tmp := conv_integer(address0);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address1);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address2);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address3);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address4);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address5);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address6);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(address7);
        write(ptr,tmp);
        write(ptr,space_txt);

        tmp := conv_integer(mem_ref0);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref1);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref2);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref3);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref4);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref5);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(mem_ref6);
        write(ptr,tmp);
```

```
                  write(ptr,blank_txt);
                  tmp := conv_integer(mem_ref7);
                  write(ptr,tmp);

              end if;                   -- Pe's

              writeline(fileTXT,ptr);

            end if;                     -- DATA OUT

            wait for period;

          end loop;                     -- WHILE

          enable <= '0';
          clear <= '1';

          ------------------------------------------------------------------
          --  WRITE TO FILE INDICATOR CHANGE OF PARAMETERS: "***"
          write(ptr,change_txt);
          writeline(fileTXT,ptr);
          ------------------------------------------------------------------
          wait for 3*period;

        end if;                         -- IMPOSSIBLES CASES

      end loop;  -- i_N
    end loop;  -- i_Pe

    -- WRITING A FLAG TO END:
    write(ptr,end_txt);
    writeline(fileTXT,ptr);

    wait;

  END PROCESS;

END;
```

XIV. *tb_SRM_ORDER.vhd*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library std;
use std.textio.all;

ENTITY tb_SRM_ORDER_vhd IS
END tb_SRM_ORDER_vhd;

ARCHITECTURE behavior OF tb_SRM_ORDER_vhd IS

  -- Component Declaration for the Unit Under Test (UUT)
  COMPONENT conectando
    PORT(
      clk : IN std_logic;
      enable : IN std_logic;
      clear : IN std_logic;
      rst_n : IN std_logic;
      N_in : IN std_logic_vector(11 downto 0);
      Pe_in : IN std_logic_vector(3 downto 0);
      scrambled : IN std_logic;
      DATAout : OUT std_logic_vector(255 downto 0);
      DATAoutVALID : OUT std_logic
      );
  END COMPONENT;

  --Inputs
  SIGNAL clk :  std_logic := '1';
  SIGNAL enable :  std_logic := '0';
  SIGNAL clear :  std_logic := '1';
  SIGNAL rst_n :  std_logic := '0';
  SIGNAL scrambled :  std_logic := '0';
  SIGNAL N_in :  std_logic_vector(11 downto 0) := (others=>'0');
  SIGNAL Pe_in :  std_logic_vector(3 downto 0) := (others=>'0');

  --Outputs
  SIGNAL DATAoutVALID :  std_logic;
  signal DATA_SISO_0 : std_logic_vector(31 downto 0);
  signal DATA_SISO_1 : std_logic_vector(31 downto 0);
  signal DATA_SISO_2 : std_logic_vector(31 downto 0);
  signal DATA_SISO_3 : std_logic_vector(31 downto 0);
  signal DATA_SISO_4 : std_logic_vector(31 downto 0);
  signal DATA_SISO_5 : std_logic_vector(31 downto 0);
  signal DATA_SISO_6 : std_logic_vector(31 downto 0);
  signal DATA_SISO_7 : std_logic_vector(31 downto 0);
  SIGNAL DATAout :  std_logic_vector(255 downto 0);

  --TIME
  constant period : time := 100 ns;

  --STRING CONSTANTS
  constant N_txt : string := "N = ";
  constant Pe_txt : string := "Pe = ";
  constant blank_txt : string := " ";
  constant empty_txt : string := "";
  constant change_txt : string := "***";
  constant space_txt : string := " - ";
  constant C_char : string := " C";
  constant end_txt : string := "END";

  --Constant for simulator
```

```
  type arrayPe is array (4 downto 0) of std_logic_vector(3 downto 0);
  type arrayN is array (16 downto 0) of std_logic_vector(11 downto 0);
  constant Pe : arrayPe := ("1000", "0110", "0100", "0010", "0001");
  constant N : arrayN := (conv_std_logic_vector(2400,12),
                          conv_std_logic_vector(1920,12),
                          conv_std_logic_vector(1440,12),
                          conv_std_logic_vector(960,12),
                          conv_std_logic_vector(480,12),
                          conv_std_logic_vector(240,12),
                          conv_std_logic_vector(216,12),
                          conv_std_logic_vector(192,12),
                          conv_std_logic_vector(180,12),
                          conv_std_logic_vector(144,12),
                          conv_std_logic_vector(120,12),
                          conv_std_logic_vector(108,12),
                          conv_std_logic_vector(96,12),
                          conv_std_logic_vector(72,12),
                          conv_std_logic_vector(48,12),
                          conv_std_logic_vector(36,12),
                          conv_std_logic_vector(24,12));

BEGIN

  -- Instantiate the Unit Under Test (UUT)
  uut: conectando PORT MAP(
    clk => clk,
    enable => enable,
    clear => clear,
    rst_n => rst_n,
    N_in => N_in,
    Pe_in => Pe_in,
    scrambled => scrambled,
    DATAout => DATAout,
    DATAoutVALID => DATAoutVALID
    );

  clk <= not clk after period/2;

  --Separating DATA OUT
  DATA_SISO_0 <= DATAout(31 downto 0);
  DATA_SISO_1 <= DATAout(63 downto 32);
  DATA_SISO_2 <= DATAout(95 downto 64);
  DATA_SISO_3 <= DATAout(127 downto 96);
  DATA_SISO_4 <= DATAout(159 downto 128);
  DATA_SISO_5 <= DATAout(191 downto 160);
  DATA_SISO_6 <= DATAout(223 downto 192);
  DATA_SISO_7 <= DATAout(255 downto 224);

  tb : process

    file fileTXT : text open WRITE_MODE is "report_SRM_ORDER_VHDL.txt";
    variable tmp : integer;
    variable ptr  : line;
    variable counter : integer := 0;

  begin


    for i_Pe in 0 to 4 loop
      Pe_in <= Pe(i_Pe);

      for i_N in 0 to 16 loop
        N_in <= N(i_N);
        wait for period;
```

```
-- Excluding cases:
if ( not( Pe_in = "1000" and (N_in = conv_std_logic_vector(36,12) or
                              N_in = conv_std_logic_vector(108,12) or
                              N_in = conv_std_logic_vector(180,12)) ) ) then
  -------------------------------------------------------------------
  --  WRITE TO FILE THE INFORMATION OF THE ACTUAL OPERATION
  tmp := conv_integer(N_in);
  write(ptr,N_txt);
  write(ptr,tmp);
  writeline(fileTXT,ptr);
  tmp := conv_integer(Pe_in);
  write(ptr,Pe_txt);
  write(ptr,tmp);
  writeline(fileTXT,ptr);
  -------------------------------------------------------------------

  -- Counter -> 0
  counter := 0;

  -- SIMULATION
  enable <= '0';
  clear<= '1';
  rst_n <= '0';
  wait for 2*period;
  clear <= '0';
  rst_n <= '1';
  wait for 2*period;

  while (counter < conv_integer(N_in)/conv_integer(Pe_in)) loop
    enable <= '1';

    if (DATAoutVALID = '1') then

      counter := counter + 1;

      -- Output valid, so, WRITE TO FILE!!!!
      ---------------------------------------------------------------
      --                              Pe = 1
      if (Pe_in = "0001") then
        tmp := conv_integer(DATA_SISO_0);
        write(ptr,tmp);

      --                              Pe = 2
      elsif (Pe_in = "0010") then
        tmp := conv_integer(DATA_SISO_0);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(DATA_SISO_1);
        write(ptr,tmp);

      --                              Pe = 4
      elsif (Pe_in = "0100") then
        tmp := conv_integer(DATA_SISO_0);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(DATA_SISO_1);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(DATA_SISO_2);
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(DATA_SISO_3);
        write(ptr,tmp);

      --                              Pe = 6
```

```
    elsif (Pe_in = "0110") then
      tmp := conv_integer(DATA_SISO_0);
      write(ptr,tmp);
      write(ptr,blank_txt);
      tmp := conv_integer(DATA_SISO_1);
      write(ptr,tmp);
      write(ptr,blank_txt);
      tmp := conv_integer(DATA_SISO_2);
      write(ptr,tmp);
      write(ptr,blank_txt);
      tmp := conv_integer(DATA_SISO_3);
      write(ptr,tmp);
      write(ptr,blank_txt);
      tmp := conv_integer(DATA_SISO_4);
      write(ptr,tmp);
      write(ptr,blank_txt);
      tmp := conv_integer(DATA_SISO_5);
      write(ptr,tmp);

    --                              Pe = 8
    elsif (Pe_in = "1000") then
      tmp := conv_integer(DATA_SISO_0);
      write(ptr,tmp);
      write(ptr,blank_txt);
      tmp := conv_integer(DATA_SISO_1);
      write(ptr,tmp);
      write(ptr,blank_txt);
      tmp := conv_integer(DATA_SISO_2);
      write(ptr,tmp);
      write(ptr,blank_txt);
      tmp := conv_integer(DATA_SISO_3);
      write(ptr,tmp);
      write(ptr,blank_txt);
      tmp := conv_integer(DATA_SISO_4);
      write(ptr,tmp);
      write(ptr,blank_txt);
      tmp := conv_integer(DATA_SISO_5);
      write(ptr,tmp);
      write(ptr,blank_txt);
      tmp := conv_integer(DATA_SISO_6);
      write(ptr,tmp);
      write(ptr,blank_txt);
      tmp := conv_integer(DATA_SISO_7);
      write(ptr,tmp);
    end if;                   -- Pe's

    writeline(fileTXT,ptr);

  end if;                     -- DATA OUT

  wait for period;

end loop;                     -- WHILE

enable <= '0';
clear <= '1';

-------------------------------------------------------------------
--  WRITE TO FILE INDICATOR CHANGE OF PARAMETERS: "***"
write(ptr,change_txt);
writeline(fileTXT,ptr);
-------------------------------------------------------------------

wait for 3*period;
```

```
        end if;                         -- IMPOSSIBLES CASES

      end loop;  -- i_N
    end loop;  -- i_Pe

    -- WRITING A FLAG TO END:
    write(ptr,end_txt);
    writeline(fileTXT,ptr);

    wait;

  END PROCESS;

END;


-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity mem_simulation is

  generic (
    DATA_SIZE    : positive := 32;
    ADDRESS_SIZE : positive := 12);

  port (
    clk      : in  std_logic;
    enable   : in  std_logic;
    offset   : in  integer;
    address  : in  std_logic_vector(ADDRESS_SIZE-1 downto 0);
    data_out : out std_logic_vector(DATA_SIZE-1 downto 0));

end mem_simulation;


architecture behav_mem_simulation of mem_simulation is

  type memContent_type is array (0 to (2**ADDRESS_SIZE)-1) of std_logic_vector(DATA_SIZE-1 downto 0);
  signal memoryContent : memContent_type;

begin  -- behav_mem_simulation

  loadingDataROM: for i in 0 to (2**ADDRESS_SIZE)-1 generate
    memoryContent(i) <= conv_std_logic_vector(i+(offset*10000),DATA_SIZE);
  end generate loadingDataROM;


  unico: process (clk)
  begin  -- process unico

    if (clk = '1' and clk'event) then
      if (enable = '1') then
        data_out <= memoryContent(conv_integer(address));
      end if;
    end if;

  end process unico;
```

```
end behav_mem_simulation;


------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------



library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity conectando is

  generic (
    DATA_SIZE    : positive := 32;
    ADDRESS_SIZE : positive := 12);

  port (
    clk          : in  std_logic;
    enable       : in  std_logic;
    clear        : in  std_logic;
    rst_n        : in  std_logic;
    N_in         : in  std_logic_vector(11 downto 0);
    Pe_in        : in  std_logic_vector(3 downto 0);
    scrambled    : in  std_logic;
    DATAout      : out std_logic_vector(8*DATA_SIZE-1 downto 0);
    DATAoutVALID : out std_logic);

end conectando;



architecture behav_conectando of conectando is

  -- COMPONENTS:
  component mem_simulation
    generic (
      DATA_SIZE    : positive;
      ADDRESS_SIZE : positive);
    port (
      clk      : in  std_logic;
      enable   : in  std_logic;
      offset   : in  integer;
      address  : in  std_logic_vector(ADDRESS_SIZE-1 downto 0);
      data_out : out std_logic_vector(DATA_SIZE-1 downto 0));
  end component;

  component switchReadMemory
    generic (
      DATA_SIZE    : positive;
      ADDRESS_SIZE : positive);
    port (
      clk          : in  std_logic;
      clear        : in  std_logic;
      rst_n        : in  std_logic;
      Pe_in        : in  std_logic_vector(3 downto 0);
      address_in   : in  std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
      memRef_in    : in  std_logic_vector(23 downto 0);
      input_valid  : in  std_logic;
```

```
      MEM_RDATA    : in  std_logic_vector(8*DATA_SIZE-1 downto 0);
      MEM_RADX     : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
      MEM_REN      : out std_logic_vector(7 downto 0);
      RDATA        : out std_logic_vector(8*DATA_SIZE-1 downto 0);
      RDATA_VALID  : out std_logic);
  end component;

  component addressGenerator
    generic (
      ADDRESS_SIZE : positive);
    port (
      clk       : in  std_logic;
      enable    : in  std_logic;
      clear     : in  std_logic;
      rst_n     : in  std_logic;
      scrambled : in  std_logic;
      N_in      : in  std_logic_vector(11 downto 0);
      Pe_in     : in  std_logic_vector(3 downto 0);
      validOut  : out std_logic;
      mem_ref   : out std_logic_vector(23 downto 0);
      address   : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0));
  end component;


  -- SIGNALS
  signal enableMEM : std_logic_vector(7 downto 0);
  signal MEM_DATA : std_logic_vector(8*DATA_SIZE-1 downto 0);
  signal MEM_ADX : std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
  signal dataVALID : std_logic;
  signal mem_ref : std_logic_vector(23 downto 0);
  signal address : std_logic_vector(8*ADDRESS_SIZE-1 downto 0);


begin  -- behav_conectando


  -- MEMORIES:
  memories: for i in 0 to 7 generate
    memory : mem_simulation
      generic map (
        DATA_SIZE    => DATA_SIZE,
        ADDRESS_SIZE => ADDRESS_SIZE)
      port map (
        clk      => clk,
        enable   => enableMEM(i),
        offset   => i,
        address  => MEM_ADX((i+1)*ADDRESS_SIZE-1 downto ADDRESS_SIZE*i),
        data_out => MEM_DATA((i+1)*DATA_SIZE-1 downto DATA_SIZE*i));
  end generate memories;


  -- ADDRESS GENERATOR
  addGen : addressGenerator
    generic map (
      ADDRESS_SIZE => ADDRESS_SIZE)
    port map (
      clk       => clk,
      enable    => enable,
      clear     => clear,
      rst_n     => rst_n,
      scrambled => scrambled,
      N_in      => N_in,
      Pe_in     => Pe_in,
      validOut  => dataVALID,
      mem_ref   => mem_ref,
```

```
      address   => address);

  -- SWITCH READ MEMORY
  SRM : switchReadMemory
    generic map (
      DATA_SIZE    => DATA_SIZE,
      ADDRESS_SIZE => ADDRESS_SIZE)
    port map (
      clk          => clk,
      clear        => clear,
      rst_n        => rst_n,
      Pe_in        => Pe_in,
      address_in   => address,
      memRef_in    => mem_ref,
      input_valid  => dataVALID,
      MEM_RDATA    => MEM_DATA,
      MEM_RADX     => MEM_ADX,
      MEM_REN      => enableMEM,
      RDATA        => DATAout,
      RDATA_VALID  => DATAoutVALID);

end behav_conectando;
```

XV. *tb_SRM_SCRAMB.vhd*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library std;
use std.textio.all;

ENTITY tb_SRM_SCRAMB_vhd IS
END tb_SRM_SCRAMB_vhd;

ARCHITECTURE behavior OF tb_SRM_SCRAMB_vhd IS

  -- Component Declaration for the Unit Under Test (UUT)
  COMPONENT conectando
    PORT(
      clk : IN std_logic;
      enable : IN std_logic;
      clear : IN std_logic;
      rst_n : IN std_logic;
      N_in : IN std_logic_vector(11 downto 0);
      Pe_in : IN std_logic_vector(3 downto 0);
      scrambled : IN std_logic;
      DATAout : OUT std_logic_vector(255 downto 0);
      DATAoutVALID : OUT std_logic
      );
  END COMPONENT;

  --Inputs
  SIGNAL clk :  std_logic := '1';
  SIGNAL enable :  std_logic := '0';
  SIGNAL clear :  std_logic := '1';
  SIGNAL rst_n :  std_logic := '0';
  SIGNAL scrambled :  std_logic := '1';
  SIGNAL N_in :  std_logic_vector(11 downto 0) := (others=>'0');
  SIGNAL Pe_in :  std_logic_vector(3 downto 0) := (others=>'0');

  --Outputs
  SIGNAL DATAoutVALID :  std_logic;
  signal DATA_SISO_0 : std_logic_vector(31 downto 0);
  signal DATA_SISO_1 : std_logic_vector(31 downto 0);
  signal DATA_SISO_2 : std_logic_vector(31 downto 0);
  signal DATA_SISO_3 : std_logic_vector(31 downto 0);
  signal DATA_SISO_4 : std_logic_vector(31 downto 0);
  signal DATA_SISO_5 : std_logic_vector(31 downto 0);
  signal DATA_SISO_6 : std_logic_vector(31 downto 0);
  signal DATA_SISO_7 : std_logic_vector(31 downto 0);
  SIGNAL DATAout :  std_logic_vector(255 downto 0);

  --TIME
  constant period : time := 100 ns;

  --STRING CONSTANTS
  constant N_txt : string := "N = ";
  constant Pe_txt : string := "Pe = ";
  constant blank_txt : string := " ";
  constant empty_txt : string := "";
  constant change_txt : string := "***";
  constant space_txt : string := " - ";
  constant C_char : string := " C";
  constant end_txt : string := "END";

  --Constant for simulator
```

```
  type arrayPe is array (4 downto 0) of std_logic_vector(3 downto 0);
  type arrayN is array (16 downto 0) of std_logic_vector(11 downto 0);
  constant Pe : arrayPe := ("1000", "0110", "0100", "0010", "0001");
  constant N : arrayN := (conv_std_logic_vector(2400,12),
                          conv_std_logic_vector(1920,12),
                          conv_std_logic_vector(1440,12),
                          conv_std_logic_vector(960,12),
                          conv_std_logic_vector(480,12),
                          conv_std_logic_vector(240,12),
                          conv_std_logic_vector(216,12),
                          conv_std_logic_vector(192,12),
                          conv_std_logic_vector(180,12),
                          conv_std_logic_vector(144,12),
                          conv_std_logic_vector(120,12),
                          conv_std_logic_vector(108,12),
                          conv_std_logic_vector(96,12),
                          conv_std_logic_vector(72,12),
                          conv_std_logic_vector(48,12),
                          conv_std_logic_vector(36,12),
                          conv_std_logic_vector(24,12));

  --Enable when there is a collision
  signal enableCollision : std_logic := '1';

BEGIN

  -- Instantiate the Unit Under Test (UUT)
  uut: conectando PORT MAP(
    clk => clk,
    enable => enable,
    clear => clear,
    rst_n => rst_n,
    N_in => N_in,
    Pe_in => Pe_in,
    scrambled => scrambled,
    DATAout => DATAout,
    DATAoutVALID => DATAoutVALID
    );

  clk <= not clk after period/2;
  enableCollision <= not enableCollision after period;

  --Separating DATA OUT
  DATA_SISO_0 <= DATAout(31 downto 0);
  DATA_SISO_1 <= DATAout(63 downto 32);
  DATA_SISO_2 <= DATAout(95 downto 64);
  DATA_SISO_3 <= DATAout(127 downto 96);
  DATA_SISO_4 <= DATAout(159 downto 128);
  DATA_SISO_5 <= DATAout(191 downto 160);
  DATA_SISO_6 <= DATAout(223 downto 192);
  DATA_SISO_7 <= DATAout(255 downto 224);

  tb : process

    file fileTXT : text open WRITE_MODE is "report_SRM_SCRAMB_VHDL.txt";
    variable tmp : integer;
    variable ptr  : line;
    variable counter : integer := 0;

  begin


    for i_Pe in 0 to 4 loop
      Pe_in <= Pe(i_Pe);
```

```
for i_N in 0 to 16 loop
  N_in <= N(i_N);
  wait for period;

  -- Excluding cases:
  if ( not( Pe_in = "1000" and (N_in = conv_std_logic_vector(36,12) or
                                N_in = conv_std_logic_vector(108,12) or
                                N_in = conv_std_logic_vector(180,12)) ) ) then
    -------------------------------------------------------------------
    --  WRITE TO FILE THE INFORMATION OF THE ACTUAL OPERATION
    tmp := conv_integer(N_in);
    write(ptr,N_txt);
    write(ptr,tmp);
    writeline(fileTXT,ptr);
    tmp := conv_integer(Pe_in);
    write(ptr,Pe_txt);
    write(ptr,tmp);
    writeline(fileTXT,ptr);
    -------------------------------------------------------------------

    ---------------------------------------------------------------------
    -- CASES WITH COLLISION:
    --
    --   N = 108       Pe = 2
    --   N = 108       Pe = 4
    --   N = 108       Pe = 6
    --   N = 72        Pe = 8
    ---------------------------------------------------------------------

    -- Counter -> 0
    counter := 0;

    -- SIMULATION
    enable <= '0';
    clear<= '1';
    rst_n <= '0';
    wait for 2*period;
    clear <= '0';
    rst_n <= '1';
    wait for 2*period;

    while (counter < conv_integer(N_in)/conv_integer(Pe_in)) loop

      if ((conv_integer(N_in) = 108 and conv_integer(Pe_in) = 2) or
          (conv_integer(N_in) = 108 and conv_integer(Pe_in) = 4) or
          (conv_integer(N_in) = 108 and conv_integer(Pe_in) = 6) or
          (conv_integer(N_in) = 72 and conv_integer(Pe_in) = 8)) then

        enable <= enableCollision;
      else
        enable <= '1';
      end if;

      if (DATAoutVALID = '1') then

        counter := counter + 1;

        -- Output valid, so, WRITE TO FILE!!!!
        ---------------------------------------------------------------
        --                              Pe = 1
        if (Pe_in = "0001") then
          tmp := conv_integer(DATA_SISO_0);
          write(ptr,tmp);

        --                              Pe = 2
```

```
elsif (Pe_in = "0010") then
  tmp := conv_integer(DATA_SISO_0);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(DATA_SISO_1);
  write(ptr,tmp);

--                              Pe = 4
elsif (Pe_in = "0100") then
  tmp := conv_integer(DATA_SISO_0);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(DATA_SISO_1);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(DATA_SISO_2);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(DATA_SISO_3);
  write(ptr,tmp);

--                              Pe = 6
elsif (Pe_in = "0110") then
  tmp := conv_integer(DATA_SISO_0);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(DATA_SISO_1);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(DATA_SISO_2);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(DATA_SISO_3);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(DATA_SISO_4);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(DATA_SISO_5);
  write(ptr,tmp);

--                              Pe = 8
elsif (Pe_in = "1000") then
  tmp := conv_integer(DATA_SISO_0);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(DATA_SISO_1);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(DATA_SISO_2);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(DATA_SISO_3);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(DATA_SISO_4);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(DATA_SISO_5);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(DATA_SISO_6);
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(DATA_SISO_7);
```

```
                write(ptr,tmp);
              end if;                    -- Pe's

              writeline(fileTXT,ptr);

            end if;                      -- DATA OUT

            wait for period;

          end loop;                      -- WHILE

          enable <= '0';
          clear <= '1';

          -------------------------------------------------------------------
          --  WRITE TO FILE INDICATOR CHANGE OF PARAMETERS: "***"
          write(ptr,change_txt);
          writeline(fileTXT,ptr);
          -------------------------------------------------------------------
          wait for 3*period;

        end if;                          -- IMPOSSIBLES CASES

      end loop;  -- i_N
    end loop;  -- i_Pe

    -- WRITING A FLAG TO END:
    write(ptr,end_txt);
    writeline(fileTXT,ptr);

    wait;

  end process;

end;


-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity mem_simulation is

  generic (
    DATA_SIZE    : positive := 32;
    ADDRESS_SIZE : positive := 12);

  port (
    clk      : in  std_logic;
    enable   : in  std_logic;
    offset   : in  integer;
    address  : in  std_logic_vector(ADDRESS_SIZE-1 downto 0);
    data_out : out std_logic_vector(DATA_SIZE-1 downto 0));

end mem_simulation;


architecture behav_mem_simulation of mem_simulation is
```

```
  type memContent_type is array (0 to (2**ADDRESS_SIZE)-1) of std_logic_vector(DATA_SIZE-1 downto 0);
  signal memoryContent : memContent_type;

begin  -- behav_mem_simulation

  loadingDataROM: for i in 0 to (2**ADDRESS_SIZE)-1 generate
    memoryContent(i) <= conv_std_logic_vector(i+(offset*10000),DATA_SIZE);
  end generate loadingDataROM;


  unico: process (clk)
  begin  -- process unico

    if (clk = '1' and clk'event) then
      if (enable = '1') then
        data_out <= memoryContent(conv_integer(address));
      end if;
    end if;

  end process unico;



end behav_mem_simulation;


-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------



library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity conectando is

  generic (
    DATA_SIZE    : positive := 32;
    ADDRESS_SIZE : positive := 12);

  port (
    clk          : in  std_logic;
    enable       : in  std_logic;
    clear        : in  std_logic;
    rst_n        : in  std_logic;
    N_in         : in  std_logic_vector(11 downto 0);
    Pe_in        : in  std_logic_vector(3 downto 0);
    scrambled    : in  std_logic;
    DATAout      : out std_logic_vector(8*DATA_SIZE-1 downto 0);
    DATAoutVALID : out std_logic);

end conectando;


architecture behav_conectando of conectando is

  -- COMPONENTS:
  component mem_simulation
    generic (
      DATA_SIZE    : positive;
      ADDRESS_SIZE : positive);
```

```
    port (
      clk       : in  std_logic;
      enable    : in  std_logic;
      offset    : in  integer;
      address   : in  std_logic_vector(ADDRESS_SIZE-1 downto 0);
      data_out  : out std_logic_vector(DATA_SIZE-1 downto 0));
  end component;

  component switchReadMemory
    generic (
      DATA_SIZE    : positive;
      ADDRESS_SIZE : positive);
    port (
      clk          : in  std_logic;
      clear        : in  std_logic;
      rst_n        : in  std_logic;
      Pe_in        : in  std_logic_vector(3 downto 0);
      address_in   : in  std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
      memRef_in    : in  std_logic_vector(23 downto 0);
      input_valid  : in  std_logic;
      MEM_RDATA    : in  std_logic_vector(8*DATA_SIZE-1 downto 0);
      MEM_RADX     : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
      MEM_REN      : out std_logic_vector(7 downto 0);
      RDATA        : out std_logic_vector(8*DATA_SIZE-1 downto 0);
      RDATA_VALID  : out std_logic);
  end component;

  component addressGenerator
    generic (
      ADDRESS_SIZE : positive);
    port (
      clk       : in  std_logic;
      enable    : in  std_logic;
      clear     : in  std_logic;
      rst_n     : in  std_logic;
      scrambled : in  std_logic;
      N_in      : in  std_logic_vector(11 downto 0);
      Pe_in     : in  std_logic_vector(3 downto 0);
      validOut  : out std_logic;
      mem_ref   : out std_logic_vector(23 downto 0);
      address   : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0));
  end component;


  -- SIGNALS
  signal enableMEM : std_logic_vector(7 downto 0);
  signal MEM_DATA : std_logic_vector(8*DATA_SIZE-1 downto 0);
  signal MEM_ADX : std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
  signal dataVALID : std_logic;
  signal mem_ref : std_logic_vector(23 downto 0);
  signal address : std_logic_vector(8*ADDRESS_SIZE-1 downto 0);


begin  -- behav_conectando


  -- MEMORIES:
  memories: for i in 0 to 7 generate
    memory : mem_simulation
      generic map (
        DATA_SIZE    => DATA_SIZE,
        ADDRESS_SIZE => ADDRESS_SIZE)
      port map (
        clk      => clk,
        enable   => enableMEM(i),
```

```
        offset   => i,
        address  => MEM_ADX((i+1)*ADDRESS_SIZE-1 downto ADDRESS_SIZE*i),
        data_out => MEM_DATA((i+1)*DATA_SIZE-1 downto DATA_SIZE*i));
  end generate memories;


  -- ADDRESS GENERATOR
  addGen : addressGenerator
    generic map (
      ADDRESS_SIZE => ADDRESS_SIZE)
    port map (
      clk       => clk,
      enable    => enable,
      clear     => clear,
      rst_n     => rst_n,
      scrambled => scrambled,
      N_in      => N_in,
      Pe_in     => Pe_in,
      validOut  => dataVALID,
      mem_ref   => mem_ref,
      address   => address);

  -- SWITCH READ MEMORY
  SRM : switchReadMemory
    generic map (
      DATA_SIZE    => DATA_SIZE,
      ADDRESS_SIZE => ADDRESS_SIZE)
    port map (
      clk          => clk,
      clear        => clear,
      rst_n        => rst_n,
      Pe_in        => Pe_in,
      address_in   => address,
      memRef_in    => mem_ref,
      input_valid  => dataVALID,
      MEM_RDATA    => MEM_DATA,
      MEM_RADX     => MEM_ADX,
      MEM_REN      => enableMEM,
      RDATA        => DATAout,
      RDATA_VALID  => DATAoutVALID);

end behav_conectando;
```

XVI. *tb_SWM_ORDER.vhd*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library std;
use std.textio.all;

ENTITY tb_SWM_ORDER_vhd IS
END tb_SWM_ORDER_vhd;

ARCHITECTURE behavior OF tb_SWM_ORDER_vhd IS

  -- Component Declaration for the Unit Under Test (UUT)
  COMPONENT conectando
    PORT(
      clk : IN std_logic;
      enable : IN std_logic;
      clear : IN std_logic;
      rst_n : IN std_logic;
      N_in : IN std_logic_vector(11 downto 0);
      Pe_in : IN std_logic_vector(3 downto 0);
      scrambled : IN std_logic;
      dataSISO : IN std_logic_vector(255 downto 0);
      dataSISOvalid : IN std_logic;
      MEM_WADX : OUT std_logic_vector(95 downto 0);
      MEM_WDATA : OUT std_logic_vector(255 downto 0);
      MEM_WEN : OUT std_logic_vector(7 downto 0)
      );
  END COMPONENT;

  --Inputs
  SIGNAL clk :  std_logic := '1';
  SIGNAL enable :  std_logic := '0';
  SIGNAL clear :  std_logic := '1';
  SIGNAL rst_n :  std_logic := '0';
  SIGNAL scrambled :  std_logic := '0';
  SIGNAL dataSISOvalid :  std_logic := '0';
  SIGNAL N_in :  std_logic_vector(11 downto 0) := (others=>'0');
  SIGNAL Pe_in :  std_logic_vector(3 downto 0) := (others=>'0');
  SIGNAL dataSISO :  std_logic_vector(255 downto 0) := (others=>'0');

  --Outputs
  SIGNAL MEM_WADX :  std_logic_vector(95 downto 0);
  SIGNAL MEM_WDATA :  std_logic_vector(255 downto 0);
  SIGNAL MEM_WEN :  std_logic_vector(7 downto 0);

  --TIME
  constant period : time := 100 ns;


  ----------------------------------------------------------------------------
  --    MEMORY
  type memoryTYPE is array (2499 downto 0) of std_logic_vector(31 downto 0);
  signal mem_0 : memoryTYPE := (others => (others => '0'));
  signal mem_1 : memoryTYPE := (others => (others => '0'));
  signal mem_2 : memoryTYPE := (others => (others => '0'));
  signal mem_3 : memoryTYPE := (others => (others => '0'));
  signal mem_4 : memoryTYPE := (others => (others => '0'));
  signal mem_5 : memoryTYPE := (others => (others => '0'));
  signal mem_6 : memoryTYPE := (others => (others => '0'));
  signal mem_7 : memoryTYPE := (others => (others => '0'));
  type memoryADX is array (7 downto 0) of std_logic_vector(11 downto 0);
```

```
  type memoryDATA is array (7 downto 0) of std_logic_vector(31 downto 0);
  signal memADX : memoryADX := (others => (others => '0'));
  signal memDATA : memoryDATA := (others => (others => '0'));
  ----------------------------------------------------------------------------



  ----------------------------------------------------------------------------
  --    SISO
  type sisoTYPE is array (7 downto 0) of std_logic_vector(31 downto 0);
  signal SISO : sisoTYPE := (others => (others => '0'));
  ----------------------------------------------------------------------------



  --STRING CONSTANTS
  constant N_txt : string := "N = ";
  constant Pe_txt : string := "Pe = ";
  constant blank_txt : string := " ";
  constant empty_txt : string := "";
  constant change_txt : string := "***";
  constant space_txt : string := " - ";
  constant C_char : string := " C";
  constant end_txt : string := "END";

  --Constant for using at the simulator
  type arrayPe is array (4 downto 0) of std_logic_vector(3 downto 0);
  type arrayN is array (16 downto 0) of std_logic_vector(11 downto 0);
  constant Pe : arrayPe := ("1000", "0110", "0100", "0010", "0001");
  constant N : arrayN := (conv_std_logic_vector(2400,12),
                          conv_std_logic_vector(1920,12),
                          conv_std_logic_vector(1440,12),
                          conv_std_logic_vector(960,12),
                          conv_std_logic_vector(480,12),
                          conv_std_logic_vector(240,12),
                          conv_std_logic_vector(216,12),
                          conv_std_logic_vector(192,12),
                          conv_std_logic_vector(180,12),
                          conv_std_logic_vector(144,12),
                          conv_std_logic_vector(120,12),
                          conv_std_logic_vector(108,12),
                          conv_std_logic_vector(96,12),
                          conv_std_logic_vector(72,12),
                          conv_std_logic_vector(48,12),
                          conv_std_logic_vector(36,12),
                          conv_std_logic_vector(24,12));


BEGIN

  -- Instantiate the Unit Under Test (UUT)
  uut: conectando PORT MAP(
    clk => clk,
    enable => enable,
    clear => clear,
    rst_n => rst_n,
    N_in => N_in,
    Pe_in => Pe_in,
    scrambled => scrambled,
    dataSISO => dataSISO,
    dataSISOvalid => dataSISOvalid,
    MEM_WADX => MEM_WADX,
    MEM_WDATA => MEM_WDATA,
    MEM_WEN => MEM_WEN);

  clk <= not clk after period/2;
```

```
-----------------------------------------------------------------------------
--    SISO
sisoLOOP: for i in 7 downto 0 generate
  dataSISO(32*(i+1)-1 downto 32*i) <= SISO(i);
end generate sisoLOOP;
-----------------------------------------------------------------------------

-----------------------------------------------------------------------------
--    MEMORY
mem: for i in 7 downto 0 generate
  memADX(i) <= MEM_WADX(12*(i+1)-1 downto 12*i);
  memDATA(i) <= MEM_WDATA(32*(i+1)-1 downto 32*i);
end generate mem;
-----------------------------------------------------------------------------

tb : PROCESS

  file fileTXT : text open WRITE_MODE is "reportSWM_ORDER_VHDL.txt";
  variable tmp : integer;
  variable ptr  : line;
  variable counter : integer := 0;

BEGIN

  for i_Pe in 0 to 4 loop
    Pe_in <= Pe(i_Pe);

    for i_N in 0 to 16 loop
      N_in <= N(i_N);
      wait for period;

      -- Excluding cases:
      if ( not( Pe_in = "1000" and (N_in = conv_std_logic_vector(36,12) or
                                    N_in = conv_std_logic_vector(108,12) or
                                    N_in = conv_std_logic_vector(180,12)) ) ) then

        ------------------------------------------------------------------
        --  WRITE TO FILE THE INFORMATION OF THE ACTUAL OPERATION
        tmp := conv_integer(N_in);
        write(ptr,N_txt);
        write(ptr,tmp);
        writeline(fileTXT,ptr);
        tmp := conv_integer(Pe_in);
        write(ptr,Pe_txt);
        write(ptr,tmp);
        writeline(fileTXT,ptr);
        ------------------------------------------------------------------

        --------------------------------------------------------------------
        for jj in 7 downto 0 loop
          SISO(jj) <= conv_std_logic_vector((jj+1)*10000,32);
        end loop;  -- jj
        --------------------------------------------------------------------

        -- Counter -> 0
        counter := 0;

        -- SIMULATION
        enable <= '0';
        clear<= '1';
        rst_n <= '0';
        wait for 2*period;
        clear <= '0';
        rst_n <= '1';
        wait for 2*period;
```

```
-- Activation of the interleaver (N/Pe + 1  periods)
enable <= '1';
wait for (1+(conv_integer(N_in)/conv_integer(Pe_in)))*period;
enable <= '0';

wait for 5*period;

-- Activation of SISO valid
while (counter < conv_integer(N_in)/conv_integer(Pe_in)) loop
  dataSISOvalid <= '1';

  counter := counter + 1;

  -------------------------------------------------------------------
  -- SCRITURA CUANDO SALIDA DE LAS MEMORIAS SEAN VALIDAS xDDD
  if (MEM_WEN(0) = '1') then  -- Writing at "MEM 0"
    mem_0(conv_integer(memADX(0))) <= memDATA(0);
  end if;

  if (MEM_WEN(1) = '1') then  -- Writing at "MEM 1"
    mem_1(conv_integer(memADX(1))) <= memDATA(1);
  end if;

  if (MEM_WEN(2) = '1') then  -- Writing at "MEM 2"
    mem_2(conv_integer(memADX(2))) <= memDATA(2);
  end if;

  if (MEM_WEN(3) = '1') then  -- Writing at "MEM 3"
    mem_3(conv_integer(memADX(3))) <= memDATA(3);
  end if;

  if (MEM_WEN(4) = '1') then  -- Writing at "MEM 4"
    mem_4(conv_integer(memADX(4))) <= memDATA(4);
  end if;

  if (MEM_WEN(5) = '1') then  -- Writing at "MEM 5"
    mem_5(conv_integer(memADX(5))) <= memDATA(5);
  end if;

  if (MEM_WEN(6) = '1') then  -- Writing at "MEM 6"
    mem_6(conv_integer(memADX(6))) <= memDATA(6);
  end if;

  if (MEM_WEN(7) = '1') then  -- Writing at "MEM 7"
    mem_7(conv_integer(memADX(7))) <= memDATA(7);
  end if;
  -------------------------------------------------------------------

  wait for period;

  for jj in 7 downto 0 loop
    SISO(jj) <= SISO(jj) + 1;
  end loop;  -- jj

end loop;  -- WHILE

dataSISOvalid <= '0';

for jj in 20 downto 0 loop
  -------------------------------------------------------------------
  -- SCRITURA CUANDO SALIDA DE LAS MEMORIAS SEAN VALIDAS xDDD
  if (MEM_WEN(0) = '1') then  -- Writing at "MEM 0"
    mem_0(conv_integer(memADX(0))) <= memDATA(0);
```

```
    end if;

    if (MEM_WEN(1) = '1') then  -- Writing at "MEM 1"
      mem_1(conv_integer(memADX(1))) <= memDATA(1);
    end if;

    if (MEM_WEN(2) = '1') then  -- Writing at "MEM 2"
      mem_2(conv_integer(memADX(2))) <= memDATA(2);
    end if;

    if (MEM_WEN(3) = '1') then  -- Writing at "MEM 3"
      mem_3(conv_integer(memADX(3))) <= memDATA(3);
    end if;

    if (MEM_WEN(4) = '1') then  -- Writing at "MEM 4"
      mem_4(conv_integer(memADX(4))) <= memDATA(4);
    end if;

    if (MEM_WEN(5) = '1') then  -- Writing at "MEM 5"
      mem_5(conv_integer(memADX(5))) <= memDATA(5);
    end if;

    if (MEM_WEN(6) = '1') then  -- Writing at "MEM 6"
      mem_6(conv_integer(memADX(6))) <= memDATA(6);
    end if;

    if (MEM_WEN(7) = '1') then  -- Writing at "MEM 7"
      mem_7(conv_integer(memADX(7))) <= memDATA(7);
    end if;
    -------------------------------------------------------------------

    wait for period;
  end loop;  -- jj


  -- REPORTING:
  -- MEM_0  MEM_1  MEM_2  MEM_3  MEM_4  MEM_5  MEM_6  MEM_7
  for jj in 0 to 2499 loop
    tmp := conv_integer(mem_0(jj));
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(mem_1(jj));
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(mem_2(jj));
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(mem_3(jj));
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(mem_4(jj));
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(mem_5(jj));
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(mem_6(jj));
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(mem_7(jj));
    write(ptr,tmp);

    writeline(fileTXT,ptr);

  end loop;  -- jj
```

```
          -- CLEAR MEMORIES:
          mem_0 <= (others => (others => '0'));
          mem_1 <= (others => (others => '0'));
          mem_2 <= (others => (others => '0'));
          mem_3 <= (others => (others => '0'));
          mem_4 <= (others => (others => '0'));
          mem_5 <= (others => (others => '0'));
          mem_6 <= (others => (others => '0'));
          mem_7 <= (others => (others => '0'));
          clear <= '1';

          -------------------------------------------------------------------
          --  WRITE TO FILE INDICATOR CHANGE OF PARAMETERS: "***"
          write(ptr,change_txt);
          writeline(fileTXT,ptr);
          -------------------------------------------------------------------
          wait for 3*period;

        end if;                         -- IMPOSSIBLES CASES

      end loop;  -- i_N
    end loop;  -- i_Pe

    -- WRITING A FLAG TO END:
    write(ptr,end_txt);
    writeline(fileTXT,ptr);

    wait;

  END PROCESS;

END;


-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity LIFOmem is

  generic (
    ADDRESS_SIZE : positive := 12;
    MEM_DEPTH    : positive := 400);

  port (
    clk       : in  std_logic;
    clear     : in  std_logic;
    rst_n     : in  std_logic;
    write_en  : in  std_logic;
    read_en   : in  std_logic;
    data_in   : in  std_logic_vector(8*ADDRESS_SIZE+23 downto 0);
    valid_out : out std_logic;
    data_out  : out std_logic_vector(8*ADDRESS_SIZE+23 downto 0));

end LIFOmem;


architecture behav_LIFOmem of LIFOmem is
```

```
  type memType is array (MEM_DEPTH-1 downto 0) of std_logic_vector(8*ADDRESS_SIZE+23 downto 0);
  signal memory : memType := (others => (others => '0'));
  signal pointer : integer := 0;

begin  -- behav_LIFOmem

  sync: process (clk, rst_n)

    -- Variable POINTER
    variable pnt : integer := 0;

  begin  -- process sync

    if (rst_n = '0') then
      valid_out <= '0';
      data_out <= (others => '0');
      pointer <= 0;
      memory <= (others => (others => '0'));

    -- SYNC: Rising edge
    elsif (clk = '1' and clk'event) then
      if (clear = '1') then
        valid_out <= '0';
        data_out <= (others => '0');
        pointer <= 0;
        memory <= (others => (others => '0'));

      else

        -----------------------------------------------------------------------
        pnt := pointer;

        if (read_en = '1') then

          if (write_en = '0') then
            if (pnt > 0) then
              valid_out <= '1';
              pnt := pnt - 1;
              data_out <= memory(pnt);
            else
              valid_out <= '0';
            end if;
          else          -- write_en = '1'
            valid_out <= '1';
            data_out <= data_in;
          end if;


        else            -- read_en = '0'
          valid_out <= '0';

          if (write_en = '1' and MEM_DEPTH > pnt) then
            memory(pnt) <= data_in;
            pnt := pnt + 1;
          end if;

        end if;

        pointer <= pnt;
        -----------------------------------------------------------------------

      end if;  -- clear
    end if;  -- clk & rst_n
```

```
  end process sync;

end behav_LIFOmem;


-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------



library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity conectando is

  generic (
    DATA_SIZE    : positive := 32;
    ADDRESS_SIZE : positive := 12);

  port (
    clk           : in  std_logic;
    enable        : in  std_logic;
    clear         : in  std_logic;
    rst_n         : in  std_logic;
    N_in          : in  std_logic_vector(11 downto 0);
    Pe_in         : in  std_logic_vector(3 downto 0);
    scrambled     : in  std_logic;
    dataSISO      : in  std_logic_vector(8*DATA_SIZE-1 downto 0);
    dataSISOvalid : in  std_logic;
    MEM_WADX      : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
    MEM_WDATA     : out std_logic_vector(8*DATA_SIZE-1 downto 0);
    MEM_WEN       : out std_logic_vector(7 downto 0));

end conectando;



architecture behav_conectando of conectando is

  component addressGenerator
    generic (
      ADDRESS_SIZE : positive);
    port (
      clk       : in  std_logic;
      enable    : in  std_logic;
      clear     : in  std_logic;
      rst_n     : in  std_logic;
      scrambled : in  std_logic;
      N_in      : in  std_logic_vector(11 downto 0);
      Pe_in     : in  std_logic_vector(3 downto 0);
      validOut  : out std_logic;
      mem_ref   : out std_logic_vector(23 downto 0);
      address   : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0));
  end component;

  component LIFOmem
    generic (
      ADDRESS_SIZE : positive;
      MEM_DEPTH    : positive);
    port (
      clk       : in  std_logic;
      clear     : in  std_logic;
```

```
      rst_n     : in  std_logic;
      write_en  : in  std_logic;
      read_en   : in  std_logic;
      data_in   : in  std_logic_vector(8*ADDRESS_SIZE+23 downto 0);
      valid_out : out std_logic;
      data_out  : out std_logic_vector(8*ADDRESS_SIZE+23 downto 0));
  end component;

  component switchWriteMemory
    generic (
      DATA_SIZE    : positive;
      ADDRESS_SIZE : positive;
      FIFO_LENGTH  : positive);
    port (
      clk         : in  std_logic;
      clear       : in  std_logic;
      rst_n       : in  std_logic;
      Pe_in       : in  std_logic_vector(3 downto 0);
      addressing  : in  std_logic_vector(8*ADDRESS_SIZE+23 downto 0);
      WDATA       : in  std_logic_vector(8*DATA_SIZE-1 downto 0);
      WDATA_VALID : in  std_logic;
      read_LIFO   : out std_logic;
      MEM_WADX    : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
      MEM_WDATA   : out std_logic_vector(8*DATA_SIZE-1 downto 0);
      MEM_WEN     : out std_logic_vector(7 downto 0));
  end component;


  -- SIGNALS
  signal enableMEM : std_logic_vector(7 downto 0);
  signal MEM_DATA : std_logic_vector(8*DATA_SIZE-1 downto 0);
  signal MEM_ADX : std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
  signal dataVALID : std_logic;
  signal mem_ref : std_logic_vector(23 downto 0);
  signal address : std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
  signal dataIN : std_logic_vector(8*ADDRESS_SIZE+23 downto 0);
  signal dataLIFOout : std_logic_vector(8*ADDRESS_SIZE+23 downto 0);
  signal dataLIFOvalid : std_logic;


begin  -- behav_conectando


  -- ADDRESS GENERATOR
  addGen : addressGenerator
    generic map (
      ADDRESS_SIZE => ADDRESS_SIZE)
    port map (
      clk       => clk,
      enable    => enable,
      clear     => clear,
      rst_n     => rst_n,
      scrambled => scrambled,
      N_in      => N_in,
      Pe_in     => Pe_in,
      validOut  => dataVALID,
      mem_ref   => mem_ref,
      address   => address);

  -- LIFO
  LIFO : LIFOmem
    generic map (
      ADDRESS_SIZE => ADDRESS_SIZE,
      MEM_DEPTH    => 2400)
    port map (
```

```
      clk       => clk,
      clear     => clear,
      rst_n     => rst_n,
      write_en  => dataVALID,
      read_en   => dataSISOvalid,
      data_in   => dataIN,
      valid_out => dataLIFOvalid,
      data_out  => dataLIFOout);

  -- SWITCH WRITE MEMORY
  SWM : switchWriteMemory
    generic map (
      DATA_SIZE    => DATA_SIZE,
      ADDRESS_SIZE => ADDRESS_SIZE,
      FIFO_LENGTH  => 6)
    port map (
      clk         => clk,
      clear       => clear,
      rst_n       => rst_n,
      Pe_in       => Pe_in,
      addressing  => dataLIFOout,
      WDATA       => dataSISO,
      WDATA_VALID => dataSISOvalid,
      MEM_WADX    => MEM_WADX,
      MEM_WDATA   => MEM_WDATA,
      MEM_WEN     => MEM_WEN);


  -- SIGNALS:
  dataIN(23 downto 0) <= mem_ref;
  dataIN(8*ADDRESS_SIZE+23 downto 24) <= address;



end behav_conectando;
```

XVII. *tb_SWM_SCRAMB.vhd*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library std;
use std.textio.all;

ENTITY tb_SWM_SCRAMB_vhd IS
END tb_SWM_SCRAMB_vhd;

ARCHITECTURE behavior OF tb_SWM_SCRAMB_vhd IS

  -- Component Declaration for the Unit Under Test (UUT)
  COMPONENT conectando
    PORT(
      clk : IN std_logic;
      enable : IN std_logic;
      clear : IN std_logic;
      rst_n : IN std_logic;
      N_in : IN std_logic_vector(11 downto 0);
      Pe_in : IN std_logic_vector(3 downto 0);
      scrambled : IN std_logic;
      dataSISO : IN std_logic_vector(255 downto 0);
      dataSISOvalid : IN std_logic;
      MEM_WADX : OUT std_logic_vector(95 downto 0);
      MEM_WDATA : OUT std_logic_vector(255 downto 0);
      MEM_WEN : OUT std_logic_vector(7 downto 0)
      );
  END COMPONENT;

  --Inputs
  SIGNAL clk :  std_logic := '1';
  SIGNAL enable :  std_logic := '0';
  SIGNAL clear :  std_logic := '1';
  SIGNAL rst_n :  std_logic := '0';
  SIGNAL scrambled :  std_logic := '1';
  SIGNAL dataSISOvalid :  std_logic := '0';
  SIGNAL N_in :  std_logic_vector(11 downto 0) := (others=>'0');
  SIGNAL Pe_in :  std_logic_vector(3 downto 0) := (others=>'0');
  SIGNAL dataSISO :  std_logic_vector(255 downto 0) := (others=>'0');

  --Outputs
  SIGNAL MEM_WADX :  std_logic_vector(95 downto 0);
  SIGNAL MEM_WDATA :  std_logic_vector(255 downto 0);
  SIGNAL MEM_WEN :  std_logic_vector(7 downto 0);

  --TIME
  constant period : time := 100 ns;


  ----------------------------------------------------------------------------
  --    MEMORY
  type memoryTYPE is array (2499 downto 0) of std_logic_vector(31 downto 0);
  signal mem_0 : memoryTYPE := (others => (others => '0'));
  signal mem_1 : memoryTYPE := (others => (others => '0'));
  signal mem_2 : memoryTYPE := (others => (others => '0'));
  signal mem_3 : memoryTYPE := (others => (others => '0'));
  signal mem_4 : memoryTYPE := (others => (others => '0'));
  signal mem_5 : memoryTYPE := (others => (others => '0'));
  signal mem_6 : memoryTYPE := (others => (others => '0'));
  signal mem_7 : memoryTYPE := (others => (others => '0'));
  type memoryADX is array (7 downto 0) of std_logic_vector(11 downto 0);
```

```
  type memoryDATA is array (7 downto 0) of std_logic_vector(31 downto 0);
  signal memADX : memoryADX := (others => (others => '0'));
  signal memDATA : memoryDATA := (others => (others => '0'));
  ----------------------------------------------------------------------------



  ----------------------------------------------------------------------------
  --    SISO
  type sisoTYPE is array (7 downto 0) of std_logic_vector(31 downto 0);
  signal SISO : sisoTYPE := (others => (others => '0'));
  ----------------------------------------------------------------------------



  --STRING CONSTANTS
  constant N_txt : string := "N = ";
  constant Pe_txt : string := "Pe = ";
  constant blank_txt : string := " ";
  constant empty_txt : string := "";
  constant change_txt : string := "***";
  constant space_txt : string := " - ";
  constant C_char : string := " C";
  constant end_txt : string := "END";

  --Constant for using at the simulator
  type arrayPe is array (4 downto 0) of std_logic_vector(3 downto 0);
  type arrayN is array (16 downto 0) of std_logic_vector(11 downto 0);
  constant Pe : arrayPe := ("1000", "0110", "0100", "0010", "0001");
  constant N : arrayN := (conv_std_logic_vector(2400,12),
                          conv_std_logic_vector(1920,12),
                          conv_std_logic_vector(1440,12),
                          conv_std_logic_vector(960,12),
                          conv_std_logic_vector(480,12),
                          conv_std_logic_vector(240,12),
                          conv_std_logic_vector(216,12),
                          conv_std_logic_vector(192,12),
                          conv_std_logic_vector(180,12),
                          conv_std_logic_vector(144,12),
                          conv_std_logic_vector(120,12),
                          conv_std_logic_vector(108,12),
                          conv_std_logic_vector(96,12),
                          conv_std_logic_vector(72,12),
                          conv_std_logic_vector(48,12),
                          conv_std_logic_vector(36,12),
                          conv_std_logic_vector(24,12));


BEGIN

  -- Instantiate the Unit Under Test (UUT)
  uut: conectando PORT MAP(
    clk => clk,
    enable => enable,
    clear => clear,
    rst_n => rst_n,
    N_in => N_in,
    Pe_in => Pe_in,
    scrambled => scrambled,
    dataSISO => dataSISO,
    dataSISOvalid => dataSISOvalid,
    MEM_WADX => MEM_WADX,
    MEM_WDATA => MEM_WDATA,
    MEM_WEN => MEM_WEN);

  clk <= not clk after period/2;
```

```
-----------------------------------------------------------------------------
--    SISO
sisoLOOP: for i in 7 downto 0 generate
  dataSISO(32*(i+1)-1 downto 32*i) <= SISO(i);
end generate sisoLOOP;
-----------------------------------------------------------------------------

-----------------------------------------------------------------------------
--    MEMORY
mem: for i in 7 downto 0 generate
  memADX(i) <= MEM_WADX(12*(i+1)-1 downto 12*i);
  memDATA(i) <= MEM_WDATA(32*(i+1)-1 downto 32*i);
end generate mem;
-----------------------------------------------------------------------------

tb : PROCESS

  file fileTXT : text open WRITE_MODE is "reportSWM_SCRAMB_VHDL.txt";
  variable tmp : integer;
  variable ptr  : line;
  variable counter : integer := 0;

BEGIN

  for i_Pe in 0 to 4 loop
    Pe_in <= Pe(i_Pe);

    for i_N in 0 to 16 loop
      N_in <= N(i_N);
      wait for period;

      -- Excluding cases:
      if ( not( Pe_in = "1000" and (N_in = conv_std_logic_vector(36,12) or
                                    N_in = conv_std_logic_vector(108,12) or
                                    N_in = conv_std_logic_vector(180,12)) ) ) then

        -------------------------------------------------------------------
        --  WRITE TO FILE THE INFORMATION OF THE ACTUAL OPERATION
        tmp := conv_integer(N_in);
        write(ptr,N_txt);
        write(ptr,tmp);
        writeline(fileTXT,ptr);
        tmp := conv_integer(Pe_in);
        write(ptr,Pe_txt);
        write(ptr,tmp);
        writeline(fileTXT,ptr);
        -------------------------------------------------------------------

        ---------------------------------------------------------------------
        for jj in 7 downto 0 loop
          SISO(jj) <= conv_std_logic_vector((jj+1)*10000,32);
        end loop;  -- jj
        ---------------------------------------------------------------------

        -- Counter -> 0
        counter := 0;

        -- SIMULATION
        enable <= '0';
        clear<= '1';
        rst_n <= '0';
        wait for 2*period;
        clear <= '0';
        rst_n <= '1';
        wait for 2*period;
```

```
-- Activation of the interleaver (N/Pe + 1  periods)
enable <= '1';
wait for (1+(conv_integer(N_in)/conv_integer(Pe_in)))*period;
enable <= '0';

wait for 5*period;

-- Activation of SISO valid
while (counter < conv_integer(N_in)/conv_integer(Pe_in)) loop
  dataSISOvalid <= '1';

  counter := counter + 1;

  -------------------------------------------------------------------
  -- SCRITURA CUANDO SALIDA DE LAS MEMORIAS SEAN VALIDAS xDDD
  if (MEM_WEN(0) = '1') then  -- Writing at "MEM 0"
    mem_0(conv_integer(memADX(0))) <= memDATA(0);
  end if;

  if (MEM_WEN(1) = '1') then  -- Writing at "MEM 1"
    mem_1(conv_integer(memADX(1))) <= memDATA(1);
  end if;

  if (MEM_WEN(2) = '1') then  -- Writing at "MEM 2"
    mem_2(conv_integer(memADX(2))) <= memDATA(2);
  end if;

  if (MEM_WEN(3) = '1') then  -- Writing at "MEM 3"
    mem_3(conv_integer(memADX(3))) <= memDATA(3);
  end if;

  if (MEM_WEN(4) = '1') then  -- Writing at "MEM 4"
    mem_4(conv_integer(memADX(4))) <= memDATA(4);
  end if;

  if (MEM_WEN(5) = '1') then  -- Writing at "MEM 5"
    mem_5(conv_integer(memADX(5))) <= memDATA(5);
  end if;

  if (MEM_WEN(6) = '1') then  -- Writing at "MEM 6"
    mem_6(conv_integer(memADX(6))) <= memDATA(6);
  end if;

  if (MEM_WEN(7) = '1') then  -- Writing at "MEM 7"
    mem_7(conv_integer(memADX(7))) <= memDATA(7);
  end if;
  -------------------------------------------------------------------

  wait for period;

  for jj in 7 downto 0 loop
    SISO(jj) <= SISO(jj) + 1;
  end loop;  -- jj

end loop;  -- WHILE

dataSISOvalid <= '0';

for jj in 20 downto 0 loop
  -------------------------------------------------------------------
  -- SCRITURA CUANDO SALIDA DE LAS MEMORIAS SEAN VALIDAS xDDD
  if (MEM_WEN(0) = '1') then  -- Writing at "MEM 0"
    mem_0(conv_integer(memADX(0))) <= memDATA(0);
```

```
    end if;

    if (MEM_WEN(1) = '1') then  -- Writing at "MEM 1"
      mem_1(conv_integer(memADX(1))) <= memDATA(1);
    end if;

    if (MEM_WEN(2) = '1') then  -- Writing at "MEM 2"
      mem_2(conv_integer(memADX(2))) <= memDATA(2);
    end if;

    if (MEM_WEN(3) = '1') then  -- Writing at "MEM 3"
      mem_3(conv_integer(memADX(3))) <= memDATA(3);
    end if;

    if (MEM_WEN(4) = '1') then  -- Writing at "MEM 4"
      mem_4(conv_integer(memADX(4))) <= memDATA(4);
    end if;

    if (MEM_WEN(5) = '1') then  -- Writing at "MEM 5"
      mem_5(conv_integer(memADX(5))) <= memDATA(5);
    end if;

    if (MEM_WEN(6) = '1') then  -- Writing at "MEM 6"
      mem_6(conv_integer(memADX(6))) <= memDATA(6);
    end if;

    if (MEM_WEN(7) = '1') then  -- Writing at "MEM 7"
      mem_7(conv_integer(memADX(7))) <= memDATA(7);
    end if;
    -------------------------------------------------------------------

    wait for period;
  end loop;  -- jj


  -- REPORTING:
  -- MEM_0  MEM_1  MEM_2  MEM_3  MEM_4  MEM_5  MEM_6  MEM_7
  for jj in 0 to 2499 loop
    tmp := conv_integer(mem_0(jj));
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(mem_1(jj));
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(mem_2(jj));
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(mem_3(jj));
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(mem_4(jj));
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(mem_5(jj));
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(mem_6(jj));
    write(ptr,tmp);
    write(ptr,blank_txt);
    tmp := conv_integer(mem_7(jj));
    write(ptr,tmp);

    writeline(fileTXT,ptr);

  end loop;  -- jj
```

```
          -- CLEAR MEMORIES:
          mem_0 <= (others => (others => '0'));
          mem_1 <= (others => (others => '0'));
          mem_2 <= (others => (others => '0'));
          mem_3 <= (others => (others => '0'));
          mem_4 <= (others => (others => '0'));
          mem_5 <= (others => (others => '0'));
          mem_6 <= (others => (others => '0'));
          mem_7 <= (others => (others => '0'));
          clear <= '1';

          -------------------------------------------------------------------
          --  WRITE TO FILE INDICATOR CHANGE OF PARAMETERS: "***"
          write(ptr,change_txt);
          writeline(fileTXT,ptr);
          -------------------------------------------------------------------
          wait for 3*period;

        end if;                        -- IMPOSSIBLES CASES

      end loop;  -- i_N
    end loop;  -- i_Pe

    -- WRITING A FLAG TO END:
    write(ptr,end_txt);
    writeline(fileTXT,ptr);

    wait;

  END PROCESS;

END;


-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity LIFOmem is

  generic (
    ADDRESS_SIZE : positive := 12;
    MEM_DEPTH    : positive := 400);

  port (
    clk       : in  std_logic;
    clear     : in  std_logic;
    rst_n     : in  std_logic;
    write_en  : in  std_logic;
    read_en   : in  std_logic;
    data_in   : in  std_logic_vector(8*ADDRESS_SIZE+23 downto 0);
    valid_out : out std_logic;
    data_out  : out std_logic_vector(8*ADDRESS_SIZE+23 downto 0));

end LIFOmem;


architecture behav_LIFOmem of LIFOmem is
```

```
  type memType is array (MEM_DEPTH-1 downto 0) of std_logic_vector(8*ADDRESS_SIZE+23 downto 0);
  signal memory : memType := (others => (others => '0'));
  signal pointer : integer := 0;

begin  -- behav_LIFOmem

  sync: process (clk, rst_n)

    -- Variable POINTER
    variable pnt : integer := 0;

  begin  -- process sync

    if (rst_n = '0') then
      valid_out <= '0';
      data_out <= (others => '0');
      pointer <= 0;
      memory <= (others => (others => '0'));

    -- SYNC: Rising edge
    elsif (clk = '1' and clk'event) then
      if (clear = '1') then
        valid_out <= '0';
        data_out <= (others => '0');
        pointer <= 0;
        memory <= (others => (others => '0'));

      else

        -----------------------------------------------------------------------
        pnt := pointer;

        if (read_en = '1') then

          if (write_en = '0') then
            if (pnt > 0) then
              valid_out <= '1';
              pnt := pnt - 1;
              data_out <= memory(pnt);
            else
              valid_out <= '0';
            end if;
          else         -- write_en = '1'
            valid_out <= '1';
            data_out <= data_in;
          end if;


        else           -- read_en = '0'
          valid_out <= '0';

          if (write_en = '1' and MEM_DEPTH > pnt) then
            memory(pnt) <= data_in;
            pnt := pnt + 1;
          end if;

        end if;

        pointer <= pnt;
        -----------------------------------------------------------------------

      end if;  -- clear
    end if;  -- clk & rst_n
```

```
  end process sync;

end behav_LIFOmem;


-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------



library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity conectando is

  generic (
    DATA_SIZE    : positive := 32;
    ADDRESS_SIZE : positive := 12);

  port (
    clk           : in  std_logic;
    enable        : in  std_logic;
    clear         : in  std_logic;
    rst_n         : in  std_logic;
    N_in          : in  std_logic_vector(11 downto 0);
    Pe_in         : in  std_logic_vector(3 downto 0);
    scrambled     : in  std_logic;
    dataSISO      : in  std_logic_vector(8*DATA_SIZE-1 downto 0);
    dataSISOvalid : in  std_logic;
    MEM_WADX      : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
    MEM_WDATA     : out std_logic_vector(8*DATA_SIZE-1 downto 0);
    MEM_WEN       : out std_logic_vector(7 downto 0));

end conectando;



architecture behav_conectando of conectando is

  component addressGenerator
    generic (
      ADDRESS_SIZE : positive);
    port (
      clk       : in  std_logic;
      enable    : in  std_logic;
      clear     : in  std_logic;
      rst_n     : in  std_logic;
      scrambled : in  std_logic;
      N_in      : in  std_logic_vector(11 downto 0);
      Pe_in     : in  std_logic_vector(3 downto 0);
      validOut  : out std_logic;
      mem_ref   : out std_logic_vector(23 downto 0);
      address   : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0));
  end component;

  component LIFOmem
    generic (
      ADDRESS_SIZE : positive;
      MEM_DEPTH    : positive);
    port (
      clk       : in  std_logic;
      clear     : in  std_logic;
```

```
      rst_n     : in  std_logic;
      write_en  : in  std_logic;
      read_en   : in  std_logic;
      data_in   : in  std_logic_vector(8*ADDRESS_SIZE+23 downto 0);
      valid_out : out std_logic;
      data_out  : out std_logic_vector(8*ADDRESS_SIZE+23 downto 0));
  end component;

  component switchWriteMemory
    generic (
      DATA_SIZE    : positive;
      ADDRESS_SIZE : positive;
      FIFO_LENGTH  : positive);
    port (
      clk         : in  std_logic;
      clear       : in  std_logic;
      rst_n       : in  std_logic;
      Pe_in       : in  std_logic_vector(3 downto 0);
      addressing  : in  std_logic_vector(8*ADDRESS_SIZE+23 downto 0);
      WDATA       : in  std_logic_vector(8*DATA_SIZE-1 downto 0);
      WDATA_VALID : in  std_logic;
      read_LIFO   : out std_logic;
      MEM_WADX    : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
      MEM_WDATA   : out std_logic_vector(8*DATA_SIZE-1 downto 0);
      MEM_WEN     : out std_logic_vector(7 downto 0));
  end component;


  -- SIGNALS
  signal enableMEM : std_logic_vector(7 downto 0);
  signal MEM_DATA : std_logic_vector(8*DATA_SIZE-1 downto 0);
  signal MEM_ADX : std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
  signal dataVALID : std_logic;
  signal mem_ref : std_logic_vector(23 downto 0);
  signal address : std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
  signal dataIN : std_logic_vector(8*ADDRESS_SIZE+23 downto 0);
  signal dataLIFOout : std_logic_vector(8*ADDRESS_SIZE+23 downto 0);
  signal dataLIFOvalid : std_logic;


begin  -- behav_conectando


  -- ADDRESS GENERATOR
  addGen : addressGenerator
    generic map (
      ADDRESS_SIZE => ADDRESS_SIZE)
    port map (
      clk       => clk,
      enable    => enable,
      clear     => clear,
      rst_n     => rst_n,
      scrambled => scrambled,
      N_in      => N_in,
      Pe_in     => Pe_in,
      validOut  => dataVALID,
      mem_ref   => mem_ref,
      address   => address);

  -- LIFO
  LIFO : LIFOmem
    generic map (
      ADDRESS_SIZE => ADDRESS_SIZE,
      MEM_DEPTH    => 2400)
    port map (
```

```
      clk       => clk,
      clear     => clear,
      rst_n     => rst_n,
      write_en  => dataVALID,
      read_en   => dataSISOvalid,
      data_in   => dataIN,
      valid_out => dataLIFOvalid,
      data_out  => dataLIFOout);

  -- SWITCH WRITE MEMORY
  SWM : switchWriteMemory
    generic map (
      DATA_SIZE    => DATA_SIZE,
      ADDRESS_SIZE => ADDRESS_SIZE,
      FIFO_LENGTH  => 6)
    port map (
      clk         => clk,
      clear       => clear,
      rst_n       => rst_n,
      Pe_in       => Pe_in,
      addressing  => dataLIFOout,
      WDATA       => dataSISO,
      WDATA_VALID => dataSISOvalid,
      MEM_WADX    => MEM_WADX,
      MEM_WDATA   => MEM_WDATA,
      MEM_WEN     => MEM_WEN);


  -- SIGNALS:
  dataIN(23 downto 0) <= mem_ref;
  dataIN(8*ADDRESS_SIZE+23 downto 24) <= address;



end behav_conectando;
```

XVIII. *tb_interleaver.vhd*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library std;
use std.textio.all;

ENTITY tb_interleaver_vhd IS
END tb_interleaver_vhd;

ARCHITECTURE behavior OF tb_interleaver_vhd IS

  -- Component Declaration for the Unit Under Test (UUT)
  COMPONENT conectando
    PORT(
      clk : IN std_logic;
      clear : IN std_logic;
      rst_n : IN std_logic;
      N : IN std_logic_vector(11 downto 0);
      PE : IN std_logic_vector(3 downto 0);
      SCRAMBLED : IN std_logic;
      BURST_LENGTH : IN std_logic_vector(11 downto 0);
      START_READ : IN std_logic;
      CLEAR_ADDGEN : IN std_logic;
      WDATA : IN std_logic_vector(255 downto 0);
      WDATA_VALID : IN std_logic;
      RDATA_VALID : OUT std_logic;
      RDATA : OUT std_logic_vector(255 downto 0);
      MEM_WADX : OUT std_logic_vector(95 downto 0);
      MEM_WDATA : OUT std_logic_vector(255 downto 0);
      MEM_WEN : OUT std_logic_vector(7 downto 0));
  END COMPONENT;

  --Inputs
  SIGNAL clk :  std_logic := '1';
  SIGNAL clear :  std_logic := '1';
  SIGNAL rst_n :  std_logic := '1';
  SIGNAL SCRAMBLED :  std_logic := '0';
  SIGNAL START_READ :  std_logic := '0';
  SIGNAL CLEAR_ADDGEN :  std_logic := '0';
  SIGNAL WDATA_VALID :  std_logic := '0';
  SIGNAL N :  std_logic_vector(11 downto 0) := (others=>'0');
  SIGNAL PE :  std_logic_vector(3 downto 0) := (others=>'0');
  SIGNAL BURST_LENGTH :  std_logic_vector(11 downto 0) := (others=>'0');
  SIGNAL WDATA :  std_logic_vector(255 downto 0) := (others=>'0');

  --Outputs
  SIGNAL RDATA_VALID :  std_logic;
  SIGNAL RDATA :  std_logic_vector(255 downto 0);
  SIGNAL MEM_WADX :  std_logic_vector(95 downto 0);
  SIGNAL MEM_WDATA :  std_logic_vector(255 downto 0);
  SIGNAL MEM_WEN :  std_logic_vector(7 downto 0);

  -- Signal COUNTER
  signal counter : integer := 0;

  -- SISO & MEMORY write signals:
  type arrayAUX is array (7 downto 0) of std_logic_vector(31 downto 0);
  signal SISOs : arrayAUX := (others => (others => '0'));
  type memoryTYPE is array (2499 downto 0) of std_logic_vector(31 downto 0);
  signal mem_0 : memoryTYPE := (others => (others => '0'));
  signal mem_1 : memoryTYPE := (others => (others => '0'));
```

```
signal mem_2 : memoryTYPE := (others => (others => '0'));
signal mem_3 : memoryTYPE := (others => (others => '0'));
signal mem_4 : memoryTYPE := (others => (others => '0'));
signal mem_5 : memoryTYPE := (others => (others => '0'));
signal mem_6 : memoryTYPE := (others => (others => '0'));
signal mem_7 : memoryTYPE := (others => (others => '0'));
type memoryADX is array (7 downto 0) of std_logic_vector(11 downto 0);
type memoryDATA is array (7 downto 0) of std_logic_vector(31 downto 0);
signal memADX : memoryADX := (others => (others => '0'));
signal memDATA : memoryDATA := (others => (others => '0'));

--Time
constant period : time := 100 ns;

--STRING CONSTANTS
constant N_txt : string := "N = ";
constant Pe_txt : string := "Pe = ";
constant BURST_txt : string := "WINDOWS LENGTH = ";
constant blank_txt : string := " ";
constant empty_txt : string := "";
constant change_txt : string := "***";
constant space_txt : string := " - ";
constant C_char : string := " C";
constant end_txt : string := "END";

--Constant for using at the simulator
type arrayPe is array (4 downto 0) of std_logic_vector(3 downto 0);
type arrayN is array (16 downto 0) of std_logic_vector(11 downto 0);
constant Pe_array : arrayPe := ("1000", "0110", "0100", "0010", "0001");
constant N_array : arrayN := (conv_std_logic_vector(2400,12),
                              conv_std_logic_vector(1920,12),
                              conv_std_logic_vector(1440,12),
                              conv_std_logic_vector(960,12),
                              conv_std_logic_vector(480,12),
                              conv_std_logic_vector(240,12),
                              conv_std_logic_vector(216,12),
                              conv_std_logic_vector(192,12),
                              conv_std_logic_vector(180,12),
                              conv_std_logic_vector(144,12),
                              conv_std_logic_vector(120,12),
                              conv_std_logic_vector(108,12),
                              conv_std_logic_vector(96,12),
                              conv_std_logic_vector(72,12),
                              conv_std_logic_vector(48,12),
                              conv_std_logic_vector(36,12),
                              conv_std_logic_vector(24,12));
constant BURST_array : arrayN := (conv_std_logic_vector(50,12),
                                  conv_std_logic_vector(40,12),
                                  conv_std_logic_vector(60,12),
                                  conv_std_logic_vector(40,12),
                                  conv_std_logic_vector(20,12),
                                  conv_std_logic_vector(10,12),
                                  conv_std_logic_vector(9,12),
                                  conv_std_logic_vector(8,12),
                                  conv_std_logic_vector(5,12),
                                  conv_std_logic_vector(6,12),
                                  conv_std_logic_vector(5,12),
                                  conv_std_logic_vector(9,12),
                                  conv_std_logic_vector(4,12),
                                  conv_std_logic_vector(3,12),
                                  conv_std_logic_vector(2,12),
                                  conv_std_logic_vector(3,12),
                                  conv_std_logic_vector(1,12));
```

```
BEGIN

  -- Instantiate the Unit Under Test (UUT)
  uut: conectando PORT MAP(
    clk => clk,
    clear => clear,
    rst_n => rst_n,
    N => N,
    PE => PE,
    SCRAMBLED => SCRAMBLED,
    BURST_LENGTH => BURST_LENGTH,
    START_READ => START_READ,
    CLEAR_ADDGEN => CLEAR_ADDGEN,
    WDATA => WDATA,
    WDATA_VALID => WDATA_VALID,
    RDATA_VALID => RDATA_VALID,
    RDATA => RDATA,
    MEM_WADX => MEM_WADX,
    MEM_WDATA => MEM_WDATA,
    MEM_WEN => MEM_WEN);

  clk <= not clk after (period/2);


  sisoLOOP: for i in 7 downto 0 generate
    WDATA(32*(i+1)-1 downto 32*i) <= SISOs(i);
  end generate sisoLOOP;

  mem: for i in 7 downto 0 generate
    memADX(i) <= MEM_WADX(12*(i+1)-1 downto 12*i);
    memDATA(i) <= MEM_WDATA(32*(i+1)-1 downto 32*i);
  end generate mem;


  writingFILE: process (clk)

    file fileTXTsiso : text open WRITE_MODE is "report_INTER_SISOs_VHDL.txt";
    variable tmp : integer;
    variable ptr : line;

  begin  -- process writingFILE

    if (clk = '1' and clk'event) then

      if (RDATA_VALID = '1') then
        if (PE = "0001") then
          tmp := conv_integer(RDATA(31 downto 0));
          write(ptr,tmp);
        elsif (PE = "0010") then
          tmp := conv_integer(RDATA(31 downto 0));
          write(ptr,tmp);
          write(ptr,blank_txt);
          tmp := conv_integer(RDATA(63 downto 32));
          write(ptr,tmp);
        elsif (PE = "0100") then
          tmp := conv_integer(RDATA(31 downto 0));
          write(ptr,tmp);
          write(ptr,blank_txt);
          tmp := conv_integer(RDATA(63 downto 32));
          write(ptr,tmp);
          write(ptr,blank_txt);
          tmp := conv_integer(RDATA(95 downto 64));
          write(ptr,tmp);
          write(ptr,blank_txt);
          tmp := conv_integer(RDATA(127 downto 96));
```

```
        write(ptr,tmp);
      elsif (PE = "0110") then
        tmp := conv_integer(RDATA(31 downto 0));
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(RDATA(63 downto 32));
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(RDATA(95 downto 64));
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(RDATA(127 downto 96));
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(RDATA(159 downto 128));
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(RDATA(191 downto 160));
        write(ptr,tmp);
      elsif (PE = "1000") then
        tmp := conv_integer(RDATA(31 downto 0));
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(RDATA(63 downto 32));
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(RDATA(95 downto 64));
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(RDATA(127 downto 96));
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(RDATA(159 downto 128));
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(RDATA(191 downto 160));
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(RDATA(223 downto 192));
        write(ptr,tmp);
        write(ptr,blank_txt);
        tmp := conv_integer(RDATA(255 downto 224));
        write(ptr,tmp);
      end if;
      writeline(fileTXTsiso,ptr);
    end if;
  end if;

end process writingFILE;


tb : PROCESS

  file fileTXTsiso : text open WRITE_MODE is "report_INTER_SISOs_VHDL.txt";
  file fileTXTmem : text open WRITE_MODE is "report_INTER_MEMs_VHDL.txt";
  variable tmp : integer;
  variable ptr : line;
  variable iterations : integer := 0;

BEGIN

  for i_Pe in 0 to 4 loop
    PE <= Pe_array(i_Pe);

    for i_N in 0 to 16 loop
```

```
-- CLEAR MEMORIES:
mem_0 <= (others => (others => '0'));
mem_1 <= (others => (others => '0'));
mem_2 <= (others => (others => '0'));
mem_3 <= (others => (others => '0'));
mem_4 <= (others => (others => '0'));
mem_5 <= (others => (others => '0'));
mem_6 <= (others => (others => '0'));
mem_7 <= (others => (others => '0'));

N <= N_array(i_N);
BURST_LENGTH <= BURST_array(i_N);
wait for period;

-- Excluding cases:
if ( not( PE = "1000" and (N = conv_std_logic_vector(36,12) or
                           N = conv_std_logic_vector(108,12) or
                           N = conv_std_logic_vector(180,12)) ) ) then

  iterations := ((conv_integer(N)/conv_integer(Pe))/conv_integer(BURST_LENGTH))-1;

  -------------------------------------------------------------------
  --  WRITE TO FILE THE INFORMATION OF THE ACTUAL OPERATION
  tmp := conv_integer(N);
  write(ptr,N_txt);
  write(ptr,tmp);
  writeline(fileTXTsiso,ptr);
  tmp := conv_integer(PE);
  write(ptr,Pe_txt);
  write(ptr,tmp);
  writeline(fileTXTsiso,ptr);
  tmp := conv_integer(BURST_LENGTH);
  write(ptr,BURST_txt);
  write(ptr,tmp);
  writeline(fileTXTsiso,ptr);
  tmp := conv_integer(N);
  write(ptr,N_txt);
  write(ptr,tmp);
  writeline(fileTXTmem,ptr);
  tmp := conv_integer(PE);
  write(ptr,Pe_txt);
  write(ptr,tmp);
  writeline(fileTXTmem,ptr);
  tmp := conv_integer(BURST_LENGTH);
  write(ptr,BURST_txt);
  write(ptr,tmp);
  writeline(fileTXTmem,ptr);
  -------------------------------------------------------------------


  for jj in 7 downto 0 loop
    SISOs(jj) <= conv_std_logic_vector(((jj+1)*10000)-1,32);
  end loop;  -- jj


  -- SIMULATION
  clear<= '1';
  rst_n <= '0';
  wait for 2*period;
  clear <= '0';
  rst_n <= '1';
  wait for 5*period;


  CLEAR_ADDGEN <= '1';
```

```
wait for period;
CLEAR_ADDGEN <= '0';
START_READ <= '1';
wait for period;
START_READ <= '0';

for jj in 2*conv_integer(BURST_LENGTH) downto 0 loop
  wait for period;
end loop;  -- jj


for ii in iterations downto 1 loop

  --Reseting counter
  counter <= 0;

  SCRAMBLED <= not( SCRAMBLED );
  START_READ <= '1';
  wait for period;
  START_READ <= '0';

  for jj in 10*conv_integer(BURST_LENGTH) downto 0 loop

    if (counter < BURST_LENGTH) then
      WDATA_VALID <= '1';
      counter <= counter + 1;
      for kk in 7 downto 0 loop
        SISOs(kk) <= SISOs(kk) + 1;
      end loop;  -- kk
    else
      WDATA_VALID <= '0';
    end if;

    if (MEM_WEN(0) = '1') then  -- Writing at "MEM 0"
      mem_0(conv_integer(memADX(0))) <= memDATA(0);
    end if;
    if (MEM_WEN(1) = '1') then  -- Writing at "MEM 1"
      mem_1(conv_integer(memADX(1))) <= memDATA(1);
    end if;
    if (MEM_WEN(2) = '1') then  -- Writing at "MEM 2"
      mem_2(conv_integer(memADX(2))) <= memDATA(2);
    end if;
    if (MEM_WEN(3) = '1') then  -- Writing at "MEM 3"
      mem_3(conv_integer(memADX(3))) <= memDATA(3);
    end if;
    if (MEM_WEN(4) = '1') then  -- Writing at "MEM 4"
      mem_4(conv_integer(memADX(4))) <= memDATA(4);
    end if;
    if (MEM_WEN(5) = '1') then  -- Writing at "MEM 5"
      mem_5(conv_integer(memADX(5))) <= memDATA(5);
    end if;
    if (MEM_WEN(6) = '1') then  -- Writing at "MEM 6"
      mem_6(conv_integer(memADX(6))) <= memDATA(6);
    end if;
    if (MEM_WEN(7) = '1') then  -- Writing at "MEM 7"
      mem_7(conv_integer(memADX(7))) <= memDATA(7);
    end if;

    wait for period;
  end loop;  -- jj

end loop;  -- ii

--Reseting counter
counter <= 0;
```

```
SCRAMBLED <= not( SCRAMBLED );

for jj in 10*conv_integer(BURST_LENGTH) downto 0 loop
  if (counter < BURST_LENGTH) then
    WDATA_VALID <= '1';
    counter <= counter + 1;
    for kk in 7 downto 0 loop
      SISOs(kk) <= SISOs(kk) + 1;
    end loop;  -- kk
  else
    WDATA_VALID <= '0';
  end if;

  if (MEM_WEN(0) = '1') then  -- Writing at "MEM 0"
    mem_0(conv_integer(memADX(0))) <= memDATA(0);
  end if;
  if (MEM_WEN(1) = '1') then  -- Writing at "MEM 1"
    mem_1(conv_integer(memADX(1))) <= memDATA(1);
  end if;
  if (MEM_WEN(2) = '1') then  -- Writing at "MEM 2"
    mem_2(conv_integer(memADX(2))) <= memDATA(2);
  end if;
  if (MEM_WEN(3) = '1') then  -- Writing at "MEM 3"
    mem_3(conv_integer(memADX(3))) <= memDATA(3);
  end if;
  if (MEM_WEN(4) = '1') then  -- Writing at "MEM 4"
    mem_4(conv_integer(memADX(4))) <= memDATA(4);
  end if;
  if (MEM_WEN(5) = '1') then  -- Writing at "MEM 5"
    mem_5(conv_integer(memADX(5))) <= memDATA(5);
  end if;
  if (MEM_WEN(6) = '1') then  -- Writing at "MEM 6"
    mem_6(conv_integer(memADX(6))) <= memDATA(6);
  end if;
  if (MEM_WEN(7) = '1') then  -- Writing at "MEM 7"
    mem_7(conv_integer(memADX(7))) <= memDATA(7);
  end if;

  wait for period;
end loop;  -- jj

wait for 5*period;

-- REPORTING MEMORY WRITE:
-- MEM_0  MEM_1  MEM_2  MEM_3  MEM_4  MEM_5  MEM_6  MEM_7
for jj in 0 to 2499 loop
  tmp := conv_integer(mem_0(jj));
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(mem_1(jj));
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(mem_2(jj));
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(mem_3(jj));
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(mem_4(jj));
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(mem_5(jj));
  write(ptr,tmp);
  write(ptr,blank_txt);
  tmp := conv_integer(mem_6(jj));
```

```
          write(ptr,tmp);
          write(ptr,blank_txt);
          tmp := conv_integer(mem_7(jj));
          write(ptr,tmp);
          writeline(fileTXTmem,ptr);
        end loop;  -- jj


        clear <= '1';
        SCRAMBLED <= '0';

        -------------------------------------------------------------------
        --  WRITE TO FILE INDICATOR CHANGE OF PARAMETERS: "***"
        write(ptr,change_txt);
        writeline(fileTXTsiso,ptr);
        write(ptr,change_txt);
        writeline(fileTXTmem,ptr);
        -------------------------------------------------------------------
        wait for 5*period;

      end if;                        -- IMPOSSIBLES CASES

    end loop;  -- i_N
  end loop;  -- i_Pe

  -- WRITING A FLAG TO END:
  write(ptr,end_txt);
  writeline(fileTXTsiso,ptr);
  write(ptr,end_txt);
  writeline(fileTXTmem,ptr);

  clear <= '1';

  wait;

 END PROCESS;

END;


-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity mem_simulation is

  generic (
    DATA_SIZE    : positive := 32;
    ADDRESS_SIZE : positive := 12);

  port (
    clk      : in  std_logic;
    enable   : in  std_logic;
    offset   : in  integer;
    address  : in  std_logic_vector(ADDRESS_SIZE-1 downto 0);
    data_out : out std_logic_vector(DATA_SIZE-1 downto 0));

end mem_simulation;
```

```
architecture behav_mem_simulation of mem_simulation is

  type memContent_type is array (0 to (2**ADDRESS_SIZE)-1) of std_logic_vector(DATA_SIZE-1 downto 0);
  signal memoryContent : memContent_type;

begin  -- behav_mem_simulation

  loadingDataROM: for i in 0 to (2**ADDRESS_SIZE)-1 generate
    memoryContent(i) <= conv_std_logic_vector(i+(offset*10000),DATA_SIZE);
  end generate loadingDataROM;

  unico: process (clk)
  begin  -- process unico

    if (clk = '1' and clk'event) then
      if (enable = '1') then
        data_out <= memoryContent(conv_integer(address));
      end if;
    end if;

  end process unico;

end behav_mem_simulation;


-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity conectando is

  generic (
    DATA_SIZE    : positive := 32;
    ADDRESS_SIZE : positive := 12);

  port (
    clk          : in  std_logic;
    clear        : in  std_logic;
    rst_n        : in  std_logic;
    N            : in  std_logic_vector(11 downto 0);
    PE           : in  std_logic_vector(3 downto 0);
    SCRAMBLED    : in  std_logic;
    BURST_LENGTH : in  std_logic_vector(11 downto 0);
    START_READ   : in  std_logic;
    CLEAR_ADDGEN : in  std_logic;
    WDATA        : in  std_logic_vector(8*DATA_SIZE-1 downto 0);
    WDATA_VALID  : in  std_logic;
    RDATA_VALID  : out std_logic;
    RDATA        : out std_logic_vector(8*DATA_SIZE-1 downto 0);
    MEM_WADX     : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
    MEM_WDATA    : out std_logic_vector(8*DATA_SIZE-1 downto 0);
    MEM_WEN      : out std_logic_vector(7 downto 0));

end conectando;


architecture behav_conectando of conectando is
```

```
  component interleaver
    generic (
      DATA_SIZE    : positive;
      ADDRESS_SIZE : positive);
    port (
      clk          : in  std_logic;
      clear        : in  std_logic;
      rst_n        : in  std_logic;
      N            : in  std_logic_vector(11 downto 0);
      PE           : in  std_logic_vector(3 downto 0);
      SCRAMBLED    : in  std_logic;
      BURST_LENGTH : in  std_logic_vector(11 downto 0);
      START_READ   : in  std_logic;
      CLEAR_ADDGEN : in  std_logic;
      WDATA        : in  std_logic_vector(8*DATA_SIZE-1 downto 0);
      WDATA_VALID  : in  std_logic;
      MEM_RDATA    : in  std_logic_vector(8*DATA_SIZE-1 downto 0);
      RDATA_VALID  : out std_logic;
      RDATA        : out std_logic_vector(8*DATA_SIZE-1 downto 0);
      MEM_RADX     : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
      MEM_REN      : out std_logic_vector(7 downto 0);
      MEM_WADX     : out std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
      MEM_WDATA    : out std_logic_vector(8*DATA_SIZE-1 downto 0);
      MEM_WEN      : out std_logic_vector(7 downto 0));
  end component;

  component mem_simulation
    generic (
      DATA_SIZE    : positive;
      ADDRESS_SIZE : positive);
    port (
      clk      : in  std_logic;
      enable   : in  std_logic;
      offset   : in  integer;
      address  : in  std_logic_vector(ADDRESS_SIZE-1 downto 0);
      data_out : out std_logic_vector(DATA_SIZE-1 downto 0));
  end component;

  -- SIGNALS
  signal MEM_RDATA : std_logic_vector(8*DATA_SIZE-1 downto 0);
  signal MEM_RADX : std_logic_vector(8*ADDRESS_SIZE-1 downto 0);
  signal MEM_REN : std_logic_vector(7 downto 0);

begin  -- behav_conectando


  -- MEMORIES:
  mem_READ: for i in 0 to 7 generate

    memoryREAD : mem_simulation
      generic map (
        DATA_SIZE    => DATA_SIZE,
        ADDRESS_SIZE => ADDRESS_SIZE)
      port map (
        clk      => clk,
        enable   => MEM_REN(i),
        offset   => i,
        address  => MEM_RADX((i+1)*ADDRESS_SIZE-1 downto ADDRESS_SIZE*i),
        data_out => MEM_RDATA((i+1)*DATA_SIZE-1 downto DATA_SIZE*i));
  end generate mem_READ;

  inter : interleaver
    generic map (
      DATA_SIZE    => DATA_SIZE,
      ADDRESS_SIZE => ADDRESS_SIZE)
```

```
    port map (
      clk          => clk,
      clear        => clear,
      rst_n        => rst_n,
      N            => N,
      PE           => PE,
      SCRAMBLED    => SCRAMBLED,
      BURST_LENGTH => BURST_LENGTH,
      START_READ   => START_READ,
      CLEAR_ADDGEN => CLEAR_ADDGEN,
      WDATA        => WDATA,
      WDATA_VALID  => WDATA_VALID,
      MEM_RDATA    => MEM_RDATA,
      RDATA_VALID  => RDATA_VALID,
      RDATA        => RDATA,
      MEM_RADX     => MEM_RADX,
      MEM_REN      => MEM_REN,
      MEM_WADX     => MEM_WADX,
      MEM_WDATA    => MEM_WDATA,
      MEM_WEN      => MEM_WEN);

end behav_conectando;
```