Universidad de Sevilla
Escuela Técnica Superior de Ingeniería Informática
Departamento de Arquitectura y Tecnología de Computadores.

Tesis doctoral

Una aportación al procesamiento de la información visual mediante técnicas bioinspiradas.

Autor:

Rafael Paz Vicente

Directores:

Dr. Alejandro Linares Barranco

Dr. Gabriel Jiménez Moreno

Sevilla, Octubre 2008

Universidad de Sevilla
Escuela Técnica Superior de Ingeniería Informática
Departamento de Arquitectura y Tecnología de Computadores.

Tesis Doctoral

Una aportación al procesamiento de la información visual mediante técnicas bioinspiradas.

Memoria presentada para aspirar al grado de doctor por

Rafael Paz Vicente.
Ingeniero Informático.

Los profesores ponentes y directores:

Dr. Gabriel Jiménez Moreno

Profesor Titular.
Departamento de Arquitectura y Tecnología de Computadores.

Dr. Alejandro Linares Barranco

Profesor Contratado Doctor
Departamento de Arquitectura y Tecnología de Computadores.

Sevilla, Octubre 2008

Agradecimientos

Este trabajo ha sido posible gracias a la dedicación e implicación de muchas personas, las cuales, de una forma u otra, han ayudado, apoyado, orientado animado y/o motivado para el desarrollo del trabajo presentado en esta tesis.. En especial:

- A Francisco Gómez Rodríguez por su trabajo, sobre todo en lo que concierne a la elaboración del mapper probabilístico.
- A Antón Civit por sus sugerencias y toda la ayuda que me ha prestado.
- A Angel Jiménez por su ayuda en en laboratorio y en las implementaciones realizadas.
- A todos los compañeros del departamento de ATC, ya que sin sus aportaciones y ánimos no hubiera sido posible esta tesis.
- A los socios de los proyectos CAVIAR, SAMANTA 1 y SAMANTA 2, por los medios aportados por dichos proyectos, y las sugerencias y comentarios recibidos.
- A mi familia y amigos que siempre me han apoyado en este trabajo.
- A Gabriel Jiménez y Alejandro Linares por su labor como directores de este trabajo.

Índice de contenido

Índice de ilustraciones

Índice de tablas

Índice de ecuaciones

1. Introducción

En el avance de la tecnología unas de las fuentes de inspiración para la humanidad más proliferas ha sido la naturaleza y, en particular, los sistemas biológicos. Desde el nacimiento de la vida en la Tierra estos sistemas se están optimizando constantemente a través de la evolución. Por tanto, es normal que se intente aprovechar este desarrollo natural imitando, en parte, a los sistemas biológicos. De este proceder por parte de la comunidad científica ha surgido el término "sistema bioinspirado" [GROSSBERG01][DELORME01][MAHOWALD01]. La variedad de sistemas que se pueden englobar en esta terminología es enorme, sin embargo, el sistema nervioso y la neurona, como unidad de éste, aparecen como unos de los que tienen mayor proyección y posibilidades [ROUSSELET01][DELORME02][FABRE01]. El interés de estos sistemas neuroinspirados es doble, por una parte se pretende probar modelos que nos permitan conocer mejor cómo es dicho funcionamiento neuronal en la naturaleza, y por otra imitar el mismo de forma artificial, para obtener dispositivos más eficaces. Es evidente que el ser humano es una buena muestra de lo efectivo que puede llegar a ser un sistema neuronal, hay ciertas características del procesado del cerebro humano que sin lugar a dudas no pueden ser superadas por las actuales computadoras. El problema en este caso radica en que no se conoce cómo se realizan estos procesos en último término, por lo que el estudio de los sistemas neuroinspirados cumplen con el doble objetivo anteriormente señalado: entender la naturaleza y copiarla en parte.

Uno de los principales pilares en los que se basan los sistemas neuroinspirados es en la naturaleza eléctrica del sistema nervioso, estando en la base de la comunicación entre neuronas la emisión de señales eléctricas pulsantes, este hecho está corroborado por numerosos estudios [GROSSBERG01][SHEPHERD01]. También hay evidencias de que el procesado en los seres vivos no se corresponde sólo con estas señales pulsantes y que, por ejemplo, en la memoria a largo plazo intervienen procesos químicos y de reordenación neuronal [SHEPHERD01][ROUSSELET01]. Pero no caben dudas de que la velocidad con la que se realizan las acciones o se percibe el mundo exterior mediante los sentidos por parte de los seres vivos son debido a la naturaleza eléctrica de estos sistemas, ya que de otra forma no se podrían explicar los tiempos de respuesta [VANRULLEN01].

Dentro de los sistemas neuroinspirados podemos encontrar dos alternativas, los que utilizan modelos y redes neuronales sin tener en cuenta la naturaleza pulsante de la comunicación entre neuronas y los que intentan emular con algo más de profundidad el comportamiento neuronal

utilizando como base el tratamiento de pulsos. En uno y otro caso la plataforma para este tipo de aplicaciones neuroinspiradas son siempre sistemas electrónicos, en su mayor parte digitales, pero con bastantes ejemplos, sobre todo en el caso de los sistemas pulsantes, de circuitos electrónicos analógicos neuroinspirados [SIVILOTTI01] [ROUSSELET02] [MORTARA02] [MORTARA01] [HIGGINS01] [MAHOWALD02]. En este trabajo nos centramos precisamente en el aspecto pulsante de este tipo de sistemas, pero utilizando sobre todo circuitos digitales que se comporten de forma parecida al sistema nervioso.

La mayoría de los sistemas pulsantes neuroinspirados que se plantean proponen mejorar alguna característica de procesado o poder hacer nuevas aplicaciones con respecto a las posibilidades de los sistemas digitales tradicionales. Cuando se pretende emular, aunque sea parcialmente, la naturaleza pulsante del sistema nervioso el primer problema al que se enfrenta el investigador es cómo codificar la información [THORPE03][THORPE01][THORPE02], hay ciertas características que se conocen de como lo hacen los seres vivos, por ejemplo, parece que a mayor estímulo más pulsos se emiten entre las neuronas, pero éste es un mundo aún por descubrir. Precisamente, un mecanismo de simplificación consiste en asociar una frecuencia de pulso a cada valor de los parámetros que intervienen en el procesado, normalmente con un esquema de modulación por frecuencia de pulso (o tiempo entre pulsos), de forma que al sistema le entran trenes de pulsos, que se tratan por parte de las neuronas, obteniéndose a la salida otros trenes de pulsos correspondientes a las especificaciones deseadas de procesado.

En la naturaleza las conexiones entre neuronas son múltiples formando una tupida red entre ellas, cuando esto se pretende emular con circuitos electrónicos esta conexión masiva se hace difícil e incluso imposible si las neuronas se encuentran en distintos chips. Para ello se ha propuesto un mecanismo de interconexión entre chips neuronales pulsantes basado en un esquema de marcado (mediante direcciones) y multiplexación de los pulsos a través de un bus digital, este esquema se denomina AER (“Address-Event Representation”) [MORTARA01][MORTARA02][BOAHEN01] [BOAHEN02][INDIVERI01][SERRANO02].

El AER intenta parecerse a la naturaleza, de forma que si una variable analógica adquiere un determinado valor, al pasarla a AER dicho valor viene representado por una frecuencia de pulso proporcional al mismo (hay otras posibilidades, en este trabajo se estudian algunas más). En el bus AER aparecería la dirección cada vez que tuviera que mostrase el pulso (evento), ya que a través del mismo se pueden transmitir otras variables correspondientes a otos elementos (multiplexación). Por tanto cabe destacar que en los sistemas AER la información siempre viene representada por dos valores: la dirección, que nos indica de qué variable o elemento se trata, y el tiempo en que se

repite la dirección (evento), que muestra el valor de dicha variable o elemento. Desde un punto de vista formal el sistema es analógico, ya que el valor se representa por un parámetro no cuantizado ni discretizado: el tiempo entre eventos. Pero dada la naturaleza pulsante (todo o nada) no es difícil apercibirse que es posible un tratamiento digital de los tiempos entre eventos y la dirección para llegar a obtener un procesado relativamente sencillo. En este caso en cuanto se asocie un valor digital a la frecuencia o tiempo entre eventos estamos cuantizando la variable, pero sin asignarle un periodo de muestreo fijo, con las consecuencias que ello trae cuando se producen cambios relativamente rápidos en las variables a representar [MORGADO01].

Utilizar sistemas AER tiene dos problemas, por una parte es muy difícil hacer un sistema basado totalmente en pulsos, siempre se precisan elementos clásicos para que el sistema pueda ser útil, normalmente computadores tradicionales, esto implica un tratamiento híbrido. Por otra parte, no existen a priori elementos con los que comprobar los resultados intermedios del procesado. Ambos problemas se solucionan con una infraestructura que permita conectar los sistemas AER con los computadores tradicionales. Desde el punto de vista del desarrollo esto permite probar la eficacia del procesado usando ordenadores personales que puedan proporcionar datos al AER o extraer datos del mismo.

Uno de los campos en los que el procesamiento AER resulta más ventajoso es el de la visión o procesado de imágenes en movimiento. Basta con comparar las posibilidades que ofrece el ojo humano (en nuestro caso artificial) con las que ofrecen los computadores digitales en este campo. Hay numerosas ventajas que podemos encontrar en el procesado de imágenes AER con respecto al tradicional, dos a tener en cuenta son: en primer lugar una cierta redundancia en la información que se transmite en aquellos aspectos que son más importantes, con lo que es precisamente lo "importante" lo que se distingue frente al resto de la información. En segundo lugar, la capacidad de transmitir y procesar imágenes de alta velocidad con un coste relativamente bajo.

Por otra parte, una de las características más destacada de estos sistemas es que no están basados en fotogramas, la información de vídeo fluye de forma continua sin estar discretizada temporalmente, sólo hay una discretización espacial al tener como elemento mínimo de sensado y tratamiento al píxel.

Este trabajo pretende proponer y evaluar diversos mecanismos para el procesado de imágenes basados en AER y por otra parte desarrollar las infraestructuras necesarias para llevarlo a cabo. Para ello se utiliza la tecnología del codiseño, basándose en FPGAs y microcontroladores, junto con ordenadores personales. Hay que señalar que se ha realizado gracias al auspicio y financiación de los proyectos de investigación: Caviar (proyecto europeo) y Samanta 1/2

(proyectos nacionales coordinados), en los que han participado entidades tan importantes como : INI (pionero a nivel mundial en los sistemas neuroinspirados) o el IMSE (líder mundial en el diseño de convolucionadores AER). En este sentido gran parte del trabajo que se presenta en esta tesis es fruto de ideas y esfuerzos colectivos desarrollados en dichos proyectos, en particular del Grupo de la USE, con el cual el autor se siente en deuda con cada uno de sus miembros y sobre todo con los mencionados en los agradecimientos, ya que sin su ayuda y aportaciones no hubiera sido posible esta tesis.

Los proyectos mencionados pretendían en general la realización de demostradores que permitieran probar las posibilidades del procesado AER, incluso llegando a realizar un sistema robótico casi en su totalidad basado en AER, llevando el sistemas de pulsos hasta los mismos motores eléctricos (Samanta 2). Dentro de estos proyectos un aspecto importante se refería al sensorial, siendo a su vez el más tratado el procesamiento visual. Los diferentes grupos participantes diseñaron una gran cantidad de elementos AER (convolucionadores, retinas, etc.) Una de las aportaciones más importantes de esta tesis a dichos proyectos ha sido la realización de parte de las infraestructuras AER necesarias para poder llevarlos a cabo.

Existen también otros ámbitos donde se están estudiando sistemas bioinspirados para el tratamiento de audio [CHAN01][OTNES01][GOMEZ04] o para implementar algoritmos de control tradicionales para el control de robots mediante sistemas pulsantes [JIMENEZ01][LINARES05] [JIMENEZ02][JIMENEZ03].

1.1 Objetivos y estructura de la tesis

El principal objetivo de esta tesis es la propuesta y evaluación de diversos mecanismos para el procesado de imágenes en movimiento AER, incluyendo el hardware necesario para su realización. Se plantea por ello la implementación de una cadena AER que sirva para ilustrar, tomar datos y contrastar los resultados que se obtengan con dicha experiencia.

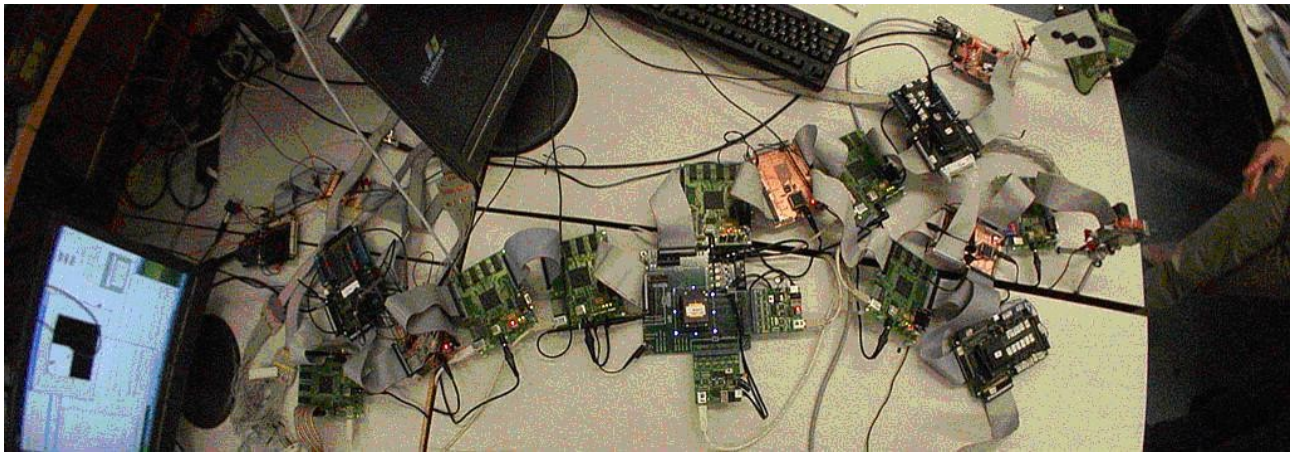

Ilustración 1.1: Cadena AER procesamiento vídeo

El procesamiento se plantea sobre una imagen o secuencia de imágenes de entrada, partiendo de un generador AER, pasando a través de una capa de neuronas que realiza alguna operación sobre el flujo AER de entrada, para ser recibido por una última capa de neuronas, que reconstruye la imagen o vídeo, tras el procesamiento, para ser mostrado o evaluado en un ordenador convencional. Por tanto, será necesario estudiar las diferentes posibilidades para cada uno de los elementos mencionados, así como realizar su evaluación. Todo esto se lleva a cabo en los capítulos que van del 3 al 7 inclusive. A parte, en el capítulo 2, se introduce el bus AER y se estudian los antecedentes. En el 8 se proponen y analizan algunas alternativas con respecto a la codificación AER (AER Diferencial, Retina Derivativa,...). El diseño del hardware necesario para poder implementar y evaluar la cadena AER se encuentra en los apéndices 1 y 2, se corresponde con el diseño de placas USB-AER y PCI-AER. Por último, en el apéndice 3 se incluyen el software y las simulaciones desarrollados para las diferentes pruebas.

Con respecto a la propuesta y evaluación de un sistema de visión basado en AER podemos distinguir los siguientes objetivos específicos en esta tesis, correspondiéndose éstos con cada uno de los elementos AER que componen la cadena de procesado y con el análisis correspondiente.

- **Objetivo 1**.- Implementar y evaluar la generación de eventos para un sistema de visión (capítulo 3). La cadena de procesado AER comienza siempre con un elemento encargado de generar eventos; estos, en nuestro caso, deben representar una imagen en movimiento (equivalente a lo que se capta en un vídeo). Aunque para la realización de pruebas es normal utilizar imágenes estáticas, hay que entender que el procesado planteado es para la visión artificial y se trata de imágenes que pueden cambiar de forma continua. Para obtener una imagen en formato AER caben las siguientes posibilidades:
    - Utilizar una retina AER, es decir una cámara expresamente diseñada para volcar datos AER. Bajo los proyectos de investigación mencionados anteriormente se han diseñado varias retinas de este tipo [LICHTSTEINER03][LICHTSTEINER02] [LICHTSTEINER04][COSTAS01] pero no es objeto de este trabajo un estudio de las mismas, simplemente se hará mención a ellas para compararlas con los dos siguientes mecanismos, los cuales si han sido tratados en esta tesis, formando una parte importante de la misma.
    - Transformar imágenes digitales en formato frames a eventos AER. Hay numerosos trabajos sobre este tema [GOMEZ02][LINARES02], en nuestro caso desarrollamos y contrastamos dos métodos el exhaustivo y random [LINARES01]. Estos métodos se caracterizan frente a otros por poder ser implementados en hardware, en una FPGA, con ahorro de recursos. Esto nos permite, entre otras cosas, utilizar un PC como fuentes de imágenes, obteniendo a partir de ellas el flujo AER. Como base para realizar esta transformación se han utilizado tanto las tarjetas PCI-AER como la USB-AER.
    - Generación a partir de una señal de vídeo. En este caso se trata de realizar la transformación directamente desde una cámara de vídeo convencional a AER, sin que la imagen originaria se encuentre en formato digital, se trata por tanto de conectar una cámara convencional directamente a la cadena AER, sin que tenga que haber un ordenador personal que mande el vídeo digital. La idea e implementación de estas seudoretinas sintéticas es crucial en el desarrollo de los sistemas AER, ya que permiten su extensión a campos muy diversos a costes relativamente bajos. Por una parte las retinas AER son caras, hay pocas y presentan ajustes relativamente complejos, debido fundamentalmente a que son desarrollos de investigación con tecnología full custom (no

se encuentra en el mercado convencional). Y , por otra parte, estas seudoretinas sintéticas, frente a las transformaciones de frames del apartado anterior, nos dan la posibilidad de eliminar el PC y dejar el sistema AER "suficientemente pequeño" como para que pueda integrarse en robots móviles u otros sistemas empotrados.

- **Objetivo 2**.- Proponer diferentes mecanismos de procesado de imágenes AER. Para probarlos se utilizará el sistema USB-AER desarrollado, ademas de la implementación, se evaluarán los resultados obtenidos (capítulo 4). El formato de imágenes AER permite un tratamiento de las mismas por mecanismos electrónicos sin interferir apenas en la cadena de transmisión de imágenes. Es decir, se puede realizar un procesado al vuelo, mientras se retransmite la imagen, con un retraso mínimo entre la señal de entrada y la de salida. Un aspecto importante de los sistemas pulsantes tratados en este trabajo es que la información es portada en tan solo dos parámetros: por una parte la dirección del evento indica el elemento que lo genera (o el destino del mismo), y por otra, el tiempo entre eventos (o frecuencia) determina el valor de la acción. Es decir, si una imagen cambiante está representada por ese sistema, el procesado de la misma consiste en variar ambos parámetros (o uno de ellos) para obtener el filtro que realice la función deseada, obteniéndose de esa forma un flujo de eventos transformados que representa la imagen filtrada. Es precisamente la simplicidad de la representación la que lleva consigo que el cambio a realizar sobre el flujo de información AER sea también "relativamente simple" con resultados, a su vez, "relativamente complejos" con imágenes en movimiento. Este aspecto es destacable, puesto que el sistema nervioso se basa en elementos y procesados aparentemente simples, pero con resultados evidentemente complejos, parece por tanto que hay un cierto grado de correspondencia; no significa esto que hayamos descubierto como opera el sistema nervioso, simplemente que nuestras propuestas utilizan esquemas paralelos al mismo. En este trabajo se estudian dos tipos de filtrados: en primer lugar, los que actúan sólo sobre la dirección, en este caso podríamos hacer cambios en la imagen que impliquen rotar, escalar, etc. En segundo lugar, los filtros que además de cambiar la dirección, actúan sobre el tiempo entre eventos, con los que se pueden realizar incluso convoluciones. Al mismo tiempo que se proponen y estudian diferentes tipos de filtros, se presentarán dos nuevos sistemas hardware: el Mapper de eventos y el Mapper Probabilístico Multievento [GOMEZ01], ambos se implementan en la placa USB-AER. El primero de ellos es un sistema bastante simple pero con limitaciones, el segundo intenta cubrir dichas limitaciones, extendiendo las posibilidades del procesado AER con sistemas de bajo coste.

- **Objetivo 3**.- Reconstrución AER (capítulo 5). El flujo de eventos AER debe reconstruirse sobre algún tipo de pantalla para que puedan ser reconocidas las imágenes en movimiento que representa (o los resultados después de filtrarlas). En trabajos anteriores [LINARES01] [MORGADO01] se utilizaba básicamente la reconstrucción mediante integradores, los cuales tenían parámetros parecidos a los de una cámara de fotos o vídeo. Así el tiempo de integración era equivalente al tiempo de exposición para cada fotograma, muestreándose la imagen con una frecuencia correspondiente a dicho tiempo de integración, de todo esto resulta una transformación de formato AER a formato de imágenes digitales basadas en frames. En este trabajo se estudia la reconstrucción de la imagen AER sin integrar durante un determinado intervalo, en este caso el valor de cada píxel se representa por el tiempo entre los dos últimos eventos más una función de "olvido". La novedad de este mecanismo viene determinada en el sentido de que no existe un frame si no que cada píxel es independiente de otro, sin tener tiempos de muestreo fijos y común a todos, lo cual parece más próximo a lo que realmente ocurre en la naturaleza. En definitiva, de lo que se se trata es de obtener la frecuencia instantánea para cada elemento (parámetro teórico) y asociarlo al valor del pixel.
- **Objetivo 4**.- Evaluar y caracterizar los errores introducidos en cada etapa de la cadena (capítulo 6). Con objeto de evaluar mejor las posibilidades de este tipo de procesado se pretende distinguir las diferentes fuentes de errores, y clasificar y cuantizar dichos errores.

2. El bus AER y los sistemas neuromórficos bioinspirados.

El término neuromórfico fue ideado por Carver Mead del California Institute of Technology de 1989 Error: No se encuentra la fuente de referenciaError: No se encuentra la fuente de referencia para describir sistemas VLSI que contienen circuitos electrónicos analógicos que imitan las arquitecturas neurobiológicas presentes en el sistema nervioso. El fin de la comunicación neuromórfica es la transmisión de la información de forma similar al producido en los sistemas neuronales biológicos.

Para establecer la comunicación entre estos sistemas biomórficos se ideó el protocolo de representación de eventos de direcciones (Address-Event Representation). Éste fue propuesto originalmente para el campo de los sistemas neuromórficos e inicialmente propuesto por Mahowald.

Address-Event-Representation *([BOAHEN02] [BOAHEN07] [BOAHEN08] [BOAHEN09] [LAZZARO02] [MAHOWALD01] [MAHOWALD02] [MAHOWALD03] [MORTARA02] [MORTARA03] [MORTARA04] [WHATLEY01] [WHATLEY02] [SIVILOTTI01])* es un mecanismo por el cual celdas de computación de distintos chips se comunican asíncronamente entre ellas. Se entiende por celdas de computación a unidades computacionales, como puede ser una neurona de silicio, un píxel de una matriz o retina, ...

Aunque la mayor parte del propósito de formalizar AER es para la comunicación entre circuitos neuromórficos, no se hacen compromisos de detalles de las unidades envueltas, de manera que su uso puede extenderse a otras aplicaciones como circuitos digitales, emulaciones software, neuronas biológicas o software estándar.

La multiplexación temporal denominada AER surgió por la necesidad de interconectar celdas de una capa de un chip con las de otras capas en otro chip, de forma que esa conexión punto a punto que existe entre las celdas se pueda simplificar al realizar sistemas neuroinspirados. Esta conexión ayuda al procesamiento de la información. Enumeremos su posible escala de complejidad:

- El uso más sencillo del AER consiste en transmitir información, de forma que un receptor reconstruye la información del emisor sin cambios. *([APSEL01])* - Dado que en el AER se transmite la dirección de los píxels, se pueden hacer transformaciones, consiguiendo que la información llegue al receptor modificada en tiempo real, como por ejemplo la realización de traslaciones o rotaciones insertando una PROM en el bus *([HIGGINS02][HIGGINS03] [HIGGINS04], [RICCI01])*.

- De forma similar, la información del bus AER puede modificarse incrementando el ancho de banda, de forma que aumente el número de celdas receptoras ante una celda emisora. Hablamos de la implementación de convoluciones insertando elementos en el bus que ante la recepción de una dirección transmitan un conjunto de direcciones siguiendo un pesado según un kernel (por ejemplo en la anchura de los pulsos), para la implementación, en tiempo real, de una convolución con la recepción de las direcciones *([GOLDBERG01] [GOLDBERG02][GOLDBERG03])*. El problema de este mecanismo es el aumento considerable de tráfico en el bus AER, que puede llegar a saturarse.
- Para solventar el problema del tráfico en el bus AER, la generación de pulsos para un vecindario de la dirección emisora, aplicando el consecuente kernel de convoluciones, puede ser implementado dentro del chip *([SERRANO01][SERRANO03][SERRANO04])*, realizándose el incremento del tráfico dentro del chip, una vez decodificada la dirección entrante, no afectando al tráfico del bus, aunque sí obligando a reducir las prestaciones, necesitándose un mayor tiempo mínimo entre direcciones en la recepción.

Según esta perspectiva, una capa debe poder comunicarse con una única capa receptora (conexión punto a punto) o con varias capas receptoras (conexión multireceptor). Y varias capas emisoras deben poder comunicarse con una (conexión multiemisor) o varias capas receptoras (multichip).

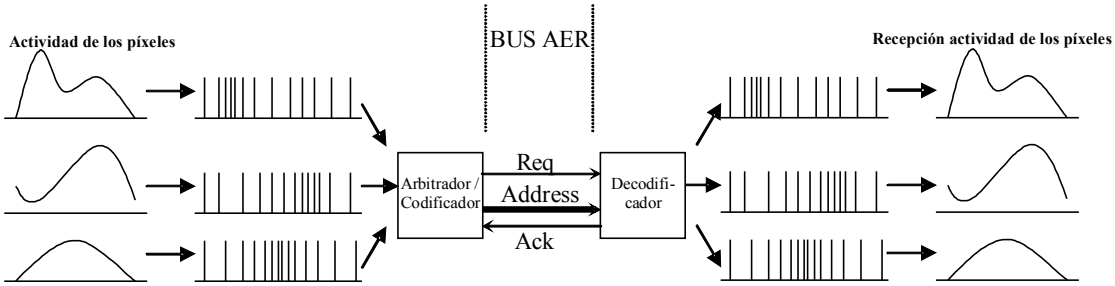

Ilustración 2.1: Escenario AER

Hasta ahora hemos comentado el AER suponiendo que la multiplexación temporal entre capas de neuronas se aplica a la unión de chips, teniendo todas las neuronas de la capa en un solo chip. Esta suposición es pobre, ya que las limitaciones que se imponen en la fabricación de chips nos empuja a la necesidad de dividir una capa de procesamiento en varios chips. Por tanto, la conectividad entre capas se complica cuando el espacio de información AER que debe recibir una

capa hay que dividirlo entre varios chips, haciendo que el resultado sea el mismo que de tener toda la capa en un chip. Esta división del espacio variará dependiendo del tipo de procesamiento que la capa receptora realice, pudiendo ser divisible el espacio en parcelas claramente disjuntas, o teniendo una frontera común para cada chip, o compartiendo completamente el espacio AER entre todos los chips.

En la actualidad existen estudios sobre una nueva versión de bus AER, en este caso utilizando transmisión serie de alta velocidad (LVDS), dado la creciente popularidad de los buses series, por las ventajas que ofrecen al trabajar con señales de muy alta frecuencia (del orden de Ghz) frente a las limitaciones de los buses paralelos (skew, crosstalking, etc) [MIRO01][MIRO02] [SERRANO05][BOAHEN03][BOAHEN04][BOAHEN05].

También hay estudios y aportaciones de aplicar la filosofía de los buses pulsantes a otros campos como la robótica, empleando el bus AER en el diseño de sistemas de control en lazo cerrado (en el control de actuadores y la realimentación con sensores)l [JIMENEZ01] [LINARES05], en conjunción con cócleas [CHAN01] u oídos AER y el procesamiento de sonidos [GOMEZ04]

2.1 Sistema AER.

Un sistema AER estará compuesto por una colección de celdas de computación que pueden enviar y recibir señales. Cada celda del sistema tiene una dirección asignada. Para enviar una señal, la celda de computación (píxel o neurona) transmite un pulso a un arbitrador, el cual asigna su dirección y la transmite mediante el protocolo asíncrono de handshake. El tiempo en el cual la dirección se transmite junto con el valor de dicha dirección determina completamente a la señal. La generación de la señal es conocida como evento. Los eventos en un sistema AER son recibidos por una o varias celdas. Cuando un evento es enviado a una celda, esto puede significar varias cosas:

- Ser la entrada sináptica de un cierto tipo de neurona neuromórfica.
- Iniciar cambios en parámetros de recepción de eventos (configuración).
- Provocar cambios en la conectividad de recepción de eventos (configuración).
- Significar una difusión del evento (broadcast) a un conjunto concreto de receptores

(convoluciones).

- Ser procesado por una interfaz gráfica para representar la actividad del sistema.
- Etc.

Cada celda podrá ser tanto emisora como receptora de eventos. Entre el transmisor y el receptor existe una maquinaria de rutado de eventos para alcanzar su correcto destino, la cual podrá estar compuesta por unidades capaces de recibir y generar o transmitir eventos. Las celdas de un sistema AER están típicamente distribuidas en forma de "cluster" o agrupamiento de celdas para realizar un procesado, como las neuronas de una capa del sistema nervioso. Los "cluster" se implementan artificialmente en uno o varios chips.

Por tanto, un sistema AER se define como una colección de emisores de eventos y receptores de eventos, generalmente distribuidos en clusters o agrupamientos, y un sistema de rutado de eventos que conecte las celdas tanto dentro del cluster como entre clusters.

2.2 Líneas del bus AER

El primer documento que intenta estandarizar el protocolo de comunicación AER es el de Mahowald y Sivilotti Error: No se encuentra la fuente de referencia en el año 1993 donde explica el protocolo de señalización asíncrono punto a punto entre un emisor y un receptor y no entra en definir la arquitectura hardware. Un documento posterior de Lazzaro y Wawrzynek Error: No se encuentra la fuente de referencia lo completa definiendo cómo se puede compartir un bus común entre varios emisores. Precisamente, uno de los retos actuales es crear un estándar AER con sus especificaciones mecánica, eléctrica, de protocolo, etc. para poder intercomunicar estos sistemas. La principal dificultad, que existe en la estandarización es que habría que acotar en rangos la velocidad, el ancho del bus, la magnitud de los pulsos así como otros aspectos como la pérdida de parte de la información cuando se diseña con arbitrador, los problemas que tiene el diseño de los filtros para reconstruir adecuadamente las señales originales, etc. Todo esto se hace evidente en el momento que distintos grupos de investigación que trabajan con este protocolo de comunicación de AER, sus diseños son incompatibles entre ellos debido a lo dicho anteriormente [MORGADO01].

El bus AER se compone de un bus de direcciones y de dos líneas de protocolo. El protocolo de intercambio de señales asíncrono entre la matriz emisora y la receptora asegura que sólo se atienda a una celda cada vez. Esta dirección se transmite por el bus a la matriz receptora donde la decodifica para activar el elemento receptor correspondiente.

- *Request (REQ)*: La controla el emisor y se usa para avisar al receptor del comienzo de la transmisión de una dirección.
- *Acknowledge(ACK)*: La controla el receptor, el cual la activará cuando haya leído la dirección del bus.
- *Bus de direcciones*: aunque su denominación confunde, es realmente por donde circulan los datos, su anchura será variable y dependerá del número total de celdas.

El comportamiento de las líneas de protocolo depende del tipo de protocolo usado, el cual dependerá de la aplicación o necesidad de conexionado de las celdas. Y el tamaño del bus puede variar en sistema AER, ya que dentro de un cluster se necesita un determinado tamaño de bus, pero al salir del cluster hay que añadir la identificación del mismo. En este capítulo se propone un tamaño del bus de direcciones para el bus AER, el cual se acordó en Telluride'2001 *([TELLURIDE01])*.

La característica principal de la AER es que se transfiere por el canal de comunicación las direcciones de los emisores activos, lo que se denomina *eventos de direcciones*. La frecuencia de trabajo de cada emisor sólo depende del estímulo recibido en él y es independiente de los demás emisores. Además por lo general, cada emisor es linealmente depende de la cantidad de estímulo recibido; a mayor estímulo, mayor frecuencia de trabajo. Esto se puede asemejar al funcionamiento de un sistema modulado en frecuencia, donde la señal moduladora será el estímulo y hará variar proporcionalmente la frecuencia de salida.

2.3 Protocolo AER punto a punto unidireccional

Surge para comunicar dos celdas entre sí, cada una residente en un "cluster" distinto. Una celda actúa de emisora y la otra de receptora. El bus en este caso no será compartido por ningún otro "cluster", de forma que cuando el arbitrador del chip emisor dé paso, el bus quedará en exclusividad para esta comunicación. *([BOAHEN07][BOAHEN08][BOAHEN09], [MAHOWALD03])* Para este protocolo se hace necesaria la existencia de un arbitrador que tome las peticiones de todas las celdas emisoras y vaya asignando el bus a cada celda de una en una. Además, el arbitrador será el encargado de poner la dirección de la celda que está haciendo la petición por medio de la salida de un codificador al que llegan las peticiones de todas las posibles celdas emisoras.

Existirá un arbitrador por cada "cluster" o agrupamiento de celdas y un bus AER de salida donde el arbitrador va dando paso a las celdas para que escriban sus eventos. Igualmente, cada "cluster" tendrá un bus AER de entrada para recibir peticiones de eventos de las celdas de otro "cluster".

El proceso a seguir cuando una celda tiene que mandar un evento será:

- Al producir un evento (spike) la celda emisora, activa su línea de salida, la cual llega a un arbitrador, que recoge el evento y le manda a la celda una señal de *Reset* para indicarle que su evento está en camino. De esta forma la celda seguirá trabajando hasta su próximo evento.
- Este arbitrador puede estar conectado con un arbitrador de un nivel superior o directamente ser el arbitrador del bus AER de salida del "cluster". En cualquier caso el arbitrador debe asignar una dirección en función de las celdas a las que está arbitrando y mandar una señal de *REQ* junto con el valor de la dirección de la celda que manda el evento.
- Se mantiene activa la señal de *REQ* y el valor de la dirección en el bus hasta que el arbitrador de destino o el arbitrador de siguiente nivel, acepten la petición y le contesten con un *ACK*.
- Cuando recibe el *ACK* retira el *REQ* y pone el bus de direcciones en alta impedancia.

- El arbitrador que recibe la petición la contesta inmediatamente y luego la propaga o bien a un arbitrador de nivel superior, si aún no hemos salido del “cluster”, o a un arbitrador de destino, si se trata del bus que comunica dos “cluster”, o bien a un arbitrador de nivel inferior si el evento está llegando a la celda destino dentro del “cluster” destino.

El sistema AER podrá tener tantos niveles de arbitradores como sea necesario para llegar a todas las celdas, cada arbitrador tendrá que añadir unos bits en la parte alta de la dirección para identificar el puerto por el que le viene la petición y así tener una dirección unívoca para cada celda. En la ilustración 2.2 puede verse un ejemplo del protocolo de handshake que acaba de ser descrito entre dos capas (punto a punto) de procesamiento AER.

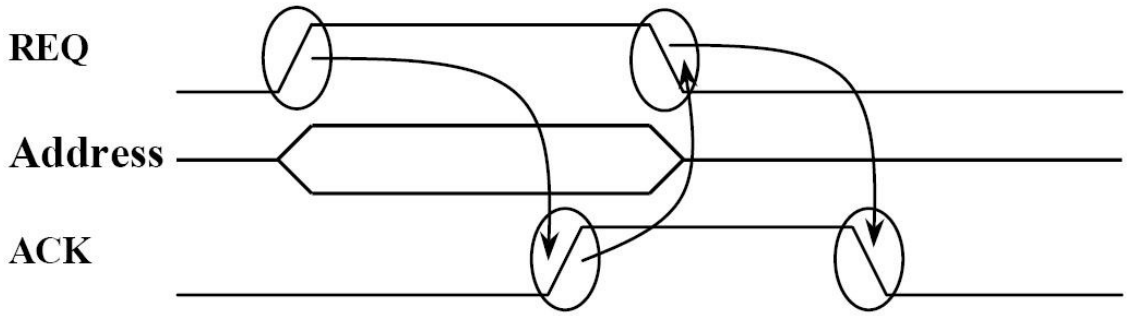

Ilustración 2.2: Protocolo handshake AER punto a punto

3. Conversión de Frames a AER: Transformación de señales de vídeo a AER.

El primer elemento de una cadena de procesado de imágenes AER es el sensor de visión con salida de eventos [BOAHEN06][KRAMER01][AZADMEHR01][HARRIS01][SHOUSHUN01] [LICHTSTEINER01][MAHOWALD02][ARIAS01][LICHTSTEINER02][LICHTSTEINER03] [BARBARO01][RUEDI01][COSTAS01]. Como se ha explicado en la introducción este tipo de dispositivos puede sustituirse por varias alternativas, una de ellas es la conversión de frames a AER y otra es la transformación de una señal de vídeo a a eventos AER. Realmente la segunda alternativa se basa en los resultados de la primera, simplemente partiendo de la digitalización de la señal de vídeo. El proceso que se describe en este capítulo pretende generar eventos AER a partir de imágenes [LINARES03][LINARES02] en movimiento en formato frames de forma similar a como lo haría una retina electrónica que imitase, a su vez, el comportamiento de una retina biológica.

En una retina biológica, una matriz de neuronas sensibles a la luz, emitirían impulsos eléctricos al ser estimuladas por la luz que reciben. Aunque se sabe que es una simplificación, podemos suponer que una célula que recibiera muy poca luz presentaría una actividad muy baja. Por otra parte, las células que recibieran mucha luz, al ser estimuladas con más intensidad, aumentan la cantidad de eventos que generarán (mayor frecuencia de eventos). A lo largo del tiempo, la cantidad de luz que recibe una determinada neurona de la retina puede variar, y por tanto la frecuencia de impulsos. Este proceso es un proceso continuo en el tiempo para cada neurona, de forma que la frecuencia de eventos para cada una de las neuronas que compondrían la retina varía de forma continua a lo largo del tiempo. Por tanto, la siguiente capa de neuronas recibirían estos eventos también de forma continua.

En una retina electrónica basada en este principio, cada elemento de la matriz de sensores recibiría una cantidad de luz, que transformada en un nivel de tensión, modularía un VCO de forma que la frecuencia de impulsos (también llamados eventos, en el bus AER), vaya en función de la cantidad de luz recibida [CULURCIELLO01]. Otra posibilidad, con la intención de reducir la cantidad de eventos presentes en el bus, es que la actividad de las neuronas no sea función directa de la luz que reciben, sino de la variación de esta luz (retina derivativa) [LICHTSTEINER03] [LICHTSTEINER02][LICHTSTEINER04][COSTAS01]. Este es el caso, por ejemplo, de una retina desarrollada en el Institute of Neuroinformatics de la Universidad ETH de Zurich, [TOBI01]. En este caso, un receptor que reciba un valor constante de luminosidad no enviará eventos, mientras

que los receptores que más variación perciban serán los que más eventos emitan. Esta variación de luminosidad puede tener dos signos: incremento o decremento de la intensidad de luz. Para codificar esto se utiliza un bit adicional, resultando en dos direcciones distintas para cada receptor [[LICHTSTEINER03]]. Más adelante se tratará el AER con signo o delta AER.

Para interconectar estas neuronas electrónicas con la siguiente capa de neuronas, utilizando un bus AER, deben ser multiplexadas por medio de un arbitrador [RIVAS01][RIVAS02]. De manera ideal, el ancho de banda del bus AER debería ser suficiente para gestionar todos los eventos generados por todas las neuronas emitiendo a su máxima frecuencia. Dado que la frecuencia de impulsos de las neuronas biológicas es del orden de KHz, frente a los MHz del bus AER, es un objetivo alcanzable para algunos miles de neuronas.

A diferencia de estos sistemas bioinspirados, los dispositivos electrónicos actuales que trabajan con imágenes en movimiento, como cámaras de vídeo, televisores, monitores VGA, etc., funcionan de manera muy diferente. Las imágenes capturadas por una cámara de televisión son concretizadas en imágenes fijas, que se suceden en el tiempo a una frecuencia determinada. En los sistemas de televisión tradicionales, el número de imágenes por segundo es de 25 (PAL/SECAM) o 30 (NTSC). En las tarjetas gráficas y los monitores de ordenador actuales, el número es mayor (60-120). Existen cámaras de vídeo de alta velocidad donde el número de frames puede ser de varios miles de imágenes por segundo. Cada una de estas imágenes se recorre fila a fila secuencialmente, de manera que la información de luminosidad para un determinado píxel cualquiera se muestrearía a esta velocidad.

Debido a estas diferencias en el procesado, las características de las imágenes que pueden representarse en ambos sistemas difiere. Al discretizar el movimiento en frames, el sistema no puede representar variaciones de luminosidad (objeto en movimiento, luz parpadeante, etc.) cuya frecuencia sea mayor a la frecuencia de discretización. Si nos fijamos en el sistema PAL, por ejemplo, un objeto que se encendiese y apagase a una frecuencia superior a 25 Hz no podría observarse con los cambios tal cual suceden, se vería encendido pero con menor intensidad. Por el contrario, en un sistema bioinspirado esta limitación no es tan estricta. La limitación de velocidad para un determinado píxel dependería de la frecuencia de eventos que se generase para ambos niveles de iluminación[MORGADO01].

A pesar de las diferencias entre ambos sistemas, resulta interesante poder interconectarlos para poder desarrollar y depurar dispositivos bioinspirados, utilizando la gran variedad de dispositivos digitales existentes, cuyo coste es muy reducido (webcams, etc).

En el caso más general, la generación AER no se limita a imágenes, sino que puede utilizarse en cualquier escenario donde un conjunto de neuronas se conecten con otra capa de neuronas utilizando el bus AER. Para una única neurona, los eventos generados por ésta pasarían directamente al bus sin necesidad de arbitración al no existir otros elementos. Al aumentar el número de neuronas es necesario compartir el bus entre las diferentes neuronas emisoras. El hardware asignado a cada una de ellas podría ir generando eventos que pasarían al bus por medio de un arbitrador, que se encarga de resolver las colisiones. Conforme el número de neuronas aumenta, el tamaño del hardware asociado también aumenta. En el caso de imágenes, incluso de pequeño tamaño, como las empleadas en la mayoría de los ejemplos de 64x64, serían necesarias 4096 neuronas emitiendo eventos. La complejidad requerida por el arbitrador para combinar tantas neuronas, junto al excesivo tamaño que representaría replicar la lógica de una neurona más de cuatro mil veces, hace necesaria una simplificación: Para la conversión de imágenes a AER, el valor de las neuronas se almacenaría en una memoria RAM, que suelen venir integradas dentro de la mayoría de FPGA, y la lógica de la neurona se implementa una sola vez, fusionándose con la arbitración

El proceso de generación consistiría entonces en ir accediendo a todas las posiciones de memoria donde se almacenan las neuronas, y de manera secuencial ir generando los eventos correspondientes a las neuronas accedidas en función de su valor en RAM. Se consulta pues el estado de una neurona en un slot de tiempo determinado, resultante de dividir el tiempo entre todas las neuronas, y en ese slot de tiempo decidir si debe o no emitir, en lugar de emitir todas a la vez, y que el arbitrador, mediante un codificador de prioridad, o recorriendo secuencialmente las distintas entradas vaya dándoles salida. El algoritmo seguido para recorrer las diferentes neuronas, así como el utilizado para repartir los eventos de una neurona en el tiempo, da lugar a los distintos métodos de generación de eventos [LINARES01][LINARES02][GOMEZ02].

3.1 Función de transferencia

En la conversión de imágenes a eventos AER, el primer parámetro que hay que definir sería la equivalencia entre un nivel de gris determinado y la frecuencia con que esa neurona emitiría impulsos o eventos. Una neurona biológica se conoce que en estado excitado puede emitir impulsos eléctricos a una frecuencia máxima del orden de KHz, y que su comportamiento no es lineal. Una vez superado el umbral mínimo de activación, la neurona comenzaría a emitir impulsos. Conforme el estimulo aumenta, también aumenta la frecuencia de los eventos de salida. Sin embargo esta progresión en lugar de ser lineal se asemeja más a una función logarítmica, donde la pendiente de la curva iría decreciendo al ir aumentando el estimulo. En una neurona digital, el perceptrón, se emplean diferentes funciones de transferencia, lineales, logarítmicas, y sobre todo, la sigmoidea.

Cualquier función matemática continua puede utilizarse como función de transferencia. La más utilizada será la función lineal donde la salida es proporcional a la entrada:

$$f(x) = k * x = \frac{f_{max}}{x_{max}} * x \tag{3.1}$$

La constante k, constante de proporcionalidad indica la relación entre un valor de entrada y el número de eventos a la salida. Cuando el valor del píxel es cero, el número de eventos en el bus es cero también, mientras que cuando el píxel tiene su valor máximo, la frecuencia de eventos en el bus será f_{max}.

Además de esta función lineal, puede ser interesante estudiar qué representarían otros tipos de curva y cómo afectarían a la transmisión y reconstrucción AER. A modo de ejemplo proponemos la siguiente función exponencial, mostrada en la ecuación 3.2.

$$f(x) = f_{max} * \frac{e^{k'*x} - 1}{e^{k'*x_{max}} - 1} = f_{max} * \frac{e^{k*\frac{x}{x_{max}}} - 1}{e^{k} - 1} \tag{3.2}$$

En este caso f_{max} sigue definiendo la frecuencia máxima de eventos, cuando la luminosidad del píxel es máxima, mientras que k define la curva de dicha función exponencial. Esta constante k=k'*xmax se define para normalizar el valor de k' y hacerlo independiente de el número de niveles de gris a representar. Cuando el valor de k es positivo, la curva es convexa mientras que para

valores negativos es cóncava. En valores muy cercanos al cero, la función se parece a una función lineal, aunque no esta definida para k=0 Cuando k=0 esta expresión no es válida, y se utilizaría la ecuación lineal 3.1 en su lugar. En las siguientes ilustraciones vemos ambas curvas.

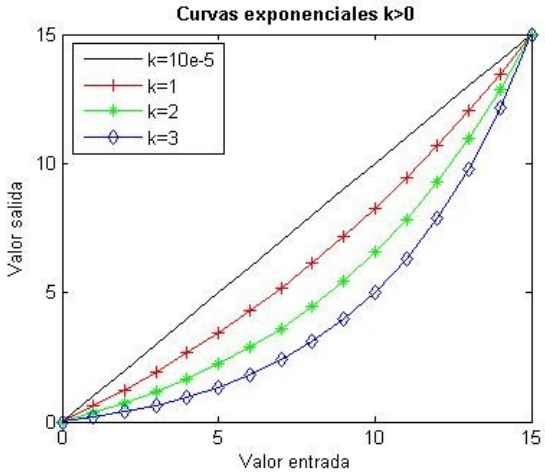

Ilustración 3.1: Exponencial positiva

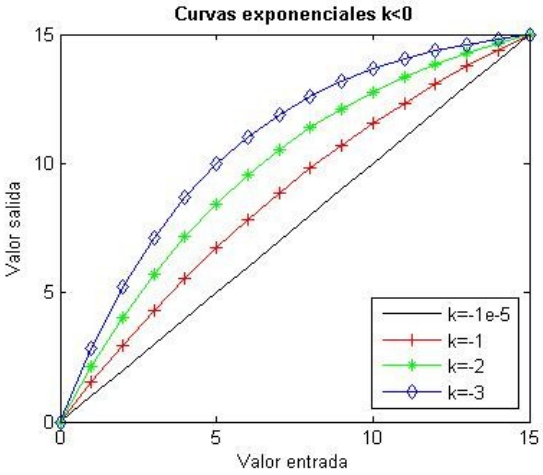

Ilustración 3.2: Exponencial negativa

En la ilustración 3.1 vemos la función de transferencia para varios valores de k positivos. Vemos que cuando k se aproxima a cero, la función se asemeja a una función lineal, y cuando k va aumentando, la curvatura aumenta haciéndose más convexa. Para el caso de las imágenes AER, esta función de transferencia permitiría en entornos con mucha luz, distinguir mejor los matices muy iluminados de la escena, mientras que en los píxels poco iluminados la imagen de salida sería aún más oscura y con menor separación entre los distintos niveles.

Para la ilustración 3.2 tenemos el caso contrario, cuando k es negativa y la función de transferencia se vuelve cóncava. En este caso el efecto sobre la imagen es el contrario. Los píxels poco iluminados se realzan, aumentando su separación, y por tanto su contraste, mientras que las partes muy iluminadas se saturarían en el blanco.

Para la conversión de imágenes en eventos AER, la función de transferencia que resulta interesante es la lineal. Cualquier otro tipo de función de transferencia sería fácilmente implementable en software, realizando un preprocesado software sobre la imagen a transmitir, como un paso previo antes de convertirla en una secuencia de eventos AER por hardware, sabiendo que el hardware tendrá un comportamiento lineal.

Por otro lado, centrándonos en una función de transferencia lineal entre el valor de la luminosidad y el número o frecuencia de eventos, se puede definir la siguiente función de

transferencia del sistema con la ecuación que define a una recta, como vemos en la ecuación 3.3, que se diferencia de la ecuación 3.1 en que añadimos una f_{min} asociada al nivel de gris cero. Para el nivel de gris máximo, la frecuencia de eventos de salida será f_{max}.

$$f(x) = \frac{f_{max} - f_{min}}{x_{max}} * x + f_{min} \tag{3.3}$$

Cuando fmin sea cero, la función se reduce a la ecuación 3.1. Para el valor de gris 0, no se emitirían eventos (estado de reposo). Conforme el nivel de gris va aumentando la frecuencia también lo hace en la misma proporción hasta alcanzar el valor de luminosidad máximo fmax. Esta fórmula describe una modulación en frecuencia, como era de esperar dada la naturaleza del AER [MORGADO01].

$$Num_{eventos} = K * Nivel_{gris} + Offset \tag{3.4}$$

La constante *K* determina la pendiente de dicha recta y la constante *Offset* determinaría el desplazamiento de la frecuencia asociada al nivel de gris nulo (0). *Offset* se asemeja con el concepto de portadora, mientras que la constante *K* define la "profundidad de modulación", o cuánto se alejará el sistema de su frecuencia portadora. Para radio, esta desviación se produce a ambos lados del espectro. Sin embargo, si consideramos que el nivel de gris siempre será mayor o igual a cero, *Offset* define la frecuencia mínima cuando la entrada sea 0, e irá incrementando hasta un valor máximo dependiente del valor máximo de entrada. Este valor podría, de forma teórica, considerarse normalizado a 1, de forma que el valor de entrada variará en el intervalo [0,1], pero como para sistemas digitales resulta más sencillo trabajar con números enteros, normalmente se utilizará un valor potencia de 2 ([0, $2^{numbits}$-1]).

Para el caso particular de una imagen, el nivel de gris no tiene sentido para valores negativos, por lo que siempre será mayor que cero. Esto no siempre será así. Si por ejemplo se tratase de un "tímpano" que escuchase sonidos, o una retina derivativa, los valores que pueden tomar en ese caso podrán ser positivos y negativos. En este caso *Offset* permitiría ajustar la función de transferencia, de forma que para cualquier valor de entrada, el número de eventos sea siempre positivo. No tendría sentido físico un número negativo de eventos. Más adelante se verá una modificación del bus AER que incorpora eventos negativos, que pueden usarse como alternativa a aplicar este offset.

En la siguiente gráfica se muestra la relación eventos/nivel de gris para dieciséis niveles de

gris (4 bits), para distintos valores de *K* (K=1, 2) y *Offset* (0 y 10).

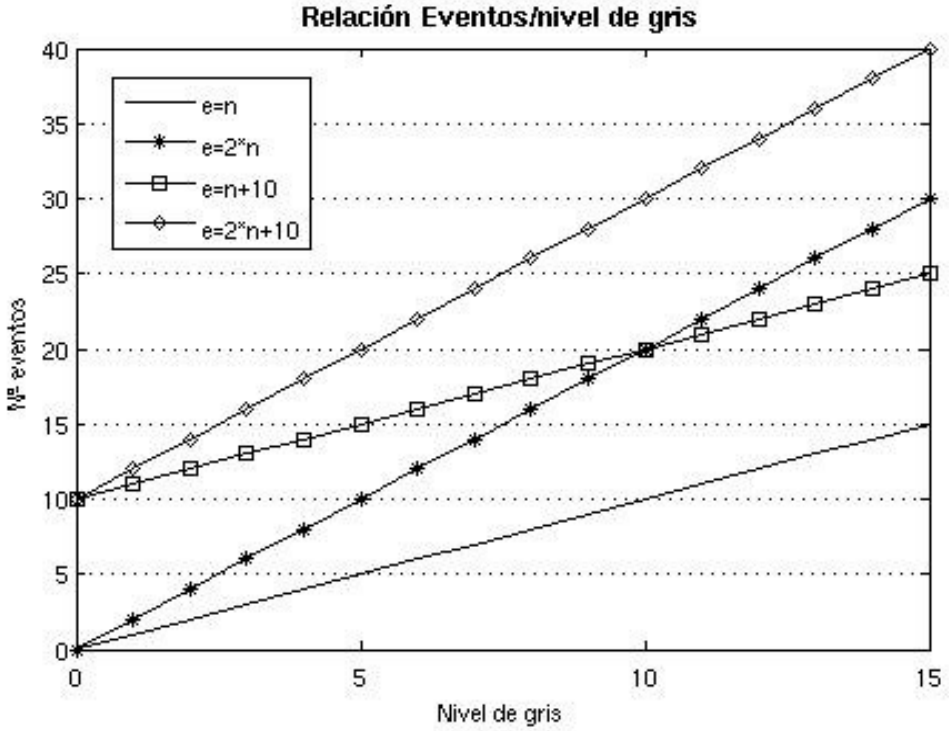

Ilustración 3.3: Relación eventos/nivel de gris

Para un píxel concreto, de manera teórica, se calcula que la frecuencia con que aparece el evento correspondiente sería:

$$Freq_{pixel} = \frac{Num_{eventos}}{T_{FRAME}} = Num_{eventos} * Freq_{FRAME} \tag{3.5}$$

Y para un píxel determinado sería

$$Freq_{pixel} = K * (nivel_{gris} + \frac{offset}{K}) * F_{frame} = K * (nivel_{gris} + offset') * F_{frame} \tag{3.6}$$

3.2 Generación AER por hardware

Los algoritmos de generación AER por hardware se han implementado utilizando el hardware de la tarjeta USB-AER. Para almacenar la imagen que se va a emitir se ha utilizado una memoria RAM de doble puerto contenida internamente en la FPGA para permitir que la imagen pueda actualizarse simultáneamente mientras el generador de eventos la recorre ininterrumpidamente generando los eventos AER correspondientes. De esta forma, actualizándose periódicamente la imagen puede extenderse la generación de eventos a partir de imágenes estáticas para enviar secuencias de vídeo.

Estas secuencias de vídeo pueden provenir del puerto USB, enviadas desde un ordenador a partir de alguna fuente de vídeo (webcam, fichero de vídeo, etc.) o desde alguna fuente de vídeo compuesto (cámara vídeo, grabadora de vídeo, DVD, etc.). Para esto último se añade un ADC al puerto de entrada, y se utiliza un diseño modificado del generador que a partir de los datos digitales provenientes del ADC, detecte los sincronismos, y guarde en memoria una porción de la imagen para transmitir.

Para generar los eventos a partir de una imagen almacenada en RAM, el mecanismo consiste en recorrer de alguna forma todos los píxels de la imagen, y en función de su nivel de gris determinar si su dirección debe salir al bus como un evento o no. El mecanismo por el cual se recorre la imagen, así como decidir si se emite o no un evento pueden variar según el mecanismo de generación AER empleado. De los diferentes métodos propuestos y estudiados [LINARES01] [LINARES02] , a saber 'Scan', 'Uniform', 'Random', 'Random-Square' y 'Exhaustive', En los métodos “scan” y “exhaustivo” la imagen se recorre secuencialmente. Un recorrido completo de la imagen corresponde a un “slice”. Varios “slice” (tantos como niveles de gris posibles) se agrupan para formar un “frame”. Para decidir si un evento se emite o no se compara el nivel de gris con el número del slice dentro del frame en cuestión. Para el método “random” sin embargo será un generador pseudoaleatorio el que genere de forma aleatoria la dirección del píxel y decida con una probabilidad proporcional al nivel de gris si se emite o no.

El esquema general de un generador AER se describe en la ilustración 3.4:

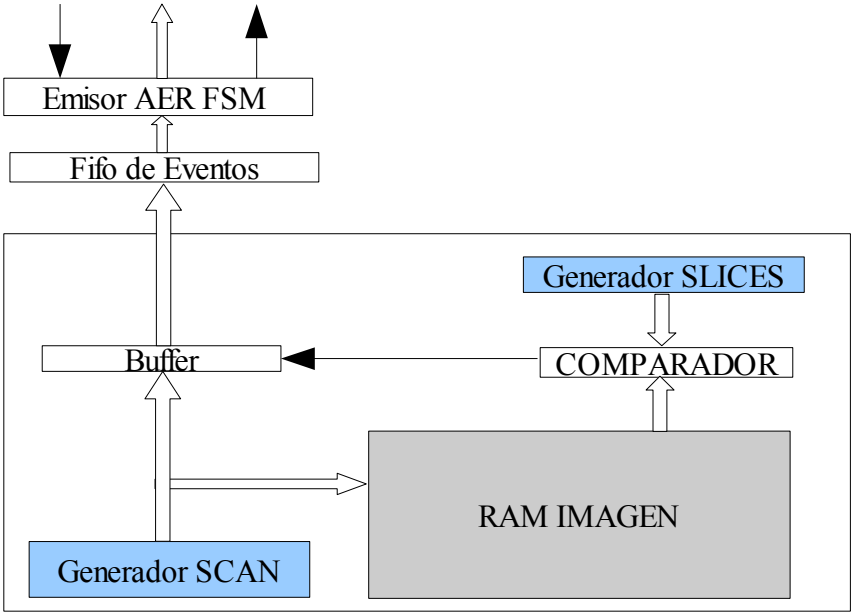

Ilustración 3.4: Generador genérico

Como se puede ver en la ilustración 3.4, el generador de "scan" y el generador de "slices" se encargan de ir generando las direcciones que accederán a los diferentes píxels de la imagen en RAM (el generador de "scan"), y las diferentes "slices" o barridos completos de la imagen necesarios para generar un frame completo. El método por el cual generar ambas direcciones puede modificarse con el fin de variar la distribución en el tiempo de los eventos que se van a generar. Un comparador, determina en función del valor del píxel y del slice en que se encuentre, si la dirección del píxel pasará a una memoria FIFO de eventos para ser transmitido al bus AER. De esto se encarga el módulo "Emisor AER FSM". Es una máquina de estados que se encarga de sacar los eventos de dicha memoria FIFO, y transmitirlos al bus AER siguiendo el protocolo de handshake asíncrono utilizando las señales REQ y ACK ("request" y "acknowledge" respectivamente).

Al implementar un método de conversión a AER a partir de imágenes en hardware, hay que tener en cuenta algunas consideraciones a la hora de elegir ciertos parámetros. El tamaño de la imagen debe ser potencia de dos. Se necesitarán n bits para acceder a 2^n posiciones de memoria. Si el tamaño de la imagen no es potencia de dos, habría direcciones de n bits que no corresponderían a ninguna posición de la imagen. Esto se puede observar en la implementación del método "random", en el que se utiliza un LFSR [MITA01][GOLOMB01][MASSEY01][CLEMENTE01], el cual generará todas las combinaciones posibles para n bits que tienen que tener su correspondencia con los píxels de la imagen.

Si tenemos N posibles niveles de gris, para un nivel de gris 0 deseamos no emita ningún evento, mientras que para un valor N-1 máximo emita en todos, por lo que el número de slices deberá ir de 1 a N-1. Se emitirán eventos cuando se cumpla la condición: “Pixel ≥ Slice” (o Pixel > Slice si el slice varía entre 0 y N-2). El número de slices no tiene por que ser consecutivo.

3.3 Método exhaustivo-bitwise

El método exhaustivo [GOMEZ02][LINARES01]intenta distribuir de manera homogénea los eventos en el tiempo. Para una distribución homogénea exacta, la separación entre dos eventos consecutivos de un mismo píxel, vendrá dada por la expresión:

$$T = \frac{T_{frame}}{Valor_{pixel}} \tag{3.7}$$

Cada tiempo T deberá emitirse un evento para dicho píxel. Si se tiene un contador de tiempo, cada vez que mod(contador ,T) == 0, o dicho de otro modo, cada "T" ciclos se emitirá un evento. El método exhaustivo-bitwise es una simplificación de este procedimiento que elimina la necesidad de calcular la división, entre el tiempo de frame y el valor del pixel, y posteriormente el módulo entre el contador de reloj y el valor del periodo calculado.

Según el método bitwise, cada evento se emitirá cuando el valor del contador sea menor que el valor del píxel. De esta forma se garantiza que el número de eventos emitidos para cada valor de entrada es el correcto. Para evitar la acumulación de eventos al principio del frame, se invierte en orden de los bits. Si suponemos un caso muy reducido en que sólo fueran posible 8 niveles de gris, tendríamos un contador de 3 bits para la división del frame en slices.

Para ello, se irá recorriendo la imagen fila a fila, y emitiendo o no en función del número del slice. Se intentarán distribuir homogéneamente en el tiempo los eventos de cada uno de los píxels. Si se utilizara un contador para el número de slice, los eventos se acumularían en los slices bajos, mientras que los slices altos estarían ocupados únicamente por los eventos correspondientes a los píxels más luminosos. Para intentar distribuirlos de manera homogénea, se invierte el orden de los bits del contador de slice. En la tabla 3.1 se observa la secuencia de emisión de eventos para esta implementación del método exhaustivo, donde se aprecia el órden en que se pregunta por los eventos de un slice para ser emitidos o no. En gris se representan los emitidos y en blanco los no emitidos.

	Contador de “slices”							
Gray	0	1	2	3	4	5	6	7
0	0	4	2	6	1	5	3	7
1	0	4	2	6	1	5	3	7
2	0	4	2	6	1	5	3	7
3	0	4	2	6	1	5	3	7
4	0	4	2	6	1	5	3	7
5	0	4	2	6	1	5	3	7
6	0	4	2	6	1	5	3	7
7	0	4	2	6	1	5	3	7

Tabla 3.1: Secuencia de emisión de eventos para implementación bitwise

El pseudo-código que describiría el algoritmo sería el siguiente:

```
for slice=0:max_level-1
	for fila=0:filas
		for col=0:cols
			if (imagen(fila,col)> invertir(slice)) then
				emitir(fila,col);
			end if;
		end;
	end;
end;
```

En la implementación VHDL, los tres bucles se han sustituido por un único contador de 20 bits Al ser la imagen de 64x64 necesitamos 6 bits para la fila y 6 para la columna. Los 12 bits menos significativos del contador se utilizan para direccionar la RAM, y el dato que sale de la RAM se compara con los 8 más significativos del contador invertidos, y si es mayor se escribe en la FIFO.

```
icounter: process(rst_n, clk)
begin
if (rst_n = '0') then
	contador <= (others=>'0');
	dcontador <= (others =>'1');
elsif(CLK'event and CLK='1') then
	if (enable ='1') then
		contador <= contador	+ 1;
		dcontador <= contador;
```

```
	end if;
end if;
end process;

evento <= dcontador(RAM_Addr_Size -1 downto 0);
iaddress <= contador (RAM_Addr_Size -1 downto 0);

invertir: process(dcontador)
begin
for i in 0 to RAM_Data_Size-1 loop
	numitera(RAM_Data_Size-1-i) <= dcontador (RAM_Addr_Size+i);
end loop;
end process;

exaustivo: process(enable, contador, image_out, numitera,dcontador)
begin
	if(enable = '1') then
		maddress <= contador (RAM_Addr_Size -1 downto 0);
		if (image_out>numitera) then
			emite <='1';
		else
			emite <= '0';
		end if;
	else
		maddress <= dcontador(RAM_Addr_Size -1 downto 0);
		emite <= '0';
	end if;
end process;
```

Ilustración 3.5: Imagen generada método bitwise

3.3.1 Error introducido por el método

Tenemos que para un píxel determinado, el tiempo ideal que separaría dos eventos consecutivos del mismo píxel:

$$T_{ideal} = \frac{T_{frame}}{N_{gris}} \tag{3.8}$$

Al dividir el tiempo de frame en slices estamos discretizando la distancia entre eventos de forma que tiene que ser múltiplo entero del tiempo de slice. Estaremos cometiendo un error en la posición del evento menor o igual a la mitad del tiempo del slice. En promedio, el tiempo medio entre eventos tenderá al tiempo ideal, por lo que el error de promedio tenderá a cero. Sin embargo, nos interesa una medida del error que nos de una medida del error que se comete individualmente.

$$Error_{absoluto} = \frac{\sum |T_{ideal} - T_{real}|}{N_{MAX}} \tag{3.9}$$

Si aproximamos la posición de los eventos situándolos en el slice más cercano, tenemos el menor error teórico posible que cometemos al dividir el frame en slices sucesivos. En la gráfica siguiente se ha representado esta distribución "óptima", junto al que se cometería en la distribución "bitwise".

El error se incrementa proporcionalmente al nivel de gris debido a que se trata de un error absoluto. La división entre N_{max} se realiza para normalizar el error. Sin embargo, para niveles de gris altos, aunque el error sea pequeño, al haber más eventos, el error total que se comete es mayor.

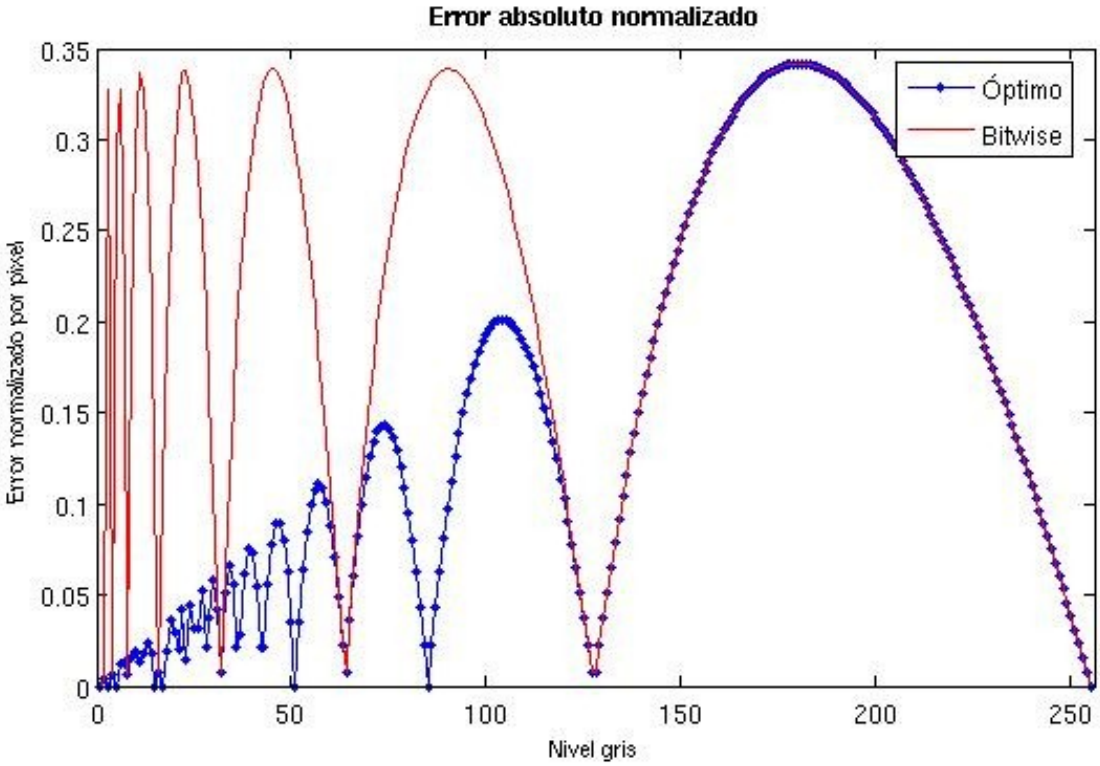

Ilustración 3.6: Error absoluto normalizado

Esta gráfica puede contrastarse con su equivalente obtenida a partir de datos prácticos publicada en el IWANN,[GOMEZ01], donde se obtenían resultados empíricos similares a los calculados aquí de manera teórica por medio de simulaciones.

Para conocer el error que se comete para cada evento, en lugar de para un píxel, se divide el error de cada píxel, por el número de eventos que lo componen. De esta manera tenemos el error relativo normalizado para cada píxel:

$$E_{rel} = \frac{\sum |T_{ideal} - T_{real}|}{N_{eventos} * N_{MAX}} \tag{3.10}$$

El error relativo de cada evento en la distribución óptima es inferior a la mitad del tiempo de slot. Al normalizar, tenemos que:

$$E_{MAX} = \frac{1}{2 * N_{MAX}} \tag{3.11}$$

Por lo tanto, para 256 niveles de gris, el error relativo normalizado para el método óptimo es inferior a 1/512 = 0,002, inferior al 0,2%. El método "bitwise" sin embargo comete un error relativo elevado (10,9% para el nivel de gris 3), aunque la amplitud de estos picos de error máximo decrece de forma exponencial conforme el número de eventos aumenta. En promedio, sólo hay 19 niveles de gris (7% del total) que presenta un error relativo superior al 1%.

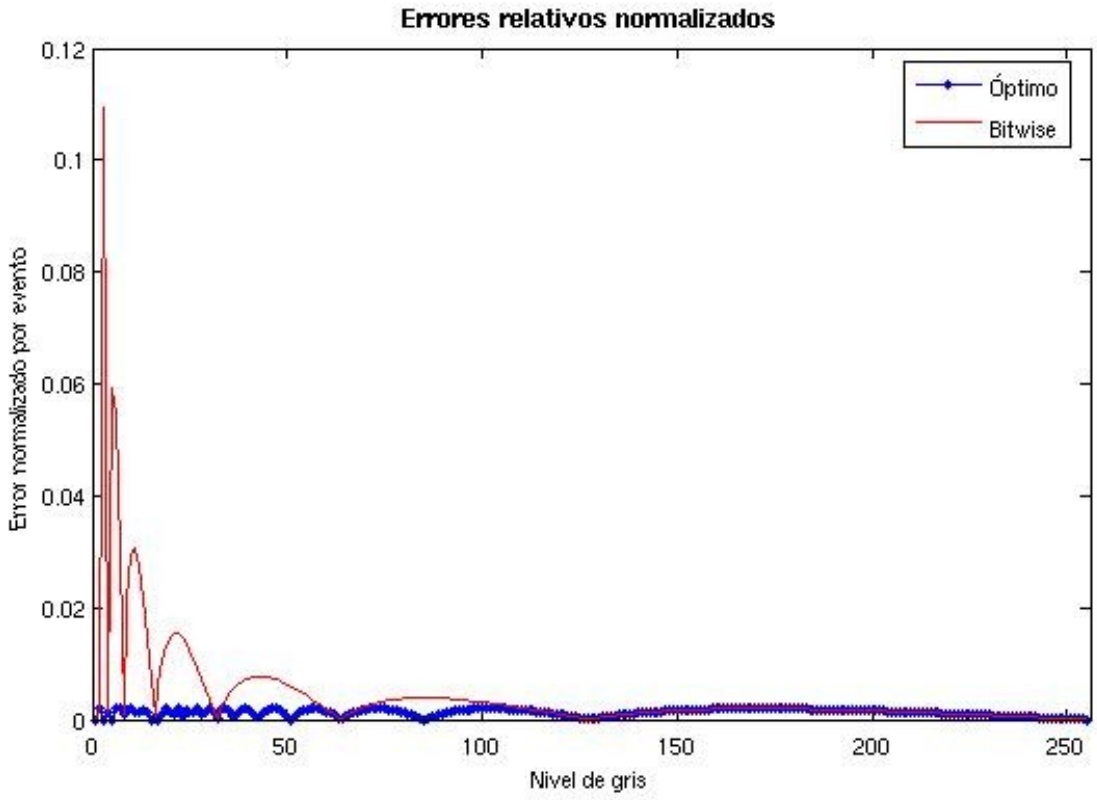

Ilustración 3.7: Error relativo normalizado

3.4 Método random

En el método random, se escoge un píxel al azar, y ese píxel tendrá una probabilidad proporcional a su nivel de gris de ser emitido o no. En promedio, todos los píxels tienen la misma probabilidad de ser accedidos.

Para implementar el generador de números pseudoaleatorios se utiliza un registro LFSR (Linear Feedback Shift Register) [MITA01][GOLOMB01][MASSEY01][CLEMENTE01]. Los circuitos LFSR se utilizan en infinidad de sistemas electrónicos para generar secuencias pseudoaleatorias (por ejemplo, en aplicaciones criptográficas) de forma simple y eficiente. Se construye utilizando un registro de desplazamiento cuyo bit entrada es una combinación lineal (módulo 2), o dicho de otra forma, la función XOR, de 2 o más bits del registro. El número y posición de bits implicados en la generación del nuevo bit de entrada se especifica utilizando polinomio de grado n, donde n es el número de bits del registro de desplazamiento. La secuencia que se genera tiene un periodo de repetición, y vuelve a repetirse indefinidamente. La longitud de este periodo depende del polinomio utilizado en la realimentación.

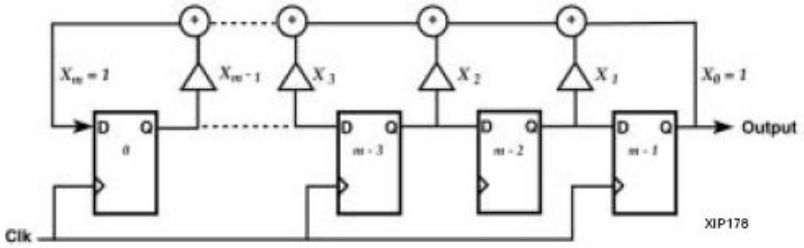

Ilustración 3.8: Implementación genérica LFSR

Dado que el estado interno de un registro de desplazamiento tiene **m** bits, la longitud máxima de la salida de un LFSR es de 2^m-1 bits, ya que el estado *"todo ceros"* no es válido (genera una salida continua a cero). Cuando el polinomio escogido proporciona el periodo máximo se dice que es un polinomio primitivo. Para que el polinomio sea primitivo, tiene que cumplir ciertas condiciones, como no ser reducible, sus coeficientes tienen que ser coprimos (primos entre si). Cada uno de los coeficientes del polinomio se corresponde con un tap o realimentación en un bit determinado. La secuencia generada será máxima únicamente si el número de taps que la realimentan es par. El número de bits que influyen en la realimentación será par en todos los

polinomios primitivos. Para un LFSR de tamaño m, puede haber varios polinomios primitivos distintos. El número de polinomios primitivos existentes con un tamaño de m bits viene dado por la siguiente expresión matemática:

$$\frac{\Phi(2^m-1)}{m} \tag{3.12}$$

Siendo phi, la función de euler, que devuelve el número de enteros menores que su argumento que son coprimos con su argumento. Además, si un polinomio es primitivo, si se escribe al revés el polinomio resultante también será primitivo. Como ejemplo, el siguiente polinomio 3.13 (representado por la secuencia de bits "`100011101`"). y el resultante de invertir el orden de sus elementos 3.14 ("`101110001` ") también lo será:

$$P(x)=x^8+x^4+x^3+x^2+1 \tag{3.13}$$

$$P(x)=x^8+x^6+x^5+x^4+1 \tag{3.14}$$

No existe una fórmula sencilla para obtener polinomios primitivos de grado arbitrario, recurriéndose típicamente a manuales de referencia con tablas o a búsqueda exhaustiva mediante programas informáticos. El siguiente código escrito para matlab devuelve los polinomios primitivos (generan una secuencia de longitud máxima) existentes para una determinada longitud de bits:

```
function allpols=polinomios_primitivos(m)
      rango=(2^m)-1;
      allpols=[];
      p=2^(m+1);
      for i =(2^m)+1:2:2^(m+1)-1
            v=2;
            for j =0:rango-2
                  v=v*2;
                  if (v>=p)
                        v=bitxor(v,2*i);
                  end
                  if (v==2)
                        break;
                  end
            end
            if (v~=2)
                  allpols=[allpols; dec2bin(i)];
            end
      end
end
```

De todos los polinomios primitivos de grado m posibles, nos interesan aquellos que son mínimos. Dicho de otra forma, aquellos que tienen menor número de sumandos, que representarán menos taps de realimentación, y por tanto más fáciles de implementar en hardware. Por ejemplo, para una longitud del LFSR de 6 bits, tendremos los siguientes polinomios primitivos posibles, que darán una longitud de secuencia de 63 palabras.

Binario	**Polinomio**
1000011	$P(x)=x^6+x^1+1$
1011011	$P(x)=x^6+x^4+x^3+x^1+1$
1100001	$P(x)=x^6+x^5+1$
1100111	$P(x)=x^6+x^5+x^2+x^1+1$
1101101	$P(x)=x^6+x^5+x^3+x^2+1$
1110011	$P(x)=x^6+x^5+x^4+x^1+1$

Tabla 3.2: Polinomios primitivos de 6 bits

Vemos que todos los polinomios tienen un número par (2 o 4), y que de las seis soluciones posibles, observamos que tres de ellas son el resultado de reescribir las otras tres al revés. De entre estos polinomios, se escogerá alguno de los dos que sólo tienen dos sumandos en x.

El registro de desplazamiento tiene que iniciarse con un valor distinto a todos los bits cero si se utilizan puertas XOR, o distinto a uno para puertas XNOR. Existen diferentes polinomios que describen el LFSR en función de su tamaño. En lugar de usar un LFSR diferente para el generador de slices y el generador de scan, se utiliza el mismo LFSR, utilizando parte de sus bits para el scan, y otros para el slice. El LFSR deberá ser de un tamaño igual o mayor al número de bits necesarios para el scan y el slice. Hay que tener en cuenta que el LFSR generará todos los valores posibles salvo todos los bits a cero.

En la tabla 3.3 puede observarse la secuencia generada por el siguiente polinomio de grado 3, que recorre los siete (2^3-1) estados de la secuencia.

$$P(x)=x^3+x^1+1 \tag{3.15}$$

Q2	*Q1*	*Q0*
1	1	1
1	0	1
0	0	1
0	1	0
1	0	0
0	1	1
1	1	0

Tabla 3.3: Ejemplo registro LFSR de 3 bits

En la ilustración 3.9 vemos el registro de desplazamiento equivalente el siguiente polinomio de grado 20 expresado en la ecuación 3.16.

$$P(x)=x^{20}+x^{17}+1 \tag{3.16}$$

Puede verse que cada elemento del polinomio se corresponde con un tap de realimentación, a través de varias operaciones lógicas XOR.

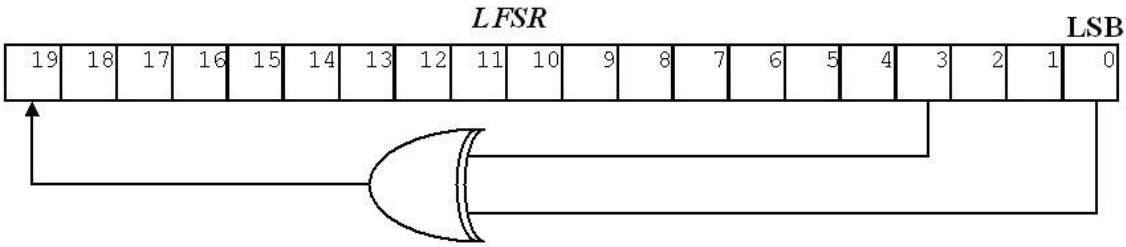

Ilustración 3.9: LFSR equivalente de grado 20

Existen estudios [LINARES04] sobre la implementación en hardware del método random, utilizando un LFSR, y cómo los eventos así generados siguen una distribución de Poisson. En los sistemas neuroinspirados, las señales pueden ser modeladas frecuentemente con una distribución de Poisson. Una distribución de Poisson se define por la siguiente función:

$$P_n(T)=\frac{(\lambda T)^n}{n!}e^{-\lambda T} \tag{3.17}$$

Donde P es la probabilidad de tener n eventos con un tiempo entre eventos determinado. La distribución del ISI (Inter-Spike Interval) con la probabilidad de que no haya eventos en el intervalo (n=0) sería la expresada en la ecuación 3.18, que coincide con una función exponencial:

$$P_0(T) = e^{-\lambda T} \tag{3.18}$$

Se ha utilizado una tarjeta USB-AER para implementar el método random, configurándola con diferentes niveles de gris para un píxel fijo. Para capturar los eventos se utiliza una tarjeta PCI-AER [RPAZ01][RPAZ02](apéndice 11.2) que captura los eventos generados y su timestamp asociado (información sobre el instante de tiempo en el que se recibió el evento) para procesarlos posteriormente en MATLAB. La ilustración 3.10 muestra la probabilidad acumulada de la distribución de los "Inter-Spike Intervals" medidas de la generación mediante del método random para los niveles de gris entre 50 y 255, con incrementos de 10 en 10, contrastada con la exponencial teórica de expresada anteriormente en la ecuación 3.18.

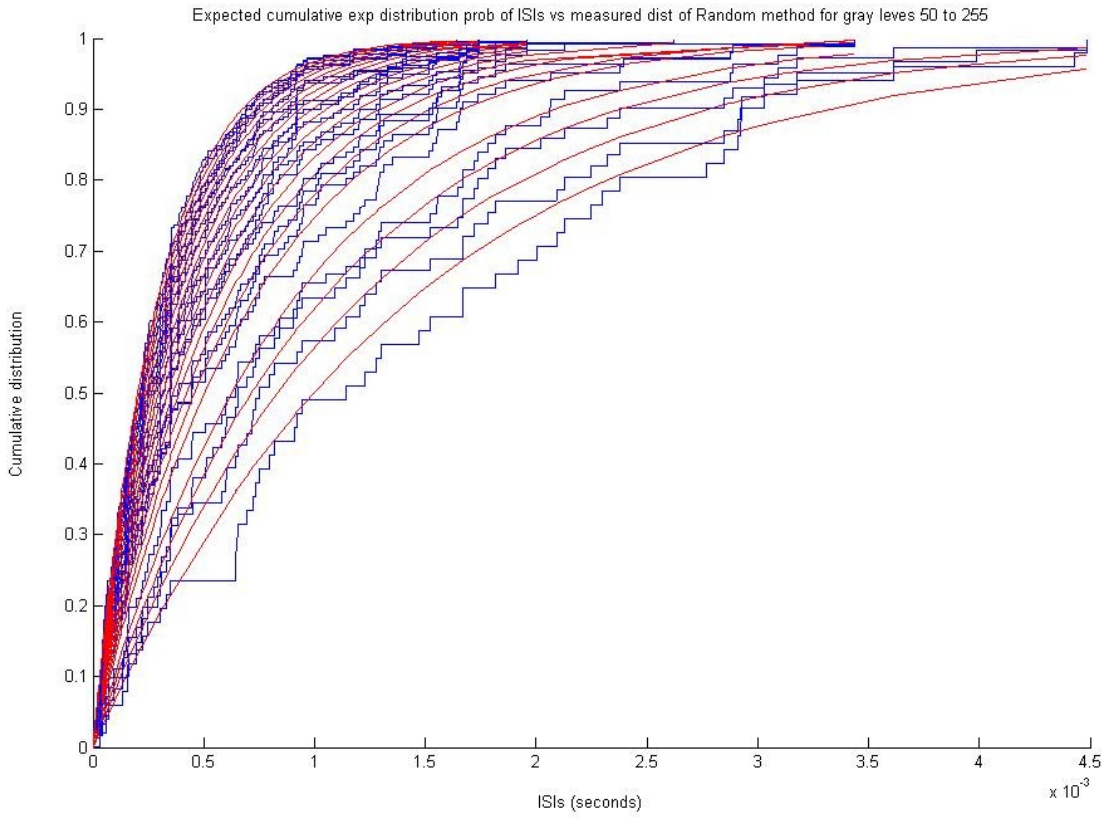

Ilustración 3.10: Distribución acumulada de ISI exponencial esperada vs medida

A continuación se utiliza el test de Kolmogorov-Smirnov (KS) para cuantificar en que medida la distribución de los intervalos entre eventos siguen la distribución exponencial descrita en la ecuación 3.18. En la ilustración 3.11 se muestra el resultado de aplicar el test de Kolmogorov-Smirnov a la distribución random obtenida para diferentes niveles de gris. El test se pasa si el resultado se encuentra por debajo del 5%. Puede verse que para pequeños niveles de gris, la distribución se separa de la distribución de Poisson. Todos los posibles números generados por el LFSR se utilizan para producir un único frame, y se vuelve a repetir. Esto causa que para niveles de gris pequeños, se tienen pocas distancias entre eventos diferentes. Se podría incrementar el periodo del LFSR incrementando el número de bits del mismo.

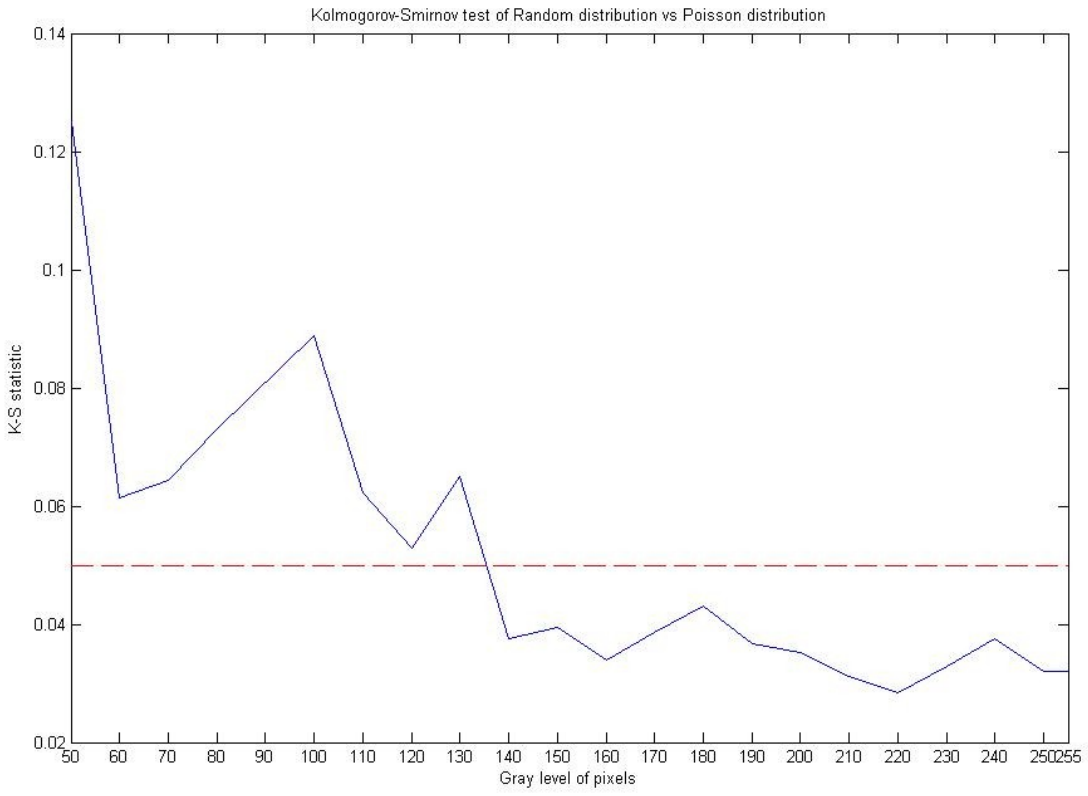

Ilustración 3.11: Test de Komogorov-Smirnov

En la ilustración 3.12 el resultado de utilizar dicho generador sobre la imagen de prueba. El ruido añadido se debe al factor aleatorio del LFSR. En el mapper probabilístico que se verá más adelante también se utiliza un registro LFSR para calcular la probabilidad y se observa un efecto similar introducido por el LFSR.

Ilustración 3.12: Imagen generada método random

3.5 Retina sintética: Captura desde una fuente de vídeo compuesto

La retina sintética intenta emular el comportamiento de algunas retinas AER ASIC existentes en la actualidad desarrolladas por diferentes grupos de investigación que se dedican a la computación neuroinspirada [LICHTSTEINER01][CULURCIELLO01]. Se ha desarrollado con el fin de poder probar diferentes mecanismos de generación de imágenes AER a bajo coste utilizando una FPGA, además de para servir como generador de estimulos AER para aquellos investigadores que se dedican al desarrollo de filtros u otros elementos dentro de una cadena de procesamiento AER y no disponen de una retina real para la depuración y desarrollo de dichos dispositivos. Para ello se utiliza una cámara de video o cualquier otra fuente de video compuesto que es digitalizada y a partir de dicha imagen generar una cadena de eventos AER utilizando algún algoritmo de conversión de frames a AER descrito en el apartado anterior. Para digitalizar la señal de video compuesto se ha conectado la salida de video compuesto producido por una cámara a un ADC de 100Mhz conectado a la entrada de la FPGA. Una máquina de estados se encargará de detectar los sincronismos horizontales y verticales (sincronismo de frame), para ir tomando las muestras en los instantes adecuados.

Describiendo a grandes rasgos las características de una señal de video compuesto, consiste en transmitir una imagen como niveles de tensión. La imagen se escanea por filas, de manera que el nivel de tensión es proporcional al nivel de luminosidad. Una vez terminada una fila, se envía un pulso de sincronismo, indicado por un nivel de tensión negativo, y se procede con la siguiente fila. Al terminar toda la imagen compuesta por 625 líneas de las cuales sólo 575 son visibles, se envía un pulso de sincronismo negativo de mayor duración que el que se empleaba al final de cada fila para indicar el final de un frame o imagen.

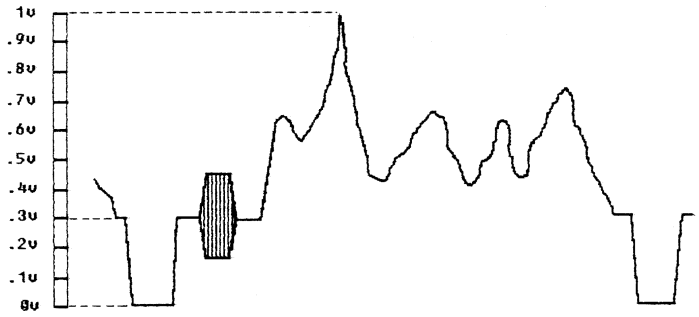

Ilustración 3.13: Señal video compuesto

Hay que comentar que para reducir la sensación de parpadeo causado por la frecuencia de actualización de cada imagen de 25 veces por segundo, la imagen se envía en dos mitades, primero las líneas impares y a continuación las pares de forma que cada semiframe se envía 50 veces por segundo. A esta característica se le llama entrelazado.

La información de color se encuentra modulada en cuadratura sobre la señal de luminosidad en una portadora que según la norma PAL es de 4.43 MHz. De esta forma un dispositivo que no decodificase la información de esta subportadora, lo que obtiene es únicamente la señal de luminosidad. Esto se hizo así para mantener la compatibilidad de la señal de video en color con los televisores antiguos monocromáticos. Como las imágenes AER actuales no tienen información de color, no será necesario decodificarla, por lo que nos centramos en la señal de luminosidad, sin portadora.

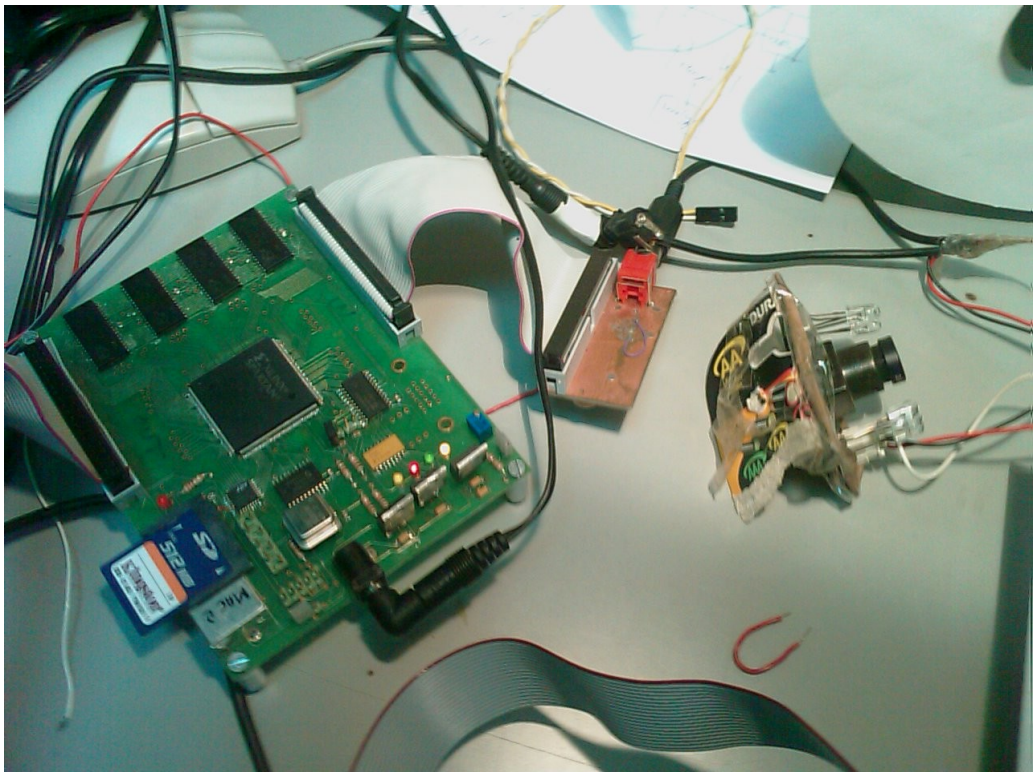

Ilustración 3.14: Placa USB-AER con ADC y camara

3.5.1 Retina sintética básica (transcodificación)

Como primera aproximación a una retina sintética, se propone un mecanismo de transcodificación que conforme va recibiendo la información de la imagen desde la señal de vídeo compuesto utilizada, va generando los eventos AER simultáneamente. Esto evita la necesidad de usar un framebuffer. Cuando se captura un determinado píxel de la imagen de entrada, se generarían todos los eventos asociados a dicho píxel. Por lo tanto, todos los eventos asociados a una neurona emisora serían emitidos agrupados en el tiempo. Esto no resulta deseable en ciertos escenarios AER por los motivos descritos anteriormente y en otros trabajos [LINARES01]. Evidentemente este problema se debe a que pretendemos eliminar el almacenamiento del frame completo, por lo que todos los eventos relativos a un píxel tienen que ser emitidos antes de que se capture el siguiente píxel.

Dado que el generador AER tiene una resolución mucho menor que la imagen de entrada, se enviará únicamente una porción de la imagen de entrada original. Si nos quedamos con una ventana de 64x64 píxels dentro de la imagen PAL de 625 líneas, se estaría transmitiendo una porción muy deducida de la imagen, con muy poco campo de visión, por lo que se opta por espaciar lo más posible los píxels que se muestrean para cubrir la mayor área posible de la imagen de entrada. Para esto, en lugar de muestrear en todas las filas, muestreará únicamente en una de cada 8 filas, y dentro de cada fila, se distribuirán homogéneamente las muestras dentro del intervalo de la fila, teniendo en cuenta que la imagen de entrada tiene una relación de aspecto de 4:3 mientras que la de salida es cuadrada (64x64 píxels).

Para capturar la señal de vídeo compuesto se ha implementado en VHDL un módulo que se encargará de detectar los sincronismos, y capturar una porción de la imagen total. Medirá la longitud de los pulsos de sincronismo, para diferenciar los sincronismos horizontales de los verticales. El sincronismo vertical (de frame) pone el contador de scanline o fila a cero. Este contador se incrementa en cada pulso de sincronismo horizontal. El valor de este contador es utilizado por la máquina de estados para determinar en qué filas debe realizar la captura. La captura de cada píxel dentro de una fila se realiza utilizando un temporizador dividiendo la señal de reloj general de la FPGA (50Mhz).El inicio de la temporización se inicia con el pulso de sincronismo. El diagrama de la máquina de estados utilizada sería el siguiente:

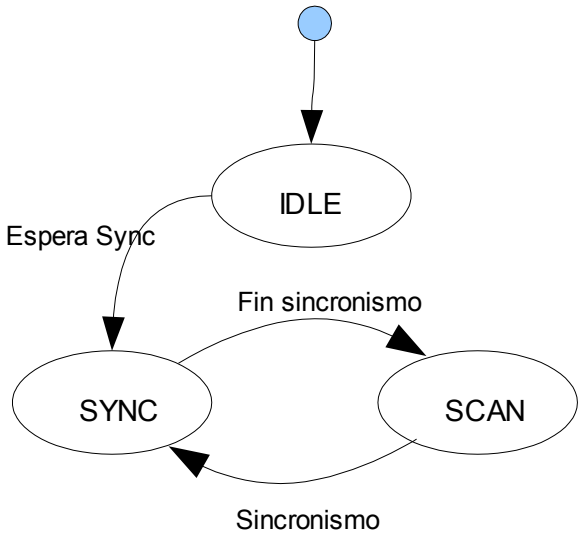

Ilustración 3.15: Diagrama estados capturadora

El estado IDLE se utiliza como estado inicial, en el que se sitúa la máquina de estados tras el reset, en espera de un sincronismo. Durante el proceso de captura, la máquina de estados alternará entre dos estados, SYNC y SCAN. En el estado SYNC se encuentra cuando el dato proveniente del ADC se encuentra por debajo del umbral que determina el nivel de tensión de la señal de sincronismo. Mientras se está en este estado se está contando el tiempo que se permanece en él. Una vez termina el sincronismo se pasa al estado SCAN. Antes del cambio de estado, en función del tiempo que se permaneció en SYNC, en el caso de ser un sincronismo horizontal, incrementará el contador de línea y pondrá a cero el contador de columna. Si se trataba de un sincronismo vertical pondrá a cero ambos contadores. El contador de fila indica qué fila se está recibiendo. Se incrementa en uno cada vez que se detecta un sincronismo horizontal, y se pondrá a cero al comienzo de cada frame (sincronismo vertical). El contador de columna indica qué columna dentro de cada fila se está recibiendo y se incrementa dentro del estado SCAN. Para incrementarlo se utiliza un generador de reloj de píxel obtenido a partir de la señal de reloj general.

El siguiente fragmento de código VHDL muestra el funcionamiento de la máquina de estados. En el estado SYNC, se determina si se trata de un sincronismo horizontal o vertical en función de la duracción del pulso de sincronismo. Si el sincronismo dura más de 800*20ns, 16us, se trata de un sincronismo vertical. El contador de fila se inicia a cero. Si el ancho de pulso el inferior, se trata de un sincronismo horizontal, y se incrementa el contador de fila. Las primeras 20 líneas son descartadas, al no contener información de video. De las restantes líneas de video, se captura una de cada cuatro.

```
when SYNC =>
	if (vidcap_data <5) then
		CS <= SYNC;
		vidcap_counter <= vidcap_counter + 1;
	elsif(vidcap_counter <800) then  -- Sincronismo horizontal
		CS <= SCAN;
		vidcap_vert <= vidcap_vert + 1;
		if (vidcap_vert > 20 and vidcap_vert(1 downto 0) ="00") then
			vidcap_y <= vidcap_y + 1;
		end if;
		vidcap_horz <= (others =>'0');
		vidcap_x <= (others => '0');
		vidcap_counter <= (others =>'0');
	else                -- Sincronismo vertical
		CS <= SCAN;
		vidcap_vert <= (others =>'0');
		vidcap_horz <= (others =>'0');
		vidcap_x <= (others =>'0');
		vidcap_y <= (others =>'0');
		vidcap_counter <= (others =>'0');
	end if;
```

En el estado SCAN se está cuando se recibe una línea (no se está en un sincronismo). En función del valor de los contadores de fila y columna, se decide si se debe capturar el nivel de gris para guardarlo en la memoria de la imagen. Se utilizan dos contadores auxiliares para llevar el índice del píxel dentro de la imagen AER, que se incrementan cada vez que se captura un evento.

```
when SCAN =>
	vidcap_counter <= vidcap_counter + 1;
	if (vidcap_counter >500 and vidcap_counter(4 downto 0) = "00000" and
	    vidcap_vert > 20 and vidcap_vert(1 downto 0) ="00") then
		vidcap_x <= vidcap_x + 1;
		if(vidcap_x < 64 and vidcap_y < 64) then
			vdata <= '1';
		end if;
end if;
	if (vidcap_data < 5) then
		CS <= SYNC;
		vidcap_counter <=(others =>'0');
	end if;
```

Si nos centramos en la señal de video compuesto, tenemos 50 semicuadros por segundo, con 315 líneas cada una, lo que da una frecuencia de línea de unos 15khz. Si tomamos 64 muestras dentro de una línea, tendremos aproximadamente un millón de capturas por segundo. Para describir un píxel, cuya luminosidad puede variar entre 0 y 255 habría que emitir en el peor caso 255 eventos entre píxel y píxel. Esto no es posible con la frecuencia de reloj actual de las placas AER (50-100Mhz), teniendo en cuenta además que la sobrecarga del protocolo de handshake impone varios ciclos de reloj para cada evento transmitido. Si el throughput de eventos no es lo bastante alto como para emitir todos los eventos de un píxel antes de pasar a capturar el siguiente, no le daría tiempo al generador AER a emitir todos los eventos necesarios. Por lo tanto, o bien se pierden eventos, con el que el nivel de gris enviado sería inferior al nivel real del píxel (se introduce un error en la reconstrucción), o se ignora la siguiente captura, dejando un píxel en negro, al no haberse podido capturar.

Para solucionar este problema podemos reducir el número de niveles de grises disponibles para cada píxel. Si reducimos de 256 niveles de grises a 16 o 32, se reduce sustancialmente el ancho de banda necesario para transmitir cada píxel. Para el caso de 16 niveles de grises, tenemos que el número de eventos se reduciría al 6.25% del tráfico original, mientras que para 32 niveles de gris se reduce al 12.5%. Por tanto estamos hablando de una reducción de tráfico de alrededor de un 90%, a costa, eso si, de sacrificar la cantidad de niveles de gris disponibles.

Por otra parte, no se toman capturas de la imagen en todas las líneas. Al sólo necesitar 64 líneas de las 315 disponibles, se espacian las capturas, ignorando 3 de cada 4 líneas de video. Tenemos entonces que hay líneas en las cuales hay que tomar 64 muestras relativamente cercanas en el tiempo, que obligarían a un flujo de datos rápido para funcionar correctamente (1 de cada 4 líneas se captura), mientras que en el resto del tiempo (75%), no se está capturando nada.

Añadir una FIFO a la salida lo bastante grande puede solucionar este error. En las líneas de video donde hay que capturar, se emitirían muchos eventos, que quedarían en la FIFO almacenados para irse transmitiendo progresivamente al receptor en función de su velocidad. Si la FIFO es lo bastante grande como para almacenar los eventos correspondientes a toda una línea, después tendrá el tiempo correspondiente a las 3 líneas siguientes para irse vaciando.

De esta forma, tenemos una frecuencia de captura “virtual” cuatro veces menor, del orden de 250.000 capturas por segundo, o una captura cada 4us, pudiéndose alcanzar una solución de compromiso que permita transmitir con fluidez video AER.

De todas formas, este mecanismo sigue presentando el problema de que los eventos de cada neurona no están homogéneamente distribuidos en el tiempo, sino que se agrupan por valores. Primero se emiten todos los eventos del primer píxel, después todos los del segundo, y así sucesivamente. Al no almacenar ninguna imagen en RAM, este mecanismo no sirve para retinas derivativas en el tiempo. La principal ventaja de este mecanismo sin recurrir a las FIFOS reside en su gran sencillez. Si se coloca las FIFOS se pueden obtener otras mejoras, como por ejemplo implementar un mecanismo por el que los eventos de una línea puedan redistribuirse de forma aleatoria antes de salir al AER sin tener que estar todos agrupados, esto es equivalente a guardar parte de la imagen en formato eventos y redistribuirlos de forma oportuna para su transmisión por el bus AER. Es decir, en lugar de guardar un frame en formato digital y a partir de él obtener la secuencia de eventos definitiva, se guarda una secuencia de eventos que representa la imagen y se reordena para transmisión. Pero uno y otro métodos son equivalentes pues se parte de una imagen de vídeo compuesto ya dividida en frames. Otra mejora es que podría capturarse interpolando entre varias líneas, de forma que en vez de capturar 64 píxels en una línea de cada 4, capturar 16 píxels en cada línea. Según la forma de realizar esta interpolación la imagen resultante estará más o menos distorsionada respecto a la original.

Como siguiente evolución a la retina sintética, a continuación vemos la retina sintética proporcional, que como veremos en el apartado siguiente, incorpora un framebuffer, implementado como una memoria de doble puerto y el generador exhaustivo descrito en el apartado anterior.

3.5.2 Retina sintética proporcional

Como adelantábamos en el apartado anterior, un mecanismo para construir una retina sintética que distribuya los eventos de manera homogénea puede obtenerse a partir del generador exhaustivo para imágenes estáticas descrito anteriormente en este mismo capítulo, el cual habrá que modificar para aumentar su funcionalidad de imágenes estáticas a vídeo en movimiento.

Para ello, la memoria RAM interna de la FPGA, que se utilizaba para almacenar la imagen que debía ser convertida a AER, es sustituida por una memoria de doble puerto (que permite dos accesos simultáneos e independientes por medio de dos buses de datos y dirección diferentes para cada uno de los puertos). De esta forma independizamos el proceso de captura de video del proceso de generación AER. Cada uno de estos procesos tiene una máquina de estados independiente que funcionan concurrentemente de forma que el generador AER continuamente va escaneando la imagen a transmitir y generando eventos cuando corresponda tal y como se describía al tratar el método exhaustivo "bitwise". Por otro lado, el capturador de vídeo se encarga de mantener esta imagen en memoria actualizada con cada nuevo frame que se recibe desde la fuente de video utilizada.

Si el ancho de banda del bus AER no fuese suficiente para transmitir todos los eventos necesarios, afectaría al número de frames por segundo que se están recibiendo. Sin embargo, los frames intermedios que se podría suponer que son descartados, no se descartan, sino que actualizan la memoria de video, de forma que el flujo AER de salida se adecúa continuamente a los nuevos valores almacenados. La imagen reconstruida, a menos que reduzcamos el tiempo de integración del receptor AER, sería el resultado de interpolar varias imágenes. Si se reduce el tiempo de integración, tenemos que sólo podemos recibir un subconjunto de los eventos que inicialmente describirían la imagen AER, por lo que la imagen se oscurecería, al haber menos eventos, pero no quedaría incompleta como en el caso anterior, dado que los eventos están bien distribuidos en el tiempo, y por tanto la información se transmite también. Solo se perdería redundancia en los eventos, empeorándose el SNR de la imagen en general. Bastaría con aumentar el contraste en la reconstrucción (aumentar la ganancia de cada píxel) para paliar el oscurecimiento de la imagen.

En la ilustración 3.16 puede verse el diagrama de bloques de la retina proporcional. Se observa que respecto al generador exhaustivo estático (ilustración 3.4), se ha sustituido la RAM por una RAM de doble puerto, como acabamos de comentar, y una nueva entidad, descrita en el

apartado anterior, en la retina básica, que se encarga de la separación de sincronismos y de mantener actualizada la memoria de la imagen a convertir a AER (proceso de captura de video compuesto).

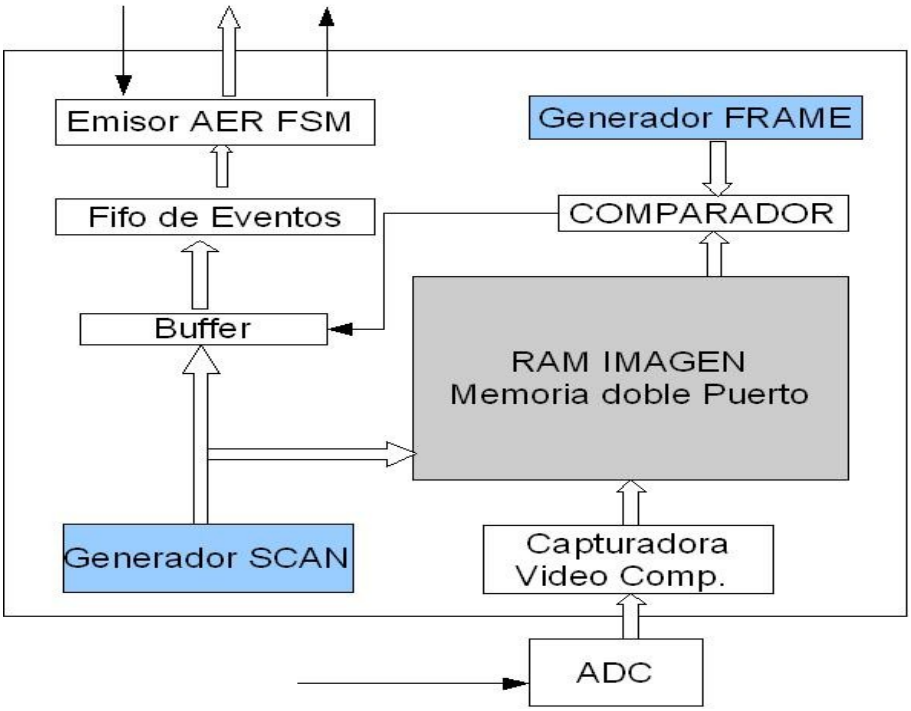

Ilustración 3.16: Retina básica

La señal de video compuesto se conecta a la entrada del ADC, que ofrece el valor en digital a la capturadora de dentro la FPGA. A partir de dicha señal capturada se separa la información de sincronismo que controlan la máquina de estados de la capturadora.

3.5.3 Retina sintética derivativa

A partir del diseño anterior se puede obtener una retina derivativa, lo que implica una significativa reducción de los eventos a transmitir. La salida de este tipo de retinas no es en función del nivel de luminosidad recibido por cada píxel, sino que es función de la variación de la luminosidad en dicho píxel. Esto es lo que se conoce como una retina derivativa en el tiempo, puesto que la salida es función de la derivada de la luminosidad en el tiempo (variación de luminosidad entre dos frames consecutivos). Esto además nos va a permitir comparar la salida generada por una retina sintética basada en un sensor de vídeo compuesto convencional y un generador sintético basado en FPGA, con la salida generada por una retina AER bio-inspirada realizada en ASIC, como la desarrollada en el INI de Zurich [TOBI01][LICHTSTEINER01].

Para que la salida sea función de la variación de luminosidad, el generador debe comparar la luminosidad entre las dos últimos frames, y generar eventos correspondiente a la diferencia entre ambas, para lo que debe substraerse de la imagen que se está capturando el valor de dicho píxel en el frame anterior. Para esto es necesario almacenar en memoria, no un único frame como ocurría en la retina en luminosidad, sino los dos últimos frames, o una combinación lineal de ambos. En nuestro caso almacenamos la última imagen capturada y la diferencia entre esa imagen almacenada y la anterior. Como se verá un poco más adelante en el diagrama de estados del capturador, al capturar un nuevo píxel, se lee de la memoria el valor previo de la posición donde se almacenará el nuevo valor, pero antes de actualizarla se calcula la diferencia entre ambas y se almacena en otra tabla en memoria, en la “imagen en diferencia”.

La FPGA Spartan II utilizada en el adaptador USB-AER tiene alrededor de 7KB de RAM interna, que en el generador anterior se utilizaban 4KB como memoria de doble puerto donde se almacenaba el frame capturado (64x64x8bits), y los 3KB restantes como FIFO para los eventos AER de salida. Para el caso de la retina derivativa propuesto en este apartado, no hay memoria interna suficiente para almacenar ambos frames, incluso eliminando las FIFOs de salida. Para almacenar ambos frames se utiliza entonces una parte de los 2MB de memoria externa de la placa USB-AER. La principal dificultad al sacar la memoria RAM fuera de la FPGA es implementar la memoria de doble puerto, que permita ser accedida independientemente por el generador de eventos, y por el capturador de video.

Los 2MB de memoria están estructurados como 512Kx32. De estos 32 bits de datos repartidos en 4 bancos, únicamente se utilizan 16, dejándose sin uso la mitad de los bancos de memoria. Uno de los bancos se utiliza para almacenar la imagen en diferencia, y en el otro la imagen de luminosidad. Cada banco tiene señal de lectura/escritura independiente, por lo que puede leerse de un banco mientras se está escribiendo en otro (comparten las líneas de dirección). Esto permite realizar la actualización de un píxel en únicamente dos ciclos de reloj. En el primer ciclo de reloj se lee el valor de la luminosidad de un píxel de un primer banco, y se escribe en el segundo banco, imagen de diferencias, el resultado de restar este valor recuperado del nuevo valor de entrada. En un segundo ciclo de reloj se escribe el valor de entrada en el primer banco, quedando así en dos ciclos actualizados ambos valores, tal como se muestra en el siguiente diagrama de estados:

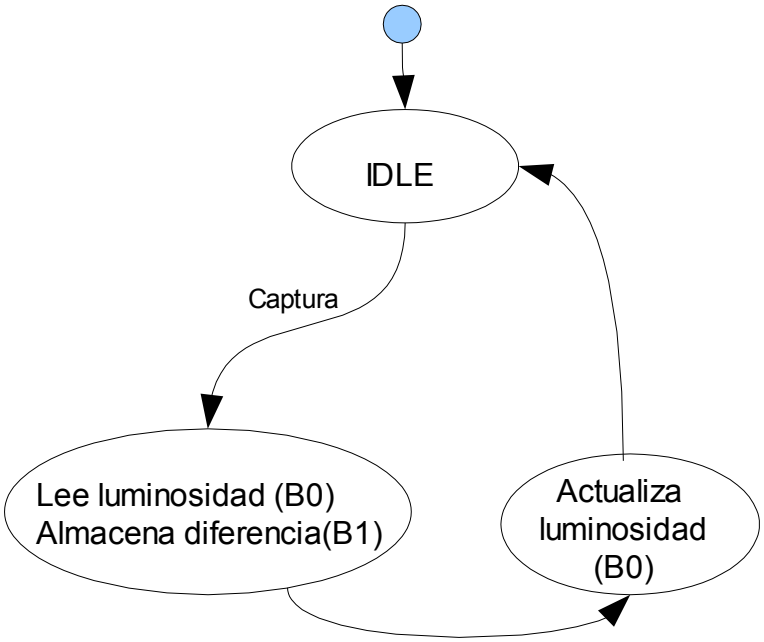

Ilustración 3.17: Diagrama estados retina derivativa

Con B0 se hace mención al banco 0 de memoria, donde se almacena la información de luminosidad del último frame, y B1, o banco 1, donde se almacena la diferencia entre el último valor del píxel (B0), y el nuevo valor que se esta capturando.

Al emplear la memoria externa, el tamaño de la memoria deja de ser una limitación, por lo que puede ampliarse a imágenes de 128x128 o incluso de 256x256, siendo este el tamaño máximo permitido por el bus AER utilizado en [CAVIAR01]. Sin embargo, la cantidad de eventos necesaria para transmitir una imagen de 256x256 requeriría de una frecuencia de reloj superior a 100Mhz para poder mantener una tasa de actualización de al menos 20fps, por debajo de la cual la percepción de

la secuencia de video deja de ser fluida. Además nos quedaríamos sin ninguna línea libre para la transmisión del bit de signo. Por esto el tamaño de la imagen se limita ahora a 128x128, utilizándose 14 bits de los 16 del bus AER para transmitir la dirección del píxel. Una línea adicional enviaría la información de signo, quedando aún una línea libre en el bus.

Con una frecuencia de reloj interna de 100Mhz, obtenida duplicando la señal de reloj de 50Mhz, según el esquema descrito anteriormente se escanea a razón de cien millones de píxels por segundo, mientras que para la captura de video se realizan 128x128x50=819200 accesos. En el diseño utilizado, la generación de eventos está normalmente accediendo a memoria, salvo cuando es interrumpida por el capturador. Mientras que el capturador de video esta actualizando la información de la imagen, el generador se encuentra detenido. El flujo de eventos, sin embargo, no tiene por que interrumpirse, mientras queden eventos por salir en las FIFOs de salida, minimizándose los efectos negativos sobre la generación provocados por los ciclos de reloj en los que la memoria no es accesible.

A diferencia del generador en luminosidad del apartado anterior, donde todos los valores de luminosidad eran positivos, dando lugar únicamente a eventos positivos, al substraer dos imágenes, el resultado de dicha sustracción puede ser positivo o negativo, en función de si la luminosidad de dicho píxel ha evolucionado de menos luminosidad a más o viceversa. Por tanto hay que modificar la generación de eventos para que traten de manera adecuada el bit de signo. Los valores negativos al quedar expresados en complemento a dos generarían muchos eventos positivos, correspondientes a 2^n-x. En lugar de esto, hay que generar tantos eventos como el valor absoluto de la resta, pero con el bit de signo activado (eventos negativos).

En la siguiente figura se observa el diagrama de bloques de la retina derivativa sintética. Pueden observarse las diferencias con la retina propuesta en el apartado anterior. Se mantiene la memoria interna para la FIFO de eventos, pero la memoria de doble puerto se ha sustituido por un arbitrador que gestiona el acceso a la memoria externa, fuera del bloque de la FPGA.

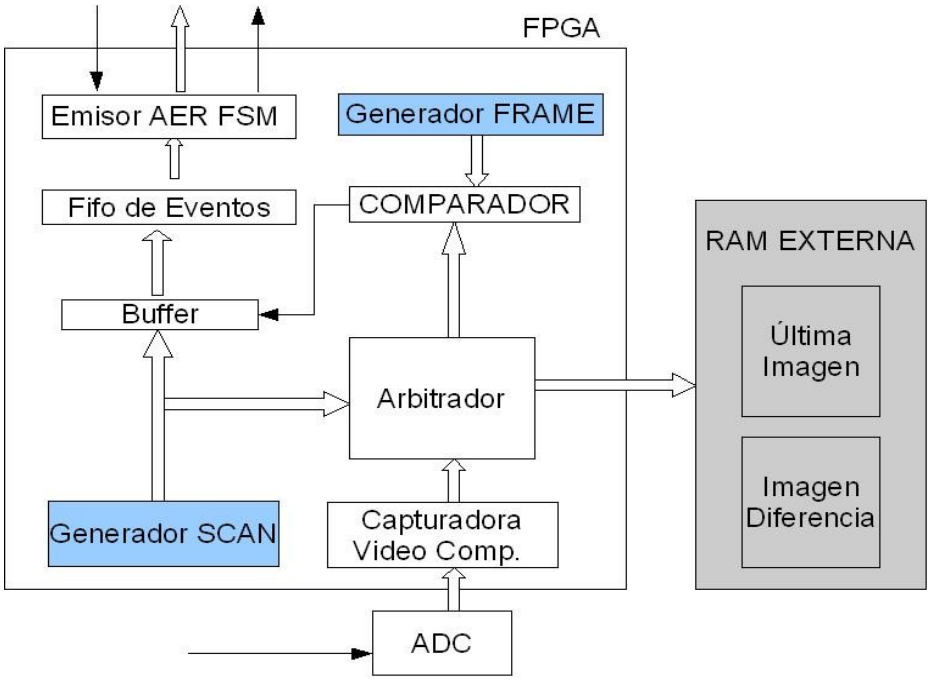

Ilustración 3.18: Retina derivativa

Este diseño es válido tanto para una retina derivativa como para una retina proporcional, pero para imágenes mayores de las conseguidas en el primer diseño propuesto. La limitación del tamaño de la imagen viene impuesto ahora por el número de líneas de datos en el bus AER, y el número de eventos por segundo, aunque este segundo factor puede solucionarse a costa de reducir el número de imágenes por segundo a transmitir, o lo que es lo mismo, aumentando la latencia del sistema al bajar la constante de tiempo del mismo.

Para cambiar entre retina derivativa o retina proporcional basta con cambiar a cual de los dos bancos de memoria accederá el generador de eventos.

3.5.4 Comparación retina derivativa sintética con una ASIC

En este apartado se va a comparar la salida obtenida con una retina AER [TOBI01] con la salida obtenida por la retina sintética derivativa a partir de una cámara de video compuesto. Para realizar esta comparación se plantea un escenario AER basado en un objeto en movimiento, que será observado por una retina y por una cámara de video. La salida de la cámara de video se utiliza como fuente de video para el generador sintético. El uso del generador sintético se debe a que la retina AER utilizada es una retina derivativa, por lo que para posibilitar la comparación se utilizan dos fuentes AER que se esperan con un comportamiento similar. La salida del generador sintético se combina mediante el uso de un Switch-AER [RIVAS01][RIVAS02] que combina el tráfico de salida de ambos dispositivos con el fin de obtener un único flujo AER y recibir los eventos de ambas fuentes con un único dispositivo datalogger [RIVAS01][RIVAS02], y así mantener la coherencia en la medición de los tiempos de los eventos, pudiendo relacionarse el instante en que se han generado los eventos en ambas retinas.

Ilustración 3.19: Montaje empleado para las pruebas

El data-logger es un diseño en VHDL para la tarjeta USB-AER que permite capturar eventos AER, junto a una información de tiempo (timestamp), para ser procesada offline en un ordenador PC posteriormente. La información de los eventos que se capturan se almacena en la memoria RAM de la placa USB-AER. Cada dato capturado ocupa 32 bits, de los cuales 16 de ellos se destinan para almacenar la información temporal del instante en que se capturo el evento, y los 16 bits restantes, 8 de ellos para la dirección de fila y otros 8 para la dirección de columna. Se dispone de 512kx32bits, por lo que podrá almacenar un total de 512keventos. El tiempo de captura dependerá por lo tanto de la cantidad de tráfico en el bus.

Tanto el "datalogger" como el "framegrabber" tienen dos modos de funcionamiento diferentes: en el modo normal, el dispositivo actúa como un elemento final de una cadena AER, gestionando convenientemente el protocolo de handshake del bus AER, y activando la señal ACK en respuesta a REQ. En un segundo modo, el modo "pass-throught" el dispositivo se intercala en una cadena AER, actuando de forma transparente, para los demás elementos de la cadena. En este modo, se monitoriza la señal REQ del bus, para realizar la captura sincronizada de los datos de los eventos. Este segundo modo resulta muy útil para la depuración de una cadena AER, al permitir conocer la información que se va transmitiendo entre las diferentes capas de una cadena.

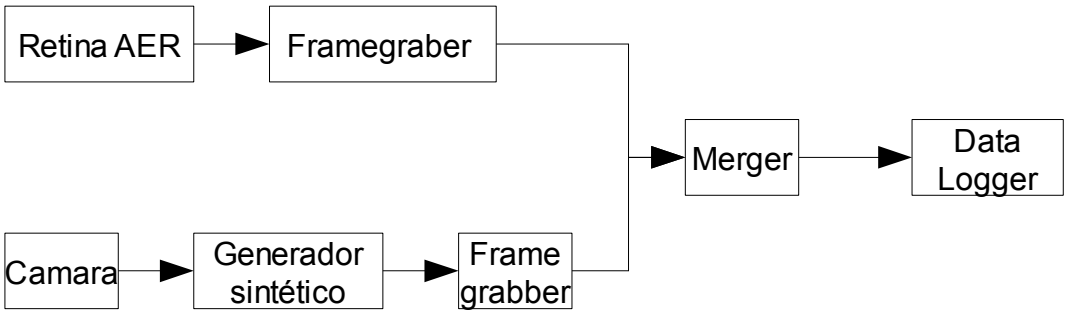

Ilustración 3.20: Escenario Retina Silicio vs Retina.Sintética

Con el fin de poder observar el correcto funcionamiento de ambas retinas, antes del dispositivo Switch, se intercala un framegrabber configurado en modo pass-throught, para poder monitorizar la salida de las retinas sin interferir en el tráfico AER.

Hay que remarcar que debido a las diferencias de concepción de una retina AER y una cámara de video compuesto, la cámara de video compuesto tiene ciertas limitaciones respecto a la

velocidad de los objetos en movimientos que pueden ser capturados. Al dividir el tiempo en frames, ningún movimiento más veloz que el tiempo de frame sería captado por la cámara de video, mientras que en la retina AER derivativa, al no fijarse ningún tiempo de frame, no existiría esta limitación, quedando limitado únicamente por el número de eventos que se generan, y en último término, por la sensibilidad del sensor. Como hemos comentado al comienzo de este capítulo, la frecuencia de eventos, o la separación entre eventos es el factor que limita la capacidad de captar movimientos rápidos.

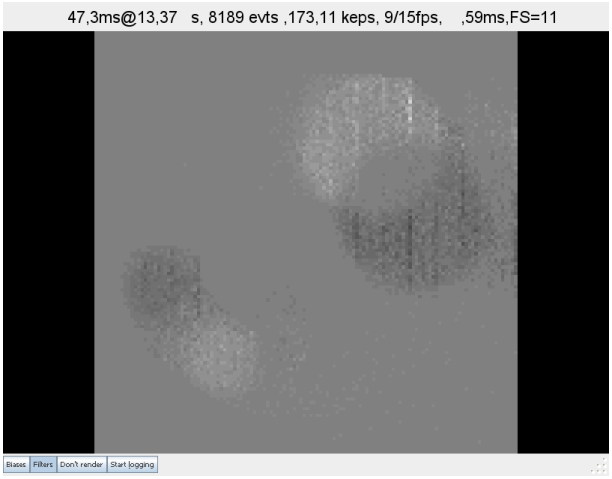

Ilustración 3.21: Captura retina silicio

Ilustración 3.22: Captura retina sintética

En las ilustraciones 3.21 y 3.22 puede verse la captura por separado de la retina original, y de la retina sintética. Hay que remarcar que resulta difícil comparar cuantitativamente ambas retinas debido a que entran en juego muchos parámetros diferentes, que además no pueden en su mayoría ajustarse o modificarse. Desde el punto de vista de la óptica, la superficie y forma del sensor, su distancia a la lente de enfoque, y las características de la lente determinan el campo de visión de la cámara, y el tamaño de los objetos observados, así como su posición dentro de la escena capturada. Aún ubicando ambas retinas en la misma posición, el tamaño de los objetos y su posición difiere entre ambas, como se observan en las ilustraciones anteriores. Que los objetos aparezcan en posiciones relativas distintas repercute en la dirección de los eventos que aparecerán en el bus. La diferencia de tamaño afecta por otra parte a la cantidad de píxels afectados.

La resolución de los sensores también sería un parámetro a tener en cuenta, aunque en este caso, la resolución de las cámaras de video actuales es superior al de las retinas AER, puede escalarse escogiendo un área o un subconjunto de píxels de la imagen original. La disposición de dichos píxels afectará también al campo de visión de la cámara.

Desde el punto de vista del sensor, la sensibilidad del sensor influye en la respuesta de dicho sensor frente a la luz. A igual iluminación, un sensor de menor sensibilidad presenta una respuesta menor. Las cámaras de video suelen tener un control automático de ganancia y también modifican la velocidad de obturación para adaptarse a las condiciones ambientales. Sin embargo, esto no suple la calidad del sensor, por lo que un sensor menos sensible se traducirá en una imagen menos contrastada, y al aumentar la ganancia de la señal, se aumenta tambíen el ruido de la misma. Por otra parte, al incrementar el tiempo de obturación para aumentar el periodo de integración y por tanto la cantidad de luz recogida, provoca que la respuesta de la cámara al movimiento empeore, apareciendo imágenes movidas.

Desde el punto de vista de la generación de eventos, las diferencias en el diseño interno de la reina y del algoritmo utilizado para generar dichos eventos, determina la distribución de los mismos, y la relación entre el nivel de cambio de luminosidad y su número (función de transferencia). Aún aunque ambas retinas tuvieran una función de transferencia lineal (como veíamos en las ecuaciones 3.4 y 3.6), la constante de proporcionalidad (pendiente de la recta) y el desplazamiento o offset pueden no coincidir, por lo que ante el mismo estímulo, el número de eventos diferirá.

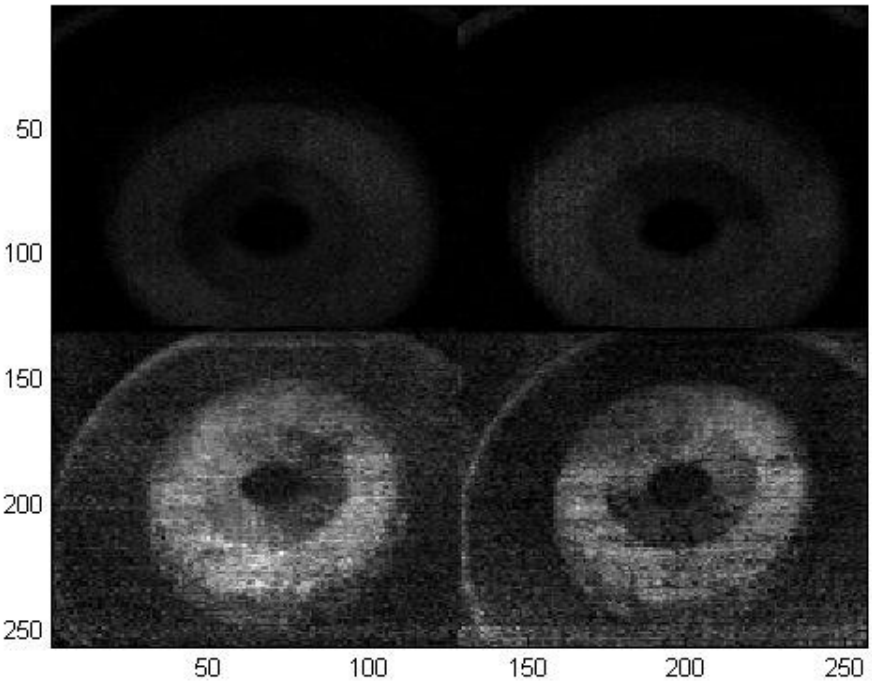

Ilustración 3.23: Imagen AER reconstruida por el datalogger

En la ilustración 3.23 se observa la imagen reconstruida en el datalogger. La mitad superior de la imagen se corresponde con lo que captura la retina real, y la mitad inferior lo capturado con la retina sintética. Para cada una de estas dos mitades, la mitad izquierda representan los eventos positivo y la mitad derecha los eventos negativos. Puede observarse que en ambos casos las reconstrucciones separadas de los eventos positivos y negativos resulta prácticamente equivalente. La imagen de la retina sintética aparece con mayor intensidad porque la retina sintética genera más eventos que la retina real. En la tabla 3.4 se resumen algunos datos de la captura realizada:

	Retina real	**Retina sintética**
Eventos positivos	44032	233789
Eventos negativos	51115	195351
Eventos totales	95147	429140
% eventos positivos	46.3%	54.5%
% trafico (total eventos)	19%	81%
Luminosidad máxima	17	63

Tabla 3.4: Resultados captura datalogger

En las ilustraciones siguiente vemos el resultado de simplificar los eventos positivos y negativos en la retina real (ilustración Error: No se encuentra la fuente de referencia) y la retina sintética (ilustración Error: No se encuentra la fuente de referencia). Como la retina sintética genera aproximadamente una quinta parte de los eventos que genera la retina sintética, la imagen de la primera se ha realzado para compensar la menor densidad de eventos.

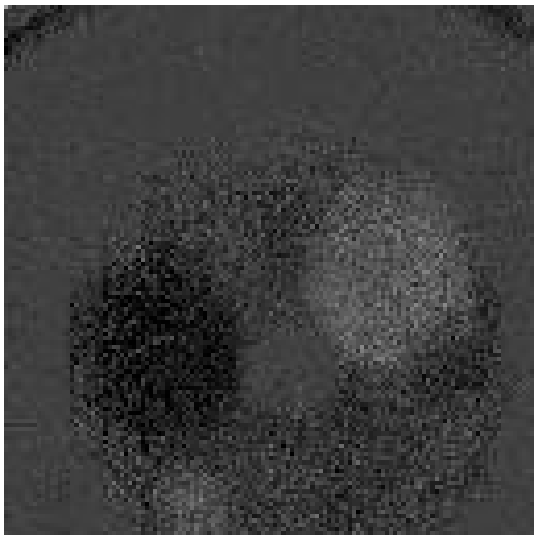

Ilustración 3.24: Retina real simplificada y ajustada

Ilustración 3.25: Retina sintética simplificada y ajustada

4. Transformaciones sobre imágenes AER

En la actualidad no se conoce exactamente como es el procesado visual que realizan los sistemas biológicos, existen diversos estudios sobre este tema [SHEPHERD01][THORPE01] [THORPE02][GROSSBERG01][DELORME02][FABRE02][VANRULLEN01][BOAHEN01] [HIGGINS01][OZALEVLI01][INDIVERI01][THORPE03]. Sin embargo, es posible llevar a cabo filtros sobre flujos de información visual sin necesidad de operar exactamente como lo haría la naturaleza.

En este capítulo vamos a deducir una serie de transformaciones en la imagen partiendo de pequeñas modificaciones en el flujo de información AER, estos filtros se obtienen realmente por aproximación y por el análisis de cómo se lleva la información en el bus AER (dirección y tiempo). Un aspecto interesante, que aún está por estudiar en profundidad, es el que se refiere a las operaciones con pulsos y su equivalencia a lo que sería el procesado digital de señal, no se ha definido lo que sería un "álgebra de pulsos" con la que poder realizar las operaciones y funciones correspondientes. Por tanto, los filtros que se estudian en este capítulo nacen por la traslación de los procesos que se realizan con un computador sobre imágenes digitales a lo que se puede hacer sobre el flujo de imágenes representadas en AER.

En una cadena AER, entre un emisor y un receptor AER pueden insertarse dispositivos [MEROLLA01][VANRULLEN01][ROUSSELET01][INDIVERI01][HIGGINS01][OZALEVLI01] [BOAHEN02][KALAYJIAN01][ROUSSELET02][LAZZARO02][MORTARA01][HAFLIGER01] que reciban, procesen o transformen los eventos y los envíen al siguiente elemento AER, actuando como un filtro sobre la imagen de entrada[SERRANO02][SERRANO01]. Estos filtros pueden irse concatenando de manera que la salida de un filtro puede utilizarse como entrada del siguiente, pudiendo a partir de transformaciones sencillas ir componiendo transformaciones más complejas mediante el empleo combinado de muchos filtros.

Ilustración 4.1: Filtrado AER serie

Por medio de elementos Splitter y Merger [RIVAS01][RIVAS02] puede separarse un flujo de eventos AER en varios caminos, aplicando filtros separados a cada uno de ellos, para más tarde volverlos a combinar.

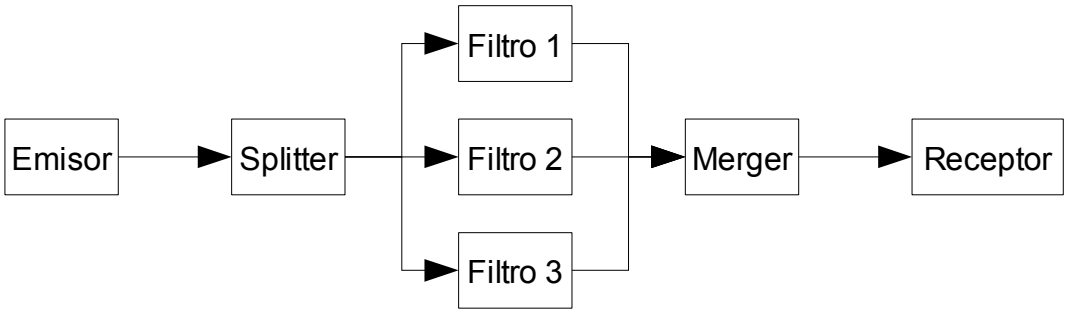

Ilustración 4.2: Filtrado AER paralelo

Existen en la actualidad muchos dispositivos para realizar procesamiento de señales AER ya sean de visión [GOMEZ01][MAHOWALD1], como pueden ser convoluciones[SERRANO03], reconocimiento de formas, winner-take-all [HAFLIGER01] o de audio [GOMEZ04]

Siguiendo la filosofía del bus AER, estos elementos de filtrado deben operar sobre los eventos de entrada, en lugar de sobre una imagen digital como sería el caso de utilizar un computador tradicional. Los filtros propuestos en este apartado, no reconstruyen la imagen AER transmitida como paso previo a la transformación, sino que operan directamente sobre los eventos, modificando su dirección y su frecuencia por medio de métodos probabilísticos.

Con el fin de realizar transformaciones sencillas en las imágenes AER que se transmiten, se inserta una placa USB-AER basada en FPGA descrita en el capítulo 11.1. Al estar basada en una FPGA, permite su reconfiguración para desempeñar distintas funciones y así experimentar con distintos diseños.

Para realizar transformaciones sobre imágenes, se ha utilizado un diseño como mapeador, en cuya versión básica consiste en realizar un mapeado 1:1 de los eventos de entrada, buscando en una tabla en memoria el evento de salida correspondiente al evento de entrada recibido. Estableciendo el contenido de estas tablas de manera adecuada pueden realizarse transformaciones básicas como desplazamiento de la imagen, rotación y escalado, aunque con ciertas limitaciones que se describen a lo largo de este capítulo. En una versión más compleja del mapeador, se contempla la posibilidad de que un único evento de entrada, dispare distintos eventos de salida con una determinada probabilidad asociada. Esto permitirá solventar algunas de estas limitaciones, y permitir otras transformaciones adicionales.

Dentro de las posibles transformaciones que estos sistemas mapeadores pueden realizar sobre imágenes AER cabría remarcar las convoluciones; lo cual representa un nuevo mecanismo y una aportación importante de esta tesis. Ya que hasta ahora estas funciones han sido realizadas por

los chips ASIC desarrollados con tal finalidad [SERRANO03], siendo la mayoría analógicos. Aunque probablemente cuando esta tesis salga a la luz ya podamos disponer de chips convolucionadores digitales (realizados bajo el proyecto Samanta2), aunque con una filosofía diferente a la mostrada en este trabajo. Dentro de las posibles transformaciones a realizar sobre imágenes AER, cabría remarcar la realización de convoluciones [RPAZ04].

4.1 Estructura mapeador básico

El mapeador básico [GOMEZ03] es un dispositivo AER que se encarga de sustituir cada evento de entrada por otro evento de salida [HIGGINS02][HIGGINS03]. Esta sustitución se realiza almacenando en unas tablas en memoria la equivalencia entre todos los posibles eventos de entrada con su evento de salida correspondiente. Hay que entender que el mapeador no modifica el número de eventos en el bus, ni su emplazamiento temporal, únicamente sustituye la dirección de un evento por otro.

Al modificar la dirección de un evento estamos sustituyendo una neurona que sería la que originariamente enviase los eventos, por otra. En el caso de imágenes esto se traduce en que podemos modificar la posición en la que se reciben cada uno de los píxels de una imagen. Si se quiere combinar por ejemplo las imágenes provenientes de varias fuentes para conseguir una única imagen de mayor tamaño, antes de combinar los impulsos provenientes de las diversas fuentes, es necesario modificar sus direcciones para evitar que distintas fuentes generen las mismas direcciones, y que las imágenes se compongan superponiéndose unas a otras.

El mapeador básico utiliza el evento de entrada como índice para buscar en una tabla en RAM el evento de salida correspondiente. El tamaño de la tabla en RAM debe tener tantas entradas como eventos de entrada existan. El número de bits del evento de salida no tiene por qué coincidir con el número de bits del evento de entrada. Volviendo al caso de las imágenes, cada retina individual, de tamaño $2^n x 2^m$, generaría eventos de n+m bits.

Retina origen	***Evento entrada***	***Evento salida***
1	(0,0)	(0,0)
1	(x-1,y-1)	(x-1,x-1)
2	(0,0)	(x,0)
2	(x-1,y-1)	(2*x-1,y-1)

Tabla 4.1: Composición de dos retinas utilizando dispositivo mapeador

Siendo $x= 2^n$, $y=2^m$ el tamaño en píxels de las imágenes de entrada, que se numerarán de 0..x-1 para la columna, 0..y-1 para las filas. Al componer la salida generada por dos retinas distintas, como vemos en la tabla 4.1, ambas retinas generan las mismas direcciones. Mediante un dispositivo mapeador, podemos convertir los eventos generados por una de ellas a otro rango de

direcciones diferente, para permitir su posterior mezcla mediante un dispositivo merger [RIVAS01] [RIVAS02].

Desde el punto de vista teórico, la implementación más sencilla del mapper, sin utilizar máquinas de estado (asíncrona), se representa en la ilustración 4.3:

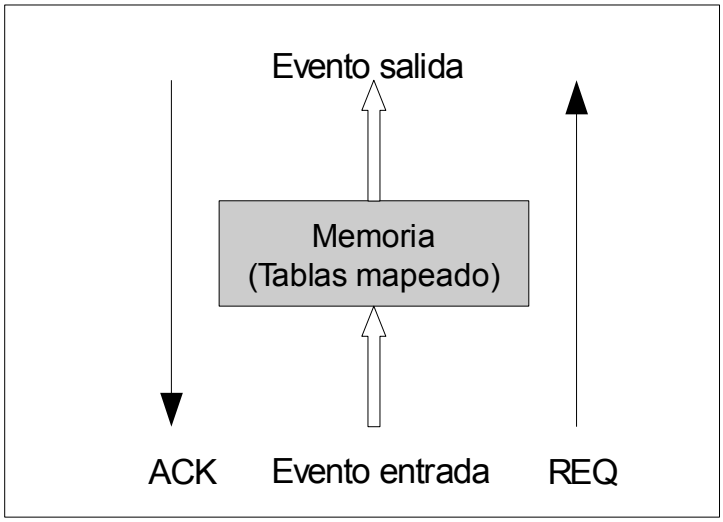

Ilustración 4.3: Diagrama mapeador asíncrono

Si se quiere combinar la imagen proveniente de dos retinas para obtener un mayor campo de visión horizontal, la imagen resultante tendría un tamaño $2^{n+1}x2^{m}$. Así para una de las dos retinas, hay que sustituir cada evento (i,j) por (i+n,j) de forma que al combinar los eventos de ambas retinas, en la imagen resultante se componga una imagen al lado de la otra, en lugar de una sobre otra.

El evento de entrada se conecta a las líneas de dirección de la memoria para indexar dentro de ella en función del evento de entrada. El contenido de dicha posición de memoria se emitirá en el bus de salida como nueva dirección del evento. En la realidad, este esquema asíncrono presenta algunos problemas:

- La memoria no presenta el evento de salida inmediatamente, por lo que el receptor podría recibir REQ cuando aún no hay una dirección válida en el bus. Hay que retrasar la señal REQ, insertando buffers, hasta que el retraso sea mayor al tiempo de acceso de la memoria.

- Este diseño no genera ACK por sí mismo, sino que propaga las señales REQ y ACK hacia los dispositivos anterior y siguiente. Al propagar esta señales introduce retrasos en la cadena AER que se van acumulando.

Para entender mejor esto último, si imaginamos varios mapeadores insertados entre un generador y un receptor AER, para cada evento de entrada, la señal REQ se iría propagando de uno a otro hasta llegar finalmente al receptor AER, que activaría su señal ACK, que se iría propagando a la inversa hasta llegar al generador. En el caso de ACK, no presenta mucho problema puesto que podría ser únicamente un elemento conductor pasivo (no necesita buffers, bastaría un cable de cobre), por lo que no introduciría más retraso en la cadena que el tiempo de propagación por el conductor. Sin embargo, la propagación de REQ sí es un problema, dado que el retraso entre REQ de entrada y de salida, se iría acumulando a lo largo de la cadena. Este retraso se consigue concatenando varios buffers o algún otro tipo de puertas de forma que el retraso total sea mayor al tiempo de acceso de la memoria.

Para solucionar estos problemas, el mapper se implemento utilizando un diseño síncrono de forma que al recibir un evento, se busca en la memoria y entonces se genera el REQ de salida para el siguiente elemento de la cadena, además de activar el ACK de entrada. Se esperará a recibir la respuesta de ACK del receptor antes de aceptar un nuevo evento. Al no tener una FIFO de entrada, no es capaz de recibir más eventos hasta que el receptor no haya procesado el evento anterior. La velocidad con la que el emisor emitirá los eventos dependerá de la velocidad con la que el receptor sea capaz de procesarlos, pero al no esperar la confirmación de ACK para activar el ACK de entrada, no introduce retrasos innecesarios.

Por otra parte, al hacer una sustitución 1:1 el tráfico de entrada será igual al tráfico de salida, por lo que no resulta tan interesante la presencia de una FIFO en la entrada, que por el contrario podría degenerar la información temporal de los eventos, al ocultar al emisor la velocidad real a la que están procesándose sus eventos. El esquema resultante sería el siguiente:

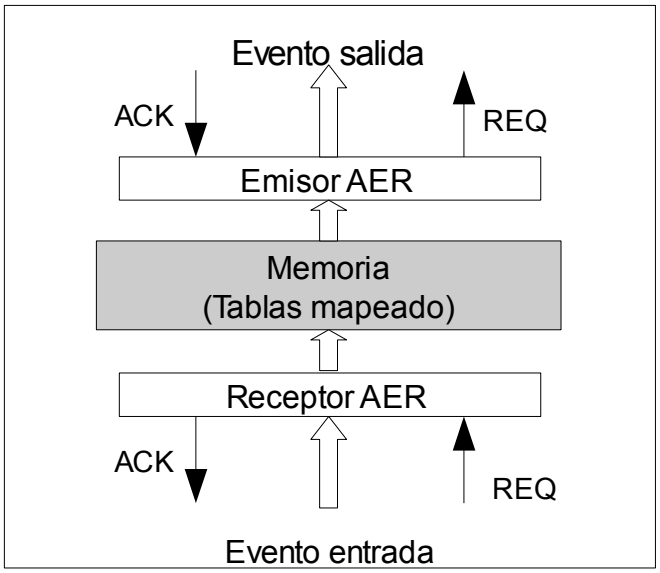

Ilustración 4.4: Mapeador básico

Esto permite realizar transformaciones en las direcciones de los eventos de forma que para cualquier dirección de entrada se le asigna otra dirección de salida. Esto permitiría por ejemplo ajustar el rango de direcciones generados por un dispositivo para combinar múltiples sensores similares, por ejemplo varias cámaras para obtener una imagen mayor a la proporcionada por una sola. Ubicando cada sensor en un espacio de direcciones diferente permitiría combinar, utilizando un dispositivo "merger" [RIVAS01][RIVAS02][GOMEZ03] o mezclador, los impulsos provenientes de los múltiples sensores.

4.2 Transformaciones realizadas por mapper

Dada la naturaleza del AER, utilizando un dispositivo mapper pueden realizarse algunas transformaciones básicas sobre la imagen, como pueden ser desplazamientos, invertir respecto un eje, o rotaciones, o algunas transformaciones de tamaño. Como el mapper se limita a sustituir unos eventos por otros, hay que calcular las posiciones de destino para los posibles eventos de entrada y almacenarla en la tabla de mapeado del mapper. En el caso de algunas rotaciones con ángulos arbitrarios distintos a múltiplos de 90º, escalado, etc., al realizar la transformación de las direcciones de entrada, puede ocurrir que la dirección de destino no sea entera, lo que significaría gráficamente que ese impulso no se corresponde con una dirección concreta, sino que haría aportaciones a las direcciones adyacentes. Para realizar este proceso de integración sería necesario un chip de convoluciones, o modificar el diseño del mapper- para que añadiese una probabilidad que indique que el evento debe ser o no enviado. Si a cada uno de los posibles eventos de salida se le asigna una probabilidad distinta, tenemos que para un evento de entrada, existe asociado un conjunto de eventos de salida, cada uno con una probabilidad distinta de ser emitido como respuesta a dicho evento de entrada.

4.2.1 Desplazamiento

Para realizar un desplazamiento un número entero de píxel en cualquiera de los dos ejes bastará con calcular las nuevas direcciones de eventos de forma que:

$$\begin{pmatrix} X' \\ Y' \end{pmatrix} = \begin{pmatrix} X \\ Y \end{pmatrix} + \begin{pmatrix} \Delta X \\ \Delta Y \end{pmatrix} \tag{4.1}$$

El segundo sumando representa el desplazamiento en ambos ejes que se sumará a la dirección original. Ambos deben ser enteros para que exista una equivalencia 1:1 entre los eventos de entrada y los de salida. Con el mapper probabilístico descrito más adelante, podrían aplicarse desplazamientos no enteros. El cálculo de las tablas de mapeado se realiza offline, utilizando para ello MATLAB (aunque se puede hacer con cualquier otro software), se debe calcular la transformación para todos los eventos de entrada posibles. Estos valores se almacenarán en las tablas de mapeado de la tarjeta USB-AER.

La anterior expresión puede modificarse un poco para que los píxels desplazados fuera de la nueva imagen vuelvan a aparecer por el extremo opuesto.

$$\begin{pmatrix} X' \\ Y' \end{pmatrix} = \begin{pmatrix} (X + \Delta X) \bmod X_{max} \\ (Y + \Delta Y) \bmod Y_{max} \end{pmatrix} \tag{4.2}$$

A modo de ejemplo se muestra el siguiente código MATLAB utilizado para la generación de las tablas. En el Apendice 11.3 se incluye el código utilizado para generar las tablas de todas las transformaciones:

```
function mapper=genera_desplaxy(destino,dx,dy,side)
    maxx=side;
    maxy=side;
    mapper=zeros(maxx*maxy,1);
    i=1;
    for oldy=0:maxy-1
        for oldx=0:maxx-1
            x=mod(oldx+dx,maxx);
            y=mod(oldy+dy,maxy);
            mapper(i)=y*maxx+x;
            i=i+1;
        end;
    end;
    save (destino,'mapper');
```

Una vez cargada dicha tabla en la memoria RAM del mapper puede observarse el resultado sobre la imagen original. En la ilustración 4.6 muestra el resultado de desplazar la imagen de la ilustración 4.5 en las dos dimensiones, horizontal y verticalmente. Para realizar dicha transformación se ha utilizado una configuración en serie, como la mostrada en la ilustración4.1 utilizando para ello una única capa de filtro.

Ilustración 4.5: Imagen original

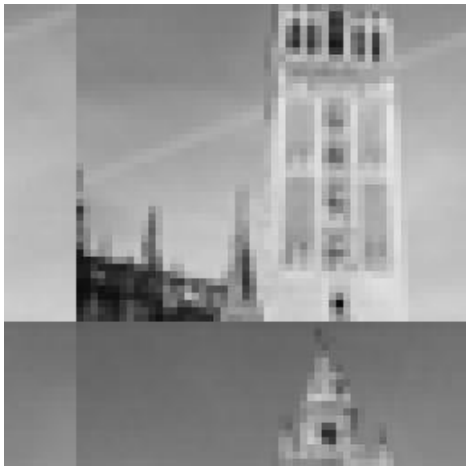

Ilustración 4.6: Imagen desplazada

4.2.2 Mirror o espejo

Para realizar una operación de mirror o espejo, el número de eventos enviados y recibidos es el mismo. Cada evento de entrada se sustituirá por el evento opuesto, considerando el centro de la imagen como eje de simetría para la inversión. La matriz de eventos se rellenará siguiendo alguna de las funciones siguientes, según se trate de un mirror horizontal:

$$\begin{pmatrix} X' \\ Y' \end{pmatrix} = \begin{pmatrix} X_{MAX} - X \\ Y \end{pmatrix} \tag{4.3}$$

De manera similar al mirror horizontal, para realizar un mirror vertical se aplica la siguiente transformación:

$$\begin{pmatrix} X' \\ Y' \end{pmatrix} = \begin{pmatrix} X \\ Y_{MAX} - Y \end{pmatrix} \tag{4.4}$$

Ilustración 4.7: Imagen invertida horizontalmente

Ilustración 4.8: Imagen invertida verticalmente

En la ilustración 4.7 podemos ver el resultado de invertir la imagen original mostrada anteriormente en la ilustración 4.5. En la figura 4.8 aplicamos el mismo filtro, pero en este caso para invertir la imagen verticalmente.

4.2.3 Rotación

Para rotar una imagen, hay que calcular el evento de destino para cada posible evento de entrada aplicando una matriz de rotación, de la siguiente forma:

$$\begin{pmatrix} X' \\ Y' \end{pmatrix} = (X \quad Y) * \begin{pmatrix} \cos\alpha & -sen\,\alpha \\ sen\,\alpha & \cos\alpha \end{pmatrix} \tag{4.5}$$

Esto rotaría la imagen respecto al punto correspondiente a las coordenadas (0,0), su origen de coordenadas, correspondiente a una esquina de la imagen. Para rotar la imagen respecto a un punto cualquiera, normalmente el propio centro de la imagen, hay que trasladar la imagen de manera que el centro de rotación deseado coincida con la posición (0 0), por lo que se resta a la posición de cada píxel, la posición del centro de giro. Una vez trasladada se realiza la rotación y nuevamente se le suma el desplazamiento para devolverla a su rango de direcciones original.

La expresión completa quedaría de la siguiente forma, incluyendo la traslación:

$$\begin{pmatrix} X' \\ Y' \end{pmatrix} = (X - \Delta X \quad Y - \Delta Y) * \begin{pmatrix} \cos\alpha & -sen\,\alpha \\ sen\,\alpha & \cos\alpha \end{pmatrix} + \begin{pmatrix} \Delta X \\ \Delta Y \end{pmatrix} \tag{4.6}$$

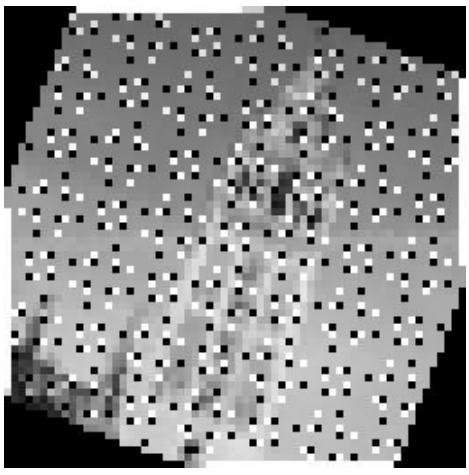

Ilustración 4.9: Rotación 20º

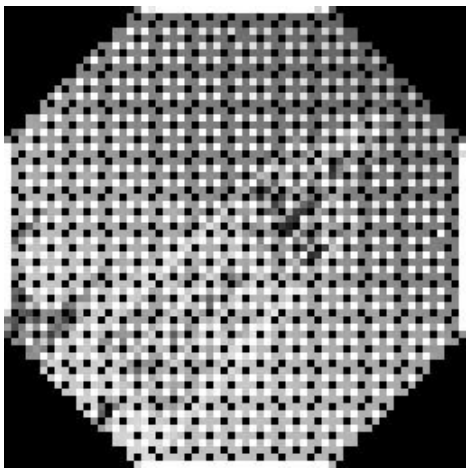

Ilustración 4.10: Rotación 45º

En las figuras 4.9, 4.10 y 4.11 se observa la rotación sobre la imágen modelo de entrada de un ángulo de 20, 45 y 90º respectivamente. Para la rotación de 90 grados, los valores del seno y del coseno son cero y uno respectivamente, por lo que el rango de direcciones de salida, al igual que el de entrada son valores enteros y la imagen resultante no presenta ninguna distorsión. Para otros ángulos arbitrarios, al truncarse la dirección de los eventos, aparecen píxels negros sin eventos, y píxels más claros (blanco). Esto se debe al error que cometemos al tener que truncar las direcciones de eventos no enteros. Esto mismo ocurre en imágenes como las de radar navales, las cuales se representan en coordenadas polares, lo clásico es interpolar para rellenar los huecos que quedan.

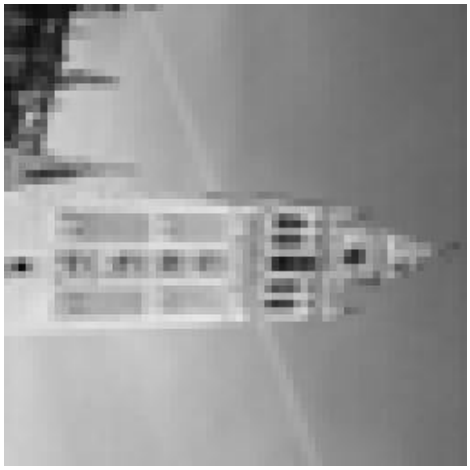

Ilustración 4.11: Rotación 90º

4.2.4 Escalado

Para aumentar el tamaño de una imagen, hay que aumentar el rango de direcciones generadas. Para convertir una imagen de NxN píxels a otra de MxM píxels, hay que convertir los eventos de entrada, en el rango 0..N, al rango de salida 0..M, por lo que habrá que multiplicar cada coordenada por el factor M/N. Para aumentar la imagen n veces, hay que aplicar la siguiente transformación.

$$\begin{pmatrix} X' \\ Y' \end{pmatrix} = \begin{pmatrix} X * n \\ Y * n \end{pmatrix} \tag{4.7}$$

Sin embargo, al ser las direcciones de entrada números enteros, las direcciones de salida estarán espaciadas n píxels, quedando los píxels intermedios a cero. Para solucionar esto es necesario que para un único evento de entrada se generasen n^2 eventos correspondientes. La solución a dicho problema se aborda en el siguiente capítulo.

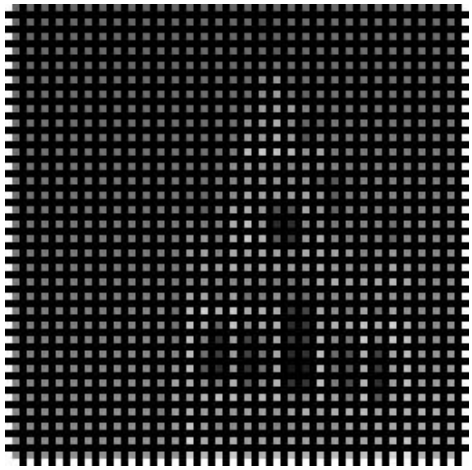

Ilustración 4.12: Imagen ampliada 2x (parte superior)

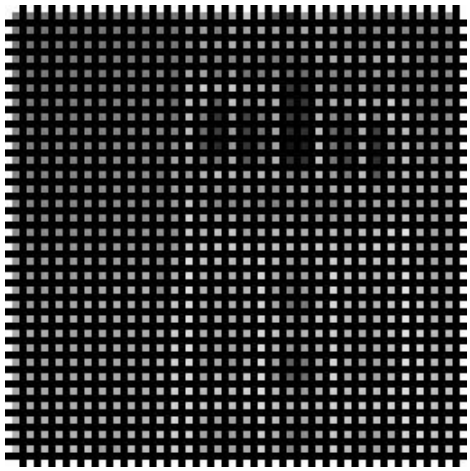

Ilustración 4.13: Imagen ampliada 2x (centro)

Para realizar la transformación inversa, y disminuir el tamaño de la imagen, el valor n debe ser menor que la unidad. Expresado como una división, para un factor n entero habría que aplicar la siguiente operación:

$$\begin{pmatrix} X' \\ Y' \end{pmatrix} = \begin{pmatrix} X/n \\ Y/n \end{pmatrix} \tag{4.8}$$

Hay que tener en cuenta que la luminosidad de la imagen se verá multiplicada por el factor n^2 entero debido a que varios eventos de entrada distintos convergerán en el mismo evento de salida. Al disminuir la imagen a la mitad, el resultado de dividir por dos la dirección x e y, será que las direcciones n y n+1 irán a generar el evento n/2 en ambos casos. Al escalarse en los dos ejes por igual, para una imagen mitad de tamaño, se recibirán 4 direcciones diferentes como una misma, por lo que la imagen será 4 veces más luminosa:

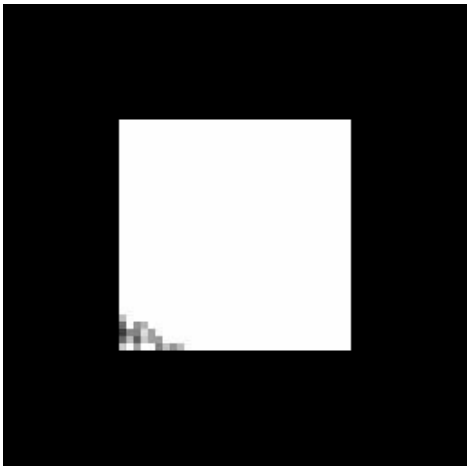

Ilustración 4.14: Imagen reducida 0.5x

Ilustración 4.15: Imagen 0.5x con luminosidad ajustada

Para un factor de reducción n, existirán n^2 direcciones de entrada correspondientes al mismo evento de salida, por lo que la luminosidad de la imagen aumentará n^2 veces. Para corregir este aumento de brillo, es necesario recurrir al uso del mapper probabilístico, de forma que se asocie a todos los eventos de salida la probabilidad $1/n^2$, por lo que en promedio, el número de eventos generados para un píxel de salida, sea similar a la media del número de eventos recibidos para los n^2 píxels correspondientes.

Con el mapper básico, es necesario que el factor de escalado n sea entero. Sin embargo, utilizando el mapper probabilístico, es posible interpolar los valores para realizar escalados siendo n no necesariamente un número entero.

4.3 Mapper probabilístico multievento

En el mapper probabilístico multievento [CAVIAR01], para cada evento es posible generar varios eventos de salida para cada evento de entrada. Esto permite que los eventos provenientes de un único píxel actúen sobre varios píxels de la imagen de salida. El valor de un píxel de salida esta determinado por lo tanto por el valor de no uno, sino varios píxels de la imagen original. Cada evento de salida lleva asociada una probabilidad de ser emitido que se comparará con un generador de números aleatorios para que no siempre que se reciba el evento de entrada se genere un evento de salida, de manera aleatoria, pero pudiendo controlarse la probabilidad con que el suceso ocurre. Esto permite ponderar en qué medida contribuye cada evento de entrada con un evento de salida.

Sin embargo, al asignar una probabilidad solo podemos disminuir el número de eventos generados. Cada evento también lleva asociado un factor de repetición que indica cuantas veces debe enviarse un determinado evento de salida.

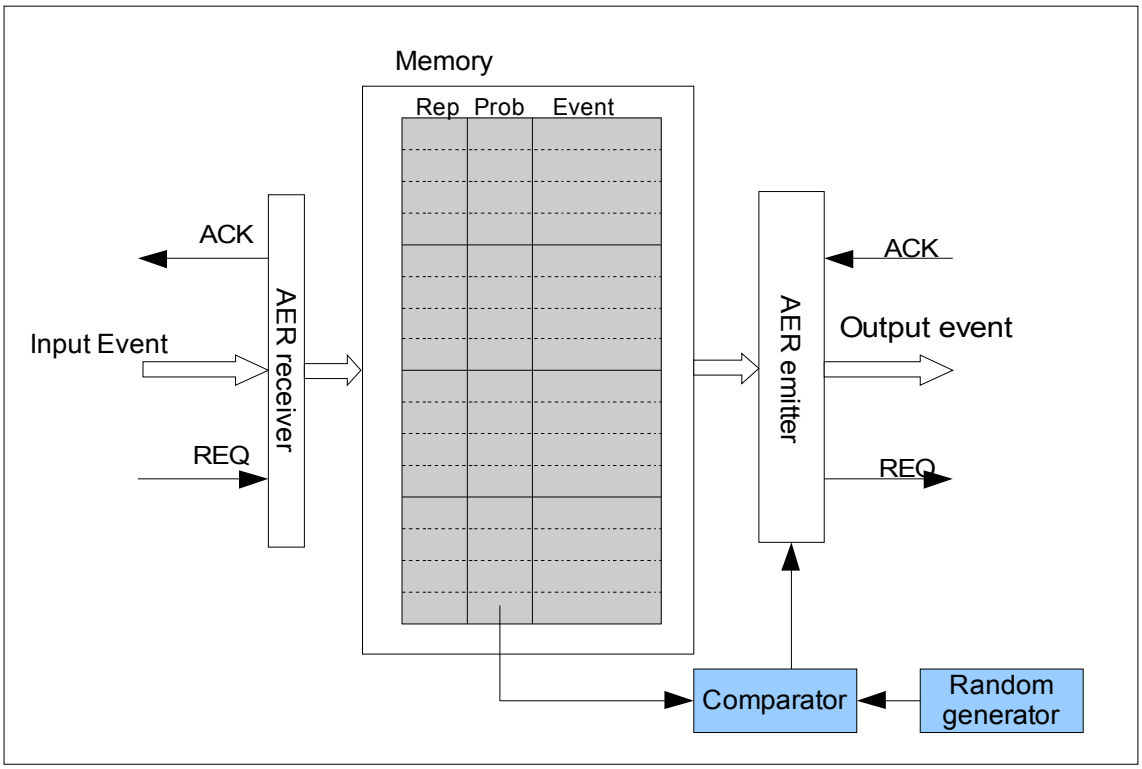

Ilustración 4.16: Diagrama bloques mapper probabilístico

En la ilustración 4.16 se observa el diagrama de bloques de dicho mapper. Cada entrada en la memoria esta dividida en tres campos. El primer campo indica el número de repeticiones asignado a un evento. El siguiente campo representa la probabilidad de que dicho evento se emita. Por último, el tercer campo indica la dirección del evento que se va a emitir. Cada evento de entrada viene asociado además a más de un evento de salida, por lo que habrá varias filas asociadas a cada evento de entrada. Por comodidad a la hora de hacer la implementación en hardware, se impone que el número de eventos asociados a cada evento de entrada sea el mismo para todos los eventos de entrada, y que sea potencia de dos, 2n, de forma que el procedimiento de indexar la tabla consiste en desplazar la dirección de entrada n bits a la izquierda, y completar estos n bits menos significativos con un índice, o contador, que irá recorriendo todos los eventos de salida asociados al evento de entrada recibido. Existe un bit, en la memoria RAM para cada evento que indica si es el último evento de la lista. Si este bit está activo se termina de recorrer la lista, permitiendo asignar a cada evento de entrada un número menor de eventos de salida. El comparador, compara el valor de la probabilidad con un valor pseudoaleatóreo, generado por un registro LFSR. Del resultado de esta comparación se decide si se envía o no el evento correspondiente.

Para entender el mapper probabilístico partimos de lo siguiente: Cada evento de entrada tiene varios eventos de salida. Cada evento de salida tiene un número de repeticiones, y una probabilidad que se comprueba en cada repetición. En función del resultado de dicha comparación, la dirección del evento de salida se cargará en el emisor AER para ser transmitido por el bus.

1. *Se indexa en memoria usando la dirección del evento. Habrá 2^n direcciones adyacentes para cada evento de entrada, aunque no todas tienen que estar ocupadas.*
2. *Se obtiene un número del LFSR que se comparará con la probabilidad asociada a dicho evento. Si el valor del LFSR es menor a la probabilidad asociada el evento se emite el evento. Se repite el proceso el número de repeticiones indicado.*
3. *Se pasa a la siguiente posición de la lista de eventos si el último evento enviado no era el último. En caso contrario, se vuelve a comenzar con el siguiente evento.*

En la ilustración 4.17 se representa gráficamente en forma de diagrama de flujo el comportamiento que acabamos de describir.

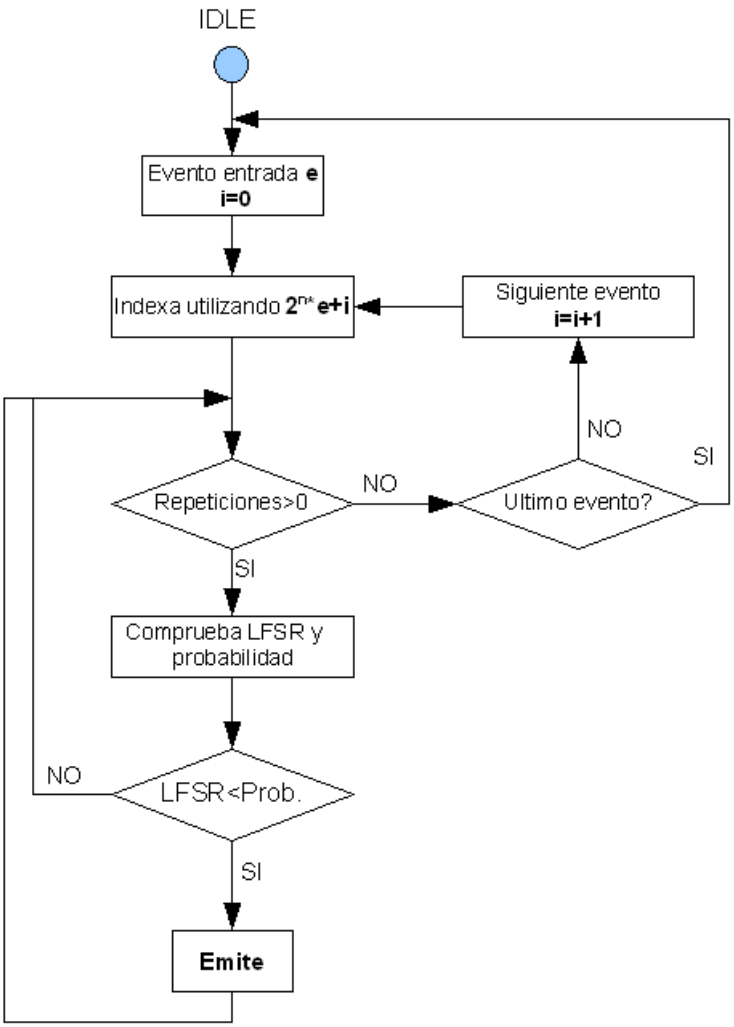

Ilustración 4.17: Diagrama de flujo del mapper probabilistico

4.4 Interpolado utilizando la probabilidad del evento

Ajustando la probabilidad de emitir un evento, podemos modificar el valor de dicho píxel reduciendo la cantidad de eventos que se generan, en una proporción indicada por la probabilidad.

Para ganancias inferiores a uno, se utilizará un intervalo de probabilidad proporcional a la ganancia deseada. Si por ejemplo la salida de aplicar una transformación fuese: X=32.4, Y=14.7, el evento aportaría un 60% a X=32, y un 40% a X=33, y por otra parte 30% de Y=14 y 70% de Y=15. De forma que habría que configurar la memoria con 4 eventos. Suponemos x=32, y=14, Px=0.4, Py=0.7. Lista de eventos asociada a un único evento.

Pcalc	***X***	***Y***	***Probabilidad***
0,18	x	y	(1-Px)*(1-Py)
0,42	x	y+1	(1-Px)*Py
0,12	x+1	y	Px*(1-Py)
0,28	x+1	y+1	Px*Py

Tabla 4.2: Distribución probabilidad para interpolación de eventos

El evento se emite si el valor del generador de números aleatorios, que en una FPGA puede implementarse utilizando un registro LFSR, es inferior a la probabilidad de emitir el evento. Tanto se emita como no, se pasaría a evaluar la siguiente probabilidad mientras queden disponibles.

Puede utilizarse para cambios en contraste, ajustando la probabilidad de un único evento de salida, de forma que no siempre emita. Para reducir una imagen, pueden promediarse píxels adyacentes ajustando en qué porcentaje participa cada evento de entrada en el evento de salida. Por ejemplo, observamos la siguiente tabla con la probabilidad asociada a cuatro eventos de entrada diferente, para generar una misma dirección de salida.

Evento entrada	***Evento salida***	***Probabilidad***
(x,y)	(x/2,y/2)	1/4
(x,y+1)	(x/2,y/2)	1/4
(x+1,y)	(x/2,y/2)	1/4
(x+1,y+1)	(x/2,y/2)	1/4

Tabla 4.3: Ejemplo probabilidad para zoom out

Como se verá más adelante, se pueden realizar convoluciones sin necesidad de integrar y generar. Cada uno de los pesos indicados en el kernel de convolución establece la probabilidad de emitir cada evento. La probabilidad esta acotada entre los valores [0,1], por lo que los coeficientes del kernel de convolución deberían estar también entre [0,1]. Utilizando la repetición se puede extender a todos los coeficientes positivos. Para poder utilizar también coeficientes negativos, se recurrirá a eventos negativos, como se describe más adelante.

Dada la naturaleza del bus AER, algunas transformaciones sencillas no son posibles con el mapper multievento probabilístico, como son la modificación de brillo:

- Modificación del brillo. Modificar el brillo significa desplazar el histograma de una imagen, sumarle un valor constante a la luminosidad de cada píxel. Para un nivel de píxel negro, no se generará ningún evento. Modificar el brillo significaría que para un píxel sin eventos, el mapper debería "inventarse" eventos. En general, para cualquier píxel, el número de eventos generados para cada píxel en un tiempo determinado debería ser fijo, e independiente de la actividad de dicho píxel.

$$Y = K_{contraste} * X + Off_{brillo} \tag{4.9}$$

Para modificar el brillo de una imagen, se podría recurrir a dos soluciones distintas, una basada en frames y otra no.

La solución basada en frames consistiría en integrar la imagen durante un tiempo de frame, y una vez que se tiene la imagen reconstruida, realizar la modificación de brillo (sumarle un valor constante a cada píxel) y mediante un generador convertir la imagen de nuevo en un stream AER. Esto evidentemente es un truco, pues no es un procesado AER.

La solución no basada en frames necesitaría de un generador AER, que generara eventos a partir de una imagen plana con todos los píxels fijados al valor Off_{brillo}, para posteriormente combinarla con la imagen original. Dado que todos los pixels de esta imagen a superponer son iguales, no es necesario emplear un convertidor frame a AER completo, bastaría con generar todas las direcciones de la imagen con una frecuencia fija igual al valor Off_{brillo} deseado. Mediante un dispositivo “merger” [RIVAS02][RIVAS01] se combinarían ambas imágenes resultando la imagen original, pero con la luminosidad añadida por la segunda imagen plana.

- Convoluciones con pesos negativos: Los pesos negativos no pueden modelarse ajustando la probabilidad debido a que no tiene sentido un valor de probabilidad negativo. Para poder realizar convoluciones con pesos negativos se recurre al empleo de eventos con signo. Ampliando el espacio de direcciones de los eventos en un bit, puede indicarse un valor de signo para cada evento que se envía, de forma que para una misma neurona receptora existen dos posibles direcciones AER, una correspondiente al evento positivo o potenciador, y otra a un evento negativo o inhibidor que contrarrestaría a su evento positivo equivalente.

$$Y_{ij} = \sum (X_{ij} * Kern_{ij}) + Offset\,,\ siendo\ Kern_{ij} < 0, Offset \neq 0 \tag{4.10}$$

Más adelante en este capítulo se describe el proceso completo para realizar convoluciones para cualquier peso deseado, introduciendo un nuevo tipo de eventos, eventos con signo, con ayuda de algo de hardware adicional o modificando los ya existente para permitir la simplificación de eventos positivos y negativos.

4.4.1 Rotación ángulo arbitrario

Si se configuran las tablas de mapeo aplicando las transformaciones básicas descritas anteriormente para el mapper no probabilístico, añadiendo la interpolación descrita en el apartado anterior, podemos rotar la imagen un ángulo arbitrario. Vemos que la imagen de la ilustración 4.19 aparece un poco distorsionada por un ruido aleatorio añadido. Este ruido se debe al generador aleatorio que calcula la probabilidad de emitir eventos.

Ilustración 4.18: Imagen original

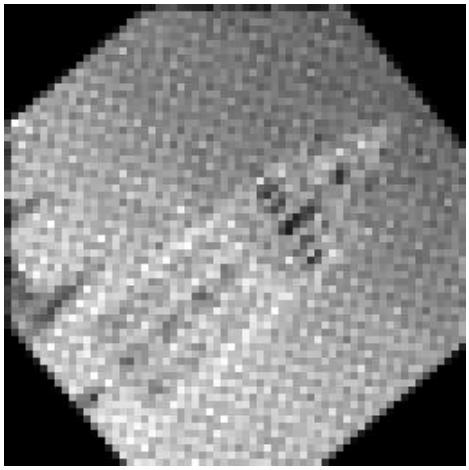

Ilustración 4.19: Rotación ángulo arbitrario(50º)

Si repetimos la transformación para una imagen mayor, de 128x128, mostrada en la ilustración 4.20, vemos que tras realizar la transformación (ilustración 4.21) el error aunque aparece, es menos evidente, dado que ahora hay cuatro píxels por cada uno de la imagen anterior (equivalente aumentar la señal para mejorar SNR).

Ilustración 4.20: Imagen original 128x128

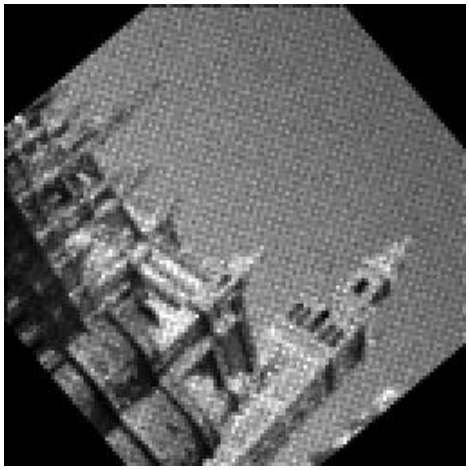

Ilustración 4.21: Imagen 128x128 rotada 50º

4.4.2 Desplazamiento

Igual que en el apartado anterior, repitiendo la transformación de desplazamiento para un desplazamiento no entero obtenemos la imagen desplazada. Según como se configure la matriz de mapeado, los pixels desplazados fuera de la nueva imagen pueden volver a aparecer o no por el extremo opuesto (ecuaciones 4.1 y 4.2) obtenerse los siguientes resultados:

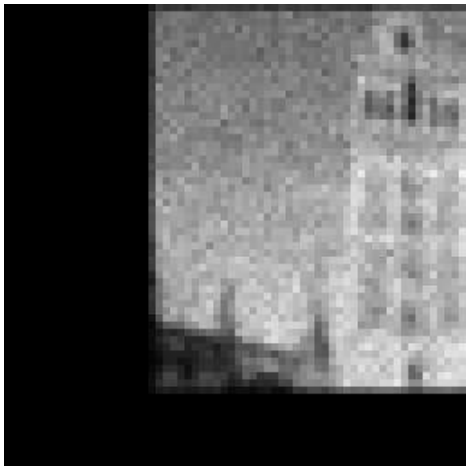

Ilustración 4.22: Desplazamiento sin clipping

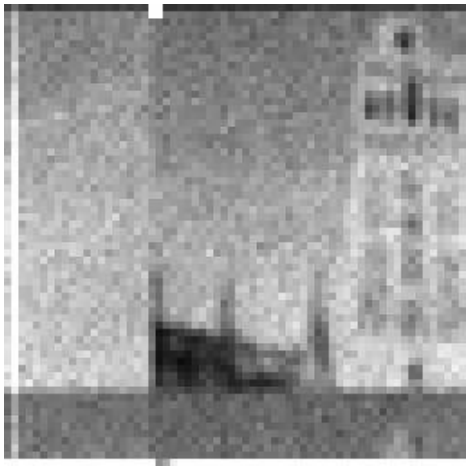

Ilustración 4.23: Desplazamiento con clipping

Como en el apartado anterior, cada píxel desplazado de la imagen original, puede corresponder a una dirección entera o no en la imagen resultante. Los píxels en posiciones no enteras tienen que ser interpolados en un entorno de hasta cuatro píxels en la imagen de destino. Cada uno de estos píxel de la imagen de destino corresponderán a su vez a varios píxel de la imagen original. Vemos que aparece el error introducido por el generador aleatorio superpuesto a la imagen desplazada. En este caso, las ventajas de poder desplazar cantidades no enteras no parece compensar el ruido añadido a las imágenes.

Para desplazamientos enteros, al no tener que interpolarse ningún píxel, el resultado sería similar al mostrado anteriormente en la ilustración 4.6 para el mapper básico, al ser uno la probabilidad asociada a cada evento

4.4.3 Escalado (zoom in o zoom out)

En este apartado se estudia la operación de escalado, cuando el factor de escala utilizado no tiene que ser necesariamente potencia de dos, como ocurria en el mapper básico, sino que puede ser cualquier valor entero o no. Los resultados presentan el mismo ruido que en los casos anteriores, aunque con varias ventajas respecto al mapper básico.

Por una parte, al ser multievento, un evento de entrada puede generar varios eventos de salida por lo que ya no aparecerían huecos en blanco. El poder aplicar una probabilidad nos permite interpolar los valores de salida. Si el factor de escala es mayor que 1, con que aumentamos la imagen, cada píxel original puede ocupar un entorno de 2x2 o incluso 3x3 píxels en la imagen de destino. Si el factor es mayor que dos, este entorno se aumenta a 3x3 o 4x4 según el caso.

Por otro lado, si el factor es menor a la unidad, al reducirse el tamaño de la imagen habrá eventos correspondientes a distintos píxels en la imagen original que coincidirán en el mismo píxel de destino. Un píxel de la imagen original ocupará un píxel, o como máximo un entorno de 2x2, según corresponda Para evitar que al disminuir el tamaño de la imagen, debido una mayor concentración de eventos, por ser la imagen resultante de menor tamaño, aumente su luminosidad, los coeficientes de probabilidad deben ser escalados además en función de la reducción de tamaño de la imagen, para que la densidad media de eventos por píxel sea la misma que en la imagen original.

Para el caso contrario, en el cual aumentamos el tamaño de una imagen el efecto sería el opuesto.

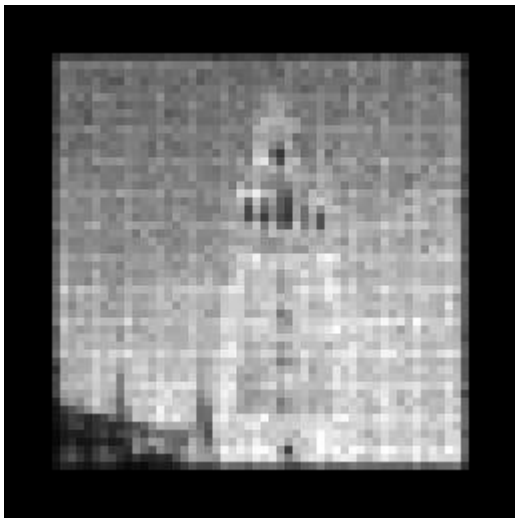

Ilustración 4.24: Imagen escalada 80%

Ilustración 4.25: Imagen escalada 120%

4.4.4 Aumento/Disminución del contraste.

Para aumentar el contraste de una imagen hay que multiplicar la luminosidad de cada píxel por un factor. Si el factor es la unidad, el contraste de la imagen permanece inalterado, para valores por debajo de la unidad la imagen se oscurecerá, y para factores por encima de la unidad se iluminará. Cuando se trata de imágenes AER, la luminosidad de cada píxel viene determinada por el número de eventos de cada píxel.

Al utilizar el mapper probabilístico para reducir el contraste de una imagen habrá que emitir menos eventos de los que se reciben. Para ello se fija la probabilidad de emitir un evento igual al factor de reducción. Para un factor de 0.8 (80%), la probabilidad de emitir el evento será del 80% (el valor a comparar con el LFSR será el valor máximo 255 multiplicado por 0.8, siendo en este ejemplo 204).

Cuando por el contrario se quiere aumentar el brillo de la imagen, multiplicando por un factor mayor a la unidad, hay que aumentar el número de eventos que aparecen en el bus. Para esto se utilizará la posibilidad de repetir un evento que se emite varias veces. Ajustando adecuadamente el factor de repetición y la probabilidad asociada, se puede conseguir una modificación del brillo para cualquier factor deseado.

$$\begin{aligned} Repeticiones &= ceil(factor) \\ Probabilidad &= factor / Repeticiones \end{aligned} \tag{4.11}$$

Por ejemplo, para un factor 1.5, habrá que repetir el evento 2 veces, con una probabilidad asociada del 75% (dividiendo 1.5 por 2, resulta 0.75).

Ilustración 4.26: Imagen 50% luminosidad

Ilustración 4.27: Imagen 90% luminosidad

Ilustración 4.28: Imagen 150% luminosidad

4.4.5 Otros tipos de transformación

Además de las transformaciones analizadas y descritas hasta ahora, utilizando el mapeador probabilístico podemos realizar cualquier otro tipo de transformación que impliquen la traslación de los píxels entre la imagen original y de destino. En los apartados anteriores hemos visto las transformaciones más comunes como son la rotación de una imagen, su traslación y su escalado. Aunque las tablas de mapeado se han calculado hasta ahora para realizar la misma transformación sobre todos los puntos de una imagen, en realidad puede aplicarse una transformación diferente sobre cada uno de los distintos píxels de la imagen de entrada. En las entradas asociadas a un determinado píxel de entrada, se indica el píxel o píxels de destino para dicho evento. Si el píxel de destino no es una dirección entera, como pasaba en los casos anteriores, debe interpolarse entre los píxels circundantes ajustando la probabilidad adecuada a cada uno de ellos.

Hay que tener en cuenta además, que si un conjunto de píxels ubicados en una determinada superficie de una imagen de entrada, se condensan en un área inferior, más pequeña, de la imagen de salida, al reducir el área, estamos aumentando la densidad de eventos por unidad de superficie, que se traduce en un incremento en la luminosidad media de dicha área. Para evitar esto, hay que ajustar la probabilidad de igual forma, para que la densidad de eventos por área siga siendo la misma que en la imagen original, o al menos sea homogénea respecto al resto de la imagen, para evitar oscurecimientos o aumentos de brillo por zonas, debido a la compresión o dispersión en los eventos.

Como ejemplo, vamos a aplicar la siguiente transformación sobre una imagen. Esta ecuación se corresponde a un escalado progresivo del ancho de la imagen, en función de la coordenada Y, hasta un 70% menor del tamaño original en la parte inferior de la imagen. El elemento Xmax/2 que se resta antes de la transformación y se vuelve a sumar para que el centro del escalado sea el eje X central de la imagen.

$$\begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} (X' - \frac{X_{max}}{2}) * (1 - \frac{0.7 * Y'}{Y_{max}}) + \frac{X_{max}}{2} \\ Y' \end{pmatrix} \tag{4.12}$$

En la ilustración 4.29 se observa el resultado de aplicar dicha transformación sobre la imagen original de patrón utilizada para ilustrar todas las transformaciones anteriores. Vemos como al comprimirse los eventos debido a la transformación, la luminosidad de la imagen se va incrementando. En la parte inferior de la imagen, Y' es igual a Ymax, por lo que el factor que multiplica la coordenada X es la imagen es 1-0.7=0.3, o el 30% del tamaño original.

Ilustración 4.29: Efecto sobre la luminosidad del aumento de la densidad de eventos

La zona inferior de la imagen, aparece saturada en blanco al concentrarse los eventos en un área menor de imagen. En la ilustración 4.30, se anula este defecto aplicando cierta corrección a la hora de calcular la probabilidad asociada a cada evento.

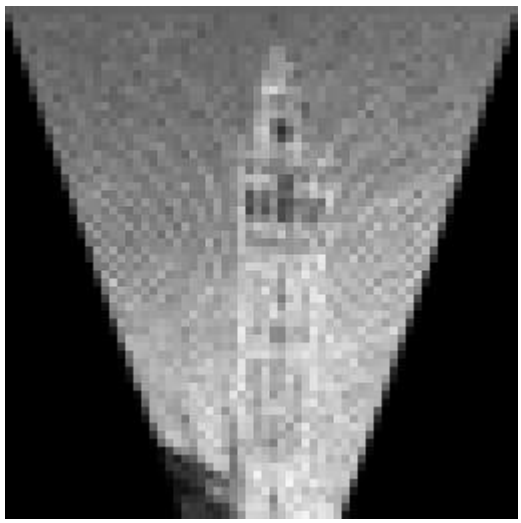

Ilustración 4.30: Luminosidad corregida con la probabilidad

En este caso, la probabilidad inicial de cada pixel viene dada por la siguiente expresión:

$$Prob = 1 - \frac{0.7 * Y'}{Y_{max}} \tag{4.13}$$

Para realizar la transformación ojo de pez, se convierte las coordenadas de cada píxel de coordenadas cartesianas a coordenadas polares. El parámetro R define el radio del efecto ojo de pez, mientras que radio y radio' definen el radio original del píxel, y el radio tras la transformación respectivamente.

$$radio' = \frac{(e^{\frac{radio}{s}} - 1)}{w}; s = \frac{R}{\log(w * R + 1)} \tag{4.14}$$

Para el ejemplo siguiente, se ha fijado w=0.025. Este parámetro indica la intensidad de la transformación. El resultado se muestra en la ilustración 4.31

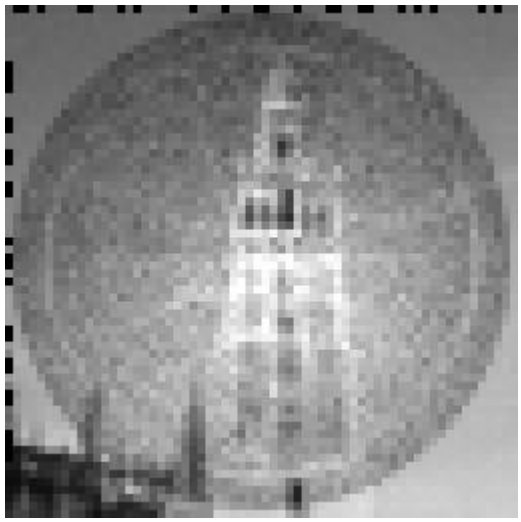

Ilustración 4.31: Transformación ojo de pez

4.4.6 Convoluciones

A diferencia del mapper no probabilístico, la posibilidad de ponderar la influencia que tienen varios eventos de entrada sobre un evento de salida, o un único evento de entrada sobre un grupo de eventos de salida, como caso particular, nos permite realizar la convolución de una imagen AER que se recibe de un generador.

$$Y(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} K(m,n) * X(i+m, j+n) \tag{4.15}$$

Esta expresión de la convolución indica que un píxel resultante Y(i,j) es el resultado de aplicar el kernel K(m,n) al entorno de X(i,j). A partir de esta expresión, si nos concentramos en un determinado evento de entrada X(i,j), este evento influirá en un entorno de Y(i+m,j+n), modulado por el kernel K(m,n), siendo m y n los índices del sumatorio, tanto positivos como negativos. Por lo tanto tenemos la inversa a la función anterior, que relaciona un píxel de entrada con un grupo de eventos de salida posibles, ponderados por una probabilidad de ser emitidos igual al índice del kernel K correspondiente:

$$X(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} K(m,n) * Y(i+m, j+n) \tag{4.16}$$

En la convolución de imágenes, tradicionalmente el kernel tiene unas dimensiones impares, que establecen un elemento centro de la matriz, correspondiente a las coordenadas (m,n)=(0,0). Para un kernel de convolución de tamaño t, siendo K(-t/2..t/2, -t/2..t/2) podemos realizar un cambio en el origen de coordenadas, de manera que K(0..t,0..t) de forma que la expresión anterior quedaría:

$$X(i,j) = \sum_{m=0}^{t} \sum_{n=0}^{t} K(m,n) * Y\left(i+m-\frac{t}{2,}\, j+n-\frac{t}{2}\right) \tag{4.17}$$

El factor -t/2 mantiene el centro de la imagen de salida centrado respecto a la imagen de entrada. Si suponemos que el espacio de direcciones de entrada es independiente del de salida, podría eliminarse quedando

$$X(i,j)=\sum_{m=0}^{t}\sum_{n=0}^{t}K(m,n)*Y(i+m,j+n) \tag{4.18}$$

Al no existir un elemento central, el tamaño de los kernel podrá ser cualquiera, y no necesariamente cuadrado.

La limitación que surge ahora, es que el kernel de convolución que podemos implementar debe tener todos los pesos positivos. Para pesos entre 0..1, se escoge una probabilidad equivalente. Si el peso fuese mayor, que la unidad, podemos adoptar la misma solución del apartado anterior, utilizando conjuntamente repeticiones+probabilidad. Cada evento de entrada participará en mayor o menor medida en la generación de nuevos eventos, pero no puede contrarrestar el efecto de otros píxels, dicho de otra forma, no puede haber eventos negativos que eliminen eventos.

$$Y_{ij}=\sum X_{ij}*Kern_{ij},\ siendo\ Kern_{ij}\geq 0 \tag{4.19}$$

Recordando la propiedad distributiva de la convolución:

$$f*(g+h)=f*g+f*h \tag{4.20}$$

Si tenemos un kernel K con pesos negativos, podemos suponer K'=(K+C), de manera que K' es el kernel K con un valor añadido C a todos sus elementos, de manera que C=abs(min(K)) para garantizar que K' tiene todos sus elementos positivos

$$M*K=M*[K'+(-C)]=M*K'+M*(-C) \tag{4.21}$$

Según esta expresión, podemos calcular la convolución I*K teniendo K pesos negativos, si sumamos el resultado de aplicar la convolución I*K' y sumándole -C*I. Si nos centramos en este segundo miembro, al ser la matriz (-C) una matriz cuadrada con todos los elementos igual al escalar -c:

$$M*(-C)=M*(-c)*I=(-c)*M \tag{4.22}$$

Por lo que no basta con procesar M*K' y después sustraerle el offset añadido al kernel, es necesario calcular por otro lado -c*M, que después se añadiría a la imagen resultante. Esto nos lleva a que en realidad aplicando esta transformación no solucionamos el problema de los eventos

negativos.

Aplicar la transformación K'=(K+C), supone calcular una convolución K'*M, calcular c*M, que no sería otra convolución, sino un ajuste de brillo sobre la imagen original, que después se restará de la convolución calculada. Esto es por tanto propagar dos imágenes distintas que después se combinan.

Para solucionar esto se recurre a utilizar eventos especiales, que se transmiten por neuronas distintas, y por tanto con direcciones distintas. Por lo tanto, cuando nos encontramos con un elemento del kernel con valor negativo, dado que la probabilidad no tendría sentido negativa, fijamos la probabilidad con el valor absoluto de dicho elemento. El signo se traslada al evento, que en lugar de asignarle el evento positivo de salida, se le asigna su equivalente con signo negativo.

Al duplicar el espacio de direcciones de salida añadiendo un bit de signo a la dirección, estamos en realidad generando dos imágenes simultáneas, una que contiene la información de la convolución asociada a los coeficientes positivos y otra correspondiente a los coeficientes negativos. Tiene la ventaja de que no es necesario realizar dos operaciones distintas, sino una única convolución, aunque en ambos casos sea necesario un último procesado de anulación de los eventos negativos por sus equivalentes positivos. El aumentar un bit el rango de direcciones de salida no implicará normalmente la necesidad de hardware adicional.

Como ejemplo sencillo se va a calcular el resultado de aplicar a la imagen de la giralda la matriz de convolución [1 0; 0 -1]. Para eso hay que programar el mapper, que ante la llegada el evento (i,j), emita dos eventos (i,j) y -(i+1,j+1), uno positivo y otro negativo. El resultado obtenido se muestra en la ilustración 4.32:

Ilustración 4.32: Mitades positiva(izquierda) y negativa(derecha)

La imagen resultante es doble, la mitad de la izquierda representa la parte positiva de la convolución, y la mitad de la derecha la parte negativa. La mitad de la izquierda esta desplazada una posición a la derecha y abajo, esto es debido al desplazamiento del coeficiente -1 en la matriz de convolución.

Si en el momento de la integración, se combinan ambas mitades, bien integrando por separado ambas imágenes y después sumándolas, o modificando el integrador de forma que además de incrementarse pueda decrementarse ante los eventos negativos, obtenemos el resultado de la convolución, que se muestra en la ilustración 4.33.

Ilustración 4.33: Imagen combinada

Sin embargo, en este caso, la convolución se termina de calcular en el proceso de integración. Si queremos incluir estas convoluciones dentro de una cadena AER, es necesario terminar de procesar esta información. Una posible solución sería insertar un elemento compuesto por un integrador y un generador de forma que se vaya integrando el valor que tendría el frame, y volviendo a generar un flujo AER que represente la imagen. Sin embargo este procedimiento es un poco contrario a la naturaleza del AER, ajena al concepto de frame introducido en los sistemas digitales. Para tratar el flujo AER como un continuo, sin la discretización en frames que supone el proceso de integrar+generar, puede insertarse un dispositivo destinado a calcular "on-the-fly" el resultado de la suma de ambas imágenes.

El concepto es muy sencillo. Si nos centramos en una única neurona que queremos simplificar sus estímulos positivos y negativos, la idea intuitiva sería ir emitiendo los eventos que

llegan, pero no inmediatamente, sino con un pequeño retraso, de manera que este pequeño retraso permite que si durante ese tiempo llegan eventos negativos, contrarrestar los positivos de forma que el flujo resultante, en el caso ideal, no tendría eventos contradictorios, sino que los positivos se anularían con los negativos, y solo se activaría una neurona, la positiva o la negativa, en función del flujo neto resultante para ese píxel. De esta forma además el tráfico de salida sería mínimo en el caso ideal.

Dado que los eventos de entrada para un determinado píxel se reciben homogéneamente repartidos en el tiempo, los eventos de salida de la convolución, tanto positivos como negativos también estarán homogéneamente repartidos en el tiempo (despreciando los errores introducidos por el generador aleatorio). Si por ejemplo un kernel de 16 elementos, tiene 15 elementos positivos, y 1 negativo, en promedio, de cada 16 eventos recibidos habrá 1 negativo y el resto positivo. Aparecerán diferencias respecto a la distribución ideal, pero nunca van a aparecer 150 eventos positivos y después 10 negativos, salvo que el generador emitiera primero todos los eventos de cada neurona antes de pasar a la siguiente, no estando el flujo de entrada repartido homogéneamente, y por tanto, se cumplirían los supuestos iniciales.

En el caso de un flujo de entrada homogéneo, podemos suponer que el rango de variación o de desorden entre eventos positivos y negativos esta limitado por el tamaño y características del kernel de convolución, y la relación entre proximidad espacial y proximidad temporal entre eventos de entrada.

Para implementar este esquema digitalmente, cada elemento simplificador, recibirá dos posibles eventos, uno positivo y otro negativo, y puede emitir ambos eventos. En el caso más simple, cada simplificador estaría implementado con un único bit, aunque la idea se generaliza después para un contador de n bits.

El funcionamiento sería el siguiente. Inicialmente el bit se inicia a 0. Ante la llegada de un evento positivo, el bit se pondría a uno, pero no se emitiría ningún evento, hasta la llegada de un segundo evento positivo, que al encontrar el bit a 1 se emitiría inmediatamente. Así ocurre con todos los eventos de entrada hasta la llegada de un evento negativo, que colocaría el bit de nuevo a cero, de forma que el siguiente evento positivo no se enviaría, sino que se emplearía en cambiar el valor del bit nuevamente. En el caso de llegar un evento negativo y valer el bit cero, se emitiría un evento negativo.

Este esquema simple de un bit tiene la limitación que no es capaz de resolver adecuadamente si llegarán dos eventos negativos seguidos, aunque en promedio hubiera una mayoría positiva, o al

revés. Para aumentar la tolerancia a la posibilidad de eventos desordenados, puede extenderse este esquema a un contador de n bits, donde sólo se emiten eventos positivos o negativos cuando el valor del contador se desborda en alguno de los dos sentidos. Para un contador de n bis, tenemos 2^n valores, por lo que si se inicia a su valor intermedio, retrasa la emisión de cualquier evento en (2^n)/ 2 eventos, añadiendo un tiempo de latencia adicional a los eventos, aunque con el sistema funcionando en continuo no representa más problema que la latencia añadida en si. Para un número de bits bajos, en la medida que el sistema se aleje de las suposiciones de homogeneidad temporal iniciales, menor será la efectividad a la hora de anular eventos.

El resultado de simplificar el flujo resultante de la convolución tal y como se ha descrito, sería la ilustración 4.34:

Ilustración 4.34: Imágenes positiva y negativa simplificadas

La mitad de la izquierda representa otra vez la parte positiva de la convolución, y la derecha la parte negativa, sin embargo en este caso apenas hay eventos (áreas negras) y solo aparecen a la izquierda los píxels cuyo valor sea positivo (resultado de la convolución positivo) y a la derecha los negativos. En el caso ideal ningún píxel debe aparecer en ambas mitades. Aumentar el tamaño del contador reduce los eventuales errores provocados por errores en la distribución temporal.

Cuanto menos eficiente sea la simplificación más se parecerá la imagen a la mostrada en el apartado anterior.

4.4.7 Errores producidos por el mapper probabilístico

Al utilizarse el mapper probabilístico para realizar transformaciones sobre imágenes, se introduce un cierto error en la imagen resultante de salida. Este error se debe a dos motivos fundamentales:

- El mapper no modifica la ubicación en el tiempo de los eventos, sino que en su lugar "absorbe" algunos eventos. Aunque los eventos absorbidos se encuentren homogéneamente distribuidos, la separación entre los eventos que si se emiten no sería homogénea, al haberse eliminado algunos.
- Al utilizar un generador aleatorio para determinar qué eventos se emiten y cuales no, para un número de muestras suficientemente grande, se puede suponer que el número de eventos de salida será el número de eventos de entrada multiplicado por la probabilidad. Sin embargo, para pocas muestras, la no homogeneidad de la función aleatoria introduce errores adicionales, al no eliminarse exactamente el número de eventos deseado.

4.4.7.1 Error por la discretización en la posición de los eventos

En esta categoría de errores se incluirían los errores en la homogeneidad de los eventos de salida, provocado sobre todo al dividirse en estos sistemas digitales el tiempo en slot que pueden ser ocupados por el evento. Digamos que el instante de salida no puede ser cualquiera, sino que existirán unas ventanas de tiempo en las cuales puede emitirse el evento.

Sin embargo, en el caso del mapper probabilistico el error de discretización se hace más evidente, al quedar limitadas las ventanas de emisión de cada evento al instante en el cual se recibe por el puerto de entrada su evento "original". El error introducido en la posición de los eventos debido a esto es del orden del tiempo Tslice.

4.4.7.2 Error producido por el generador aleatorio

Debido al carácter aleatorio asociado a la probabilidad con que un evento será generado, incluso en las transformaciones simuladas utilizando MATLAB, donde se eliminan los errores debidos a elementos externos al algoritmo, se observan que las imágenes resultantes presentan un ruido añadido. Esto se debe a que al asociar una probabilidad inferior a 1 (255), habrá eventos de entrada que no generarán un evento de salida. Para periodos de integración largos, puede suponerse

que la distribución de estos eventos absorbidos por el mapper será homogénea, pero para periodos de integración pequeños, la distribución de estos elementos eliminados puede ser irregular y producir valores de luminosidad que varían ligeramente alrededor del valor deseado.

Para calcular el error introducido por el generador aleatorio, se supone que la separación del los eventos de entrada es homogéneo para un determinado píxel. Lo cual no ocurre con los métodos de generación AER descritos anteriormente, ya que introducían errores adicionales debido a la discretización en slots, etc. El error producido por el generador aleatorio se sumaría al del generador, y al de los demás elementos de la cadena AER. Sin embargo, para estudiar el error producido por el generador aleatorio de forma independiente a los demás, se considera que los eventos vienen homogéneamente distribuidos (caso ideal).

Si se considera un único píxel, con un valor determinado V, con una probabilidad de emitirse P, el valor a la salida en el caso ideal sería V*P. Si ajustamos el tiempo de integración T de forma que coincida con el tiempo de frame, en el receptor se recibirán V*P eventos en el caso ideal. Para medir cuánto se desvía el valor obtenido en la realidad del caso ideal, se utiliza la medida estadística de la varianza, dada por la siguiente fórmula:

$$\sigma^2 = \frac{\sum (x_i - \overline{x})^2}{n} \tag{4.23}$$

Siendo n el número de eventos de entrada V, Xi el número de eventos obtenidos al aplicar la probabilidad P, V veces, y $\overline{x} = V * P$.

El error al reconstruir un píxel se expresa como la diferencia entre el valor reconstruido (x_i) y el valor que se esperaba reconstruir:

$$E = |x_i - \overline{x}| \tag{4.24}$$

El error relativo correspondería a:

$$E_{rel} = \frac{E}{\overline{x}} = \frac{|x_i - \overline{x}|}{\overline{x}} = \left| \frac{x_i}{\overline{x}} - 1 \right| \tag{4.25}$$

Si en lugar del valor absoluto, tomamos en consideración el signo del error, al representar varias simulaciones superpuestas podemos tener además idea de los valores máximos que tiene la desviación en ambos sentidos. Por lo tanto podemos representar:

$$Desv_{rel} = \frac{x_i}{\bar{x}} \tag{4.26}$$

En la ilustración 4.35, podemos ver la gráfica superpuesta de varias simulaciones de los valores reconstruidos normalizados para un valor de entrada Vin=255, con una probabilidad de emitir el evento del 50%. Por lo tanto el valor teórico de salida Vout sería un promedio de aproximadamente 128 eventos recibidos (255 eventos de entrada iniciales, de los cuales sólo el 50% se emitirían). En el eje de las X se indican diferentes tiempos de integración, expresados en número de frames, comenzando por un frame hasta 30 frames. En el eje de las Y se representa el valor reconstruido dividido por el valor esperado, para tener una indicación del error relativo que se comete.

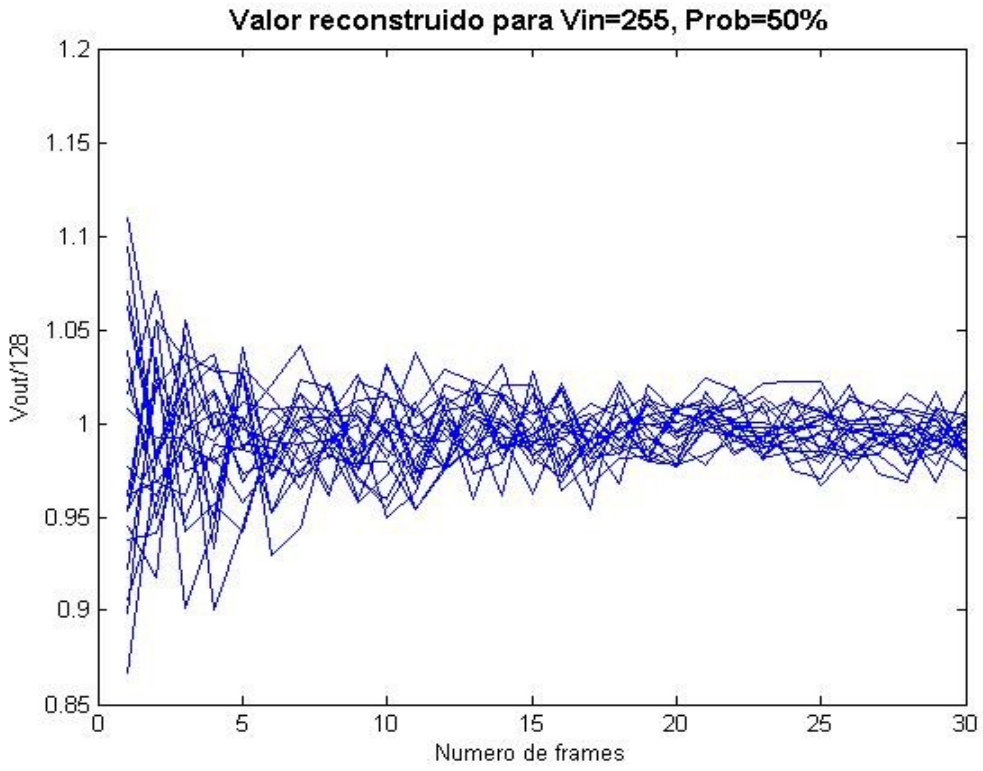

Ilustración 4.35: Valor reconstruido normalizado

Como se puede observar en la gráfica, el error en la reconstrucción es máximo cuando se integra un único frame, pero converge conforme se incrementa el número de frames que se promedian. Se representa Vout normalizado para tener una medida relativa del error, independientemente del valor de entrada. En el caso ideal el valor de salida normalizado sería por tanto 1. En el caso más desfavorable, cuando se integra durante un único frame, la desviación que se observa es inferior al 15%. Cuando el tiempo de integración se incrementa, este error disminuye. Para un tiempo T mayor a 5 frames vemos como el error que se observa es inferior al 5%.

Análisis de la varianza en función de la probabilidad

En la ilustración 4.36 se representa el promedio de la varianza respecto a la probabilidad de emitir el píxel, para analizar la dependencia entre el error que se comete en función de la probabilidad. Cuando la probabilidad es 0 (no se emite ningún evento) o 1 (se emiten todos los eventos), el generador aleatorio no introduce ningún error (la condición de emisión no depende del valor de la probabilidad). Sin embargo, para cualquier otro valor de probabilidad, la emisión o no de un evento depende un valor aleatorio, por lo que si habrá errores. Podemos observar como el error máximo se comete cuando la probabilidad es del 50%, y es además simétrica respecto a este 50%.

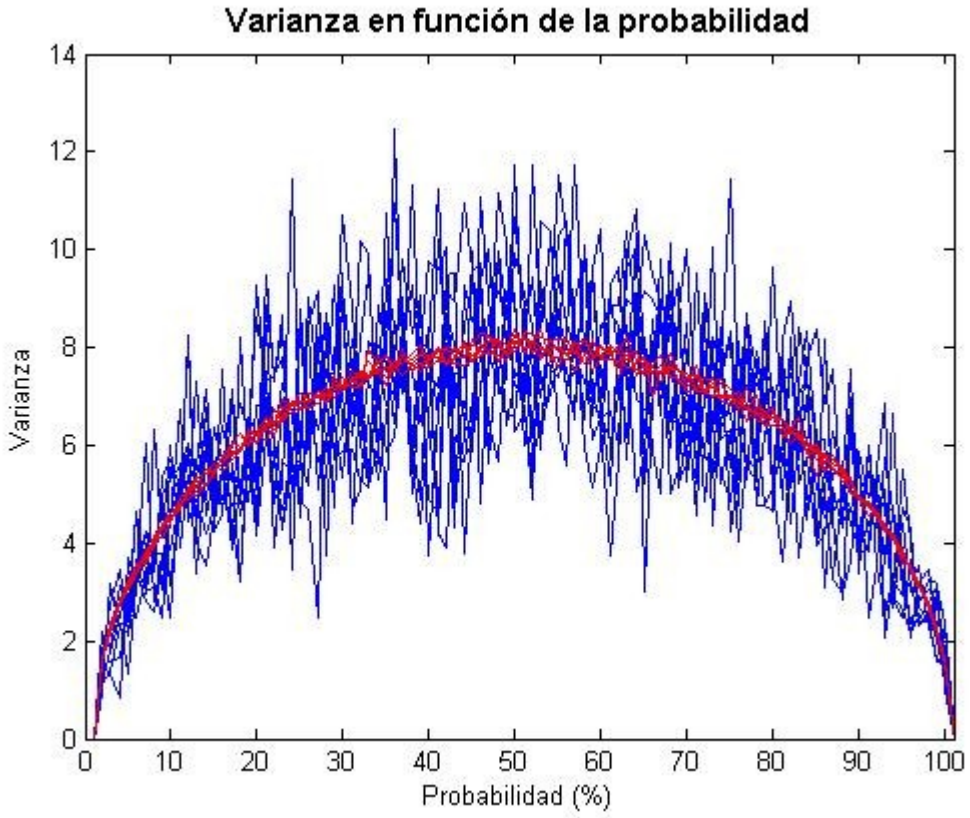

Ilustración 4.36: Estimación error en funcion de la probabilidad

En azul se representan varias simulaciones cuando el tiempo de integración es de 10 frames. En rojo se representa la varianza, pero en este caso cuando el tiempo de integración es de 1000 frames. En el eje de las Y se refleja la varianza y en las X la probabilidad, desde 0% (no se emite ningún píxel) hasta el 100% (se emiten todos).

Cuando el tiempo de integración es pequeño, el valor que toma la varianza en distintas simulaciones presenta mayores variaciones. El valor medio de que toma la varianza, se puede observar con mayor claridad cuando se incrementa el tiempo de integración. En la gráfica roja se observa con mayor precisión el valor medio que toma la varianza en función de distintas probabilidades.

Por último, vamos a ver como influye el valor del píxel sobre el error al reconstruir su valor. Para ello, se ha representado en la ilustración 4.37en azul los valores normalizados, para poder comparar el porcentaje de variación al reconstruirlo. La probabilidad de emisión se ha fijado en el 50%, para obtener el error en el caso más desfavorable. En rojo se representan los mismo valores, cuando el tiempo de integración se corresponde con 20 frames.

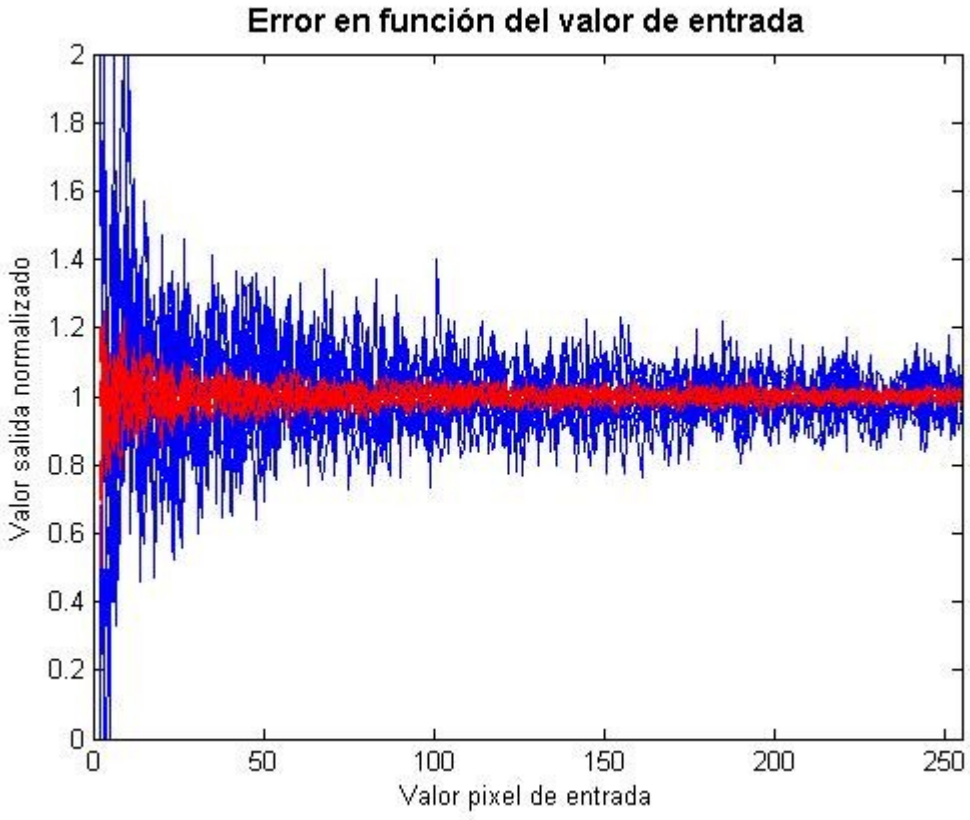

Ilustración 4.37: Error en función del valor de entrada

5. Métodos de conversión AER-Frame

El flujo AER que representan una imagen cambiante no pueden ser visualizados directamente, es necesario que un receptor conectado al bus AER, convierta las secuencias de eventos transmitidas a través del bus y reconstruya la imagen con los diferentes niveles de gris de cada píxel. Es evidente que el ojo capta cambios de luminosidad por lo que al final con el bus AER tendremos que tener algún elemento que produzca dichos cambios, en definitiva una pantalla. Aunque en ocasiones puede que el procesado de imagen no tenga como finalidad la observación, si no la tomas de decisiones; por ejemplo en el seguimiento de blancos la salida del procesado de imagen son las coordenadas del objeto a seguir (en el caso AER sería que los pixels activos se corresponden con la posición del blanco). Otro ejemplo sería el de reconocimiento, en el que un sólo evento indica si se encuentra o no el objeto a reconocer en escena. En ambos ejemplos no se reconstruye la imagen tras el procesado, sólo se emiten eventos con otro tipo de información. Sin embargo, en numerosas ocasiones es preciso visualizar las imágenes tratadas, sobre todo cuando se están depurando filtros y se quiere comprobar su eficacia.

Como se ha indicado anteriormente el mecanismo de reconstrucción AER basado en la integración de eventos es equivalente al proceso que se realiza en los sistemas de fotografía y vídeo convencionales [LINARES01]. Incluso para una retina derivativa este mecanismo muestra resultados admisibles cuando se produce movimiento en las imágenes.

En este trabajo se generaliza la reconstrucción de imágenes AER sin necesidad de digitalizar directamente su flujo mediante la conversión a frames durante un periodo de integración. En este caso se trata de reconstruir la imagen tal cual nos llega, obteniéndose de forma continua e independiente la frecuencia instantánea para cada pixel, esto es equivalente a indicar que cada pixel tendría una frecuencia de muestreo diferente a las de los demás y cambiante en el tiempo, ya que depende de cuando ocurran los eventos de cada uno. Pero el cálculo de la frecuencia instantánea en principio es imposible, ya que en el caso de PFM dicha frecuencia no se puede calcular sin la presencia de pulsos, cuando estos no se dan sólo podemos decir que frecuencia es la que seguro que no se corresponde con la instantánea, pero hasta que no aparezca el siguiente pulso no se tiene certeza de cuál ha sido. Siguiendo este mecanismo, en el presente capítulo se propone procesos de olvido para representar la información visual, es decir, el tiempo entre eventos representa el valor del píxel, pero cuando no llegan eventos dicho valor se debe ir corrigiendo, puesto que ya no se cumple el anterior tiempo entre eventos y el valor del píxel es otro. En el caso de que el tiempo

entre eventos fuese cada vez menor el olvido no entraría en juego, pues se obtiene una nueva frecuencia instantánea, sólo tiene sentido cuando no aparecen nuevos eventos y aumenta el tiempo entre ellos.

El número de eventos correspondientes a un píxel tiene una relación directa con el nivel de luminosidad de dicho píxel. Un píxel muy luminoso emitirá más eventos que un píxel oscuro. Esta relación no tiene por que ser lineal, como vimos en el capítulo 3. Dependiendo de la naturaleza de la imagen que se envía por un canal AER, esta presenta distintas características. Una imagen de niveles de gris, genera mucho tráfico en el bus, independientemente del movimiento de la escena. La cantidad de eventos en el bus es proporcional a la luz presente en el ambiente. Por otro lado, una retina derivativa generaría muy pocos eventos cuando la escena que se está observando permanece inmóvil. Al solo enviarse la información de cuanto cambia la cantidad de luz recibida por un píxel, el número de eventos es reducido salvo cuando la retina ve movimiento. En este caso además pueden aparecer eventos negativos. Al aparecer eventos negativos la naturaleza del bus AER cambia. El valor reconstruido en este caso es función de la frecuencia de eventos, pero los eventos negativos anularían a los positivos, por lo que el valor real depende de la cantidad de eventos de ambos signos.

Para realizar dicha reconstrucción se han desarrollado varios dispositivos que permiten diferentes aproximaciones para conseguir este fin. La tarjeta PCI-AER (Apéndice 11.2), permite recibir un gran número de eventos a gran velocidad junto con una información de tiempo asociada a cada evento. Mediante software, puede tratarse esta información para reconstruir y mostrar la imagen reconstruida. El datalogger [RIVAS01][RIVAS02] tiene un funcionamiento similar pero la cantidad y la velocidad de eventos leídos esta limitada por el bus USB. Este dispositivo es útil para capturar los eventos y posteriormente procesarlos offline. Por último, un dispositivo framegrabber consiste en un hardware que realiza el proceso de conversión, eliminando la necesidad de un procesado software de los eventos. Teniendo en cuenta que para imágenes de 64x64 bits, la cantidad de eventos en el bus es muy elevada, es interesante poder procesar todos estos eventos en hardware, dejando al software únicamente la tarea de mostrar en pantalla la imagen ya convertida.

Independientemente del dispositivo hardware empleado, la tarea de la reconstrucción AER puede abordarse desde dos puntos de vista. Si nos centramos únicamente en el bus AER con eventos positivos, el nivel de gris depende de la frecuencia con que aparece una dirección en el bus. Para calcular dicha frecuencia existen dos mecanismos. Como la frecuencia se define como la cantidad de eventos por unidad de tiempo, el primer método de reconstrucción descrito en este capítulo, el método del integrador, en contar la cantidad de eventos en un intervalo fijo de tiempo.

Por otra parte, la frecuencia también se define como la inversa del periodo, por lo que podemos calcular dicha frecuencia a partir de medir el periodo, o tiempo que transcurre entre dos ocurrencias de dicho evento. Siguiendo este procedimiento se describe más adelante el método de la frecuencia instantánea. Este método, requiere por ejemplo que la separación entre eventos sea constante mientras el valor de gris que representan también lo sea. Pequeñas fluctuaciones en la posición de los eventos en el tiempo introducidos por los métodos de generación se reconstruyen como ruido utilizando este mecanismo. Tampoco contempla qué ocurre si existen eventos negativos.

Para el método del integrador, la presencia de eventos negativos no presenta ningún problema. De igual forma que la llegada de un evento positivo hace que se incremente la cuenta de eventos, la llegada de un evento negativo provoca que dicho valor se decremente. Dependiendo de la naturaleza de los eventos, su distribución, y qué estemos representando, un mecanismo puede ser más adecuado que otro.

Para comparar y evaluar ambos métodos, se ha implementado en VHDL sobre la placa USB-AER dos dispositivos framegrabber distintos. Cada uno siguiendo una de estas alternativas. Para el método del integrador el framegrabber entrega directamente la información de luminosidad de los píxels, mientras que en la frecuencia instantánea, mide el periodo entre eventos y envía dicha información del periodo. Debido a lo costoso de implementar en una Spartan II divisores para calcular la inversa, se ha delegado esta función al software, que únicamente debería calcular la inversa del valor recibido y mostrarlo en pantalla.

5.1 Método del integrador

5.1.1 Introducción

Independientemente del significado de la información enviada, para la reconstrucción de la imagen se necesita convertir la frecuencia con que un píxel determinado aparece en el bus en la magnitud correspondiente. Si suponemos un único píxel aislado, podemos suponer que no ocurrirán colisiones entre diversos píxels a la hora de emitir eventos, y cuyo valor permanece constante con el tiempo, tenemos que los eventos de ese píxel se encontrarían homogéneamente distribuidos en el tiempo de forma equidistante, de tal forma que la frecuencia para ese píxel puede expresarse:

$$Freq\,P_{ij} = N * F_{MAX} \tag{5.1}$$

Siendo F_{MAX} la frecuencia máxima con la que aparecerá el píxel en su valor máximo de frecuencia, y N el valor del píxel perteneciente al intervalo [0,1].

Para la reconstrucción de la imagen es necesario calcular la frecuencia independiente para cada píxel. Esto puede conseguirse de dos formas diferentes. La primera consistiría en calcular el numero de eventos que ocurren para cada píxel entre dos instantes de tiempo determinado, que llamaremos tiempo de frame. De esta forma estaríamos dividiendo una secuencia AER en una sucesión de imágenes estáticas que se van presentando secuencialmente, de forma similar a la transmisión de imágenes de televisión.

$$Freq\,P_{ij} = \frac{NumEventos\,P_{ij}}{T_{FRAME}} \tag{5.2}$$

Sustituyendo en la ecuación anterior el valor de la frecuencia, y despejando el valor del píxel indicado por N, tenemos:

$$N = \frac{Freq\,P_{ij}}{F_{MAX}} = \frac{NumEventos\,P_{ij}}{T_{FRAME} * F_{MAX}} = \frac{NumEventos\,P_{ij}}{NumEventos_{MAX}} \tag{5.3}$$

Este mecanismo de conversión tiene la ventaja de asemejarse a los mecanismos tradicionales de transmisión de imágenes. Sin embargo, es necesaria la elección de un tiempo de frame adecuado en función de las características de la imagen. Hay que tener en cuenta que se supone que el valor de un píxel permanece constante durante todo el tiempo de un frame. Por el teorema del muestreo, cambios cuya frecuencia sea superior a la mitad de la frecuencia de frame no podrán reconstruirse correctamente. En su lugar, el valor obtenido será un valor intermedio entre los valores máximo y mínimo. La elección de un tiempo de frame demasiado amplio provoca que la imagen se actualice muy lentamente, y los movimientos más rápidos aparecerían como "movidos" . Si por el contrario se elige un tiempo de frame demasiado pequeño, se reciben pocos eventos por cada píxel, lo que provoca imágenes poco contrastadas y con poca luminosidad.

Matemáticamente puede modificarse el contraste de la imagen multiplicando el valor de cada píxel por una constante K. De manera que mientras el producto T_{FRAME}*K se mantenga constante, el valor calculado para el píxel no variará. La interpretación física sería que si integramos durante por ejemplo la mitad del tiempo de frame, recibiremos la mitad de eventos para cada píxel, por lo que habrá que multiplicar por dos la cuenta de eventos para obtener el valor deseado de luminosidad. Para un píxel de frecuencia F, en un tiempo T obtendríamos T*F eventos. Si reducimos la duración del periodo siendo T'=T/K, el numero de eventos que obtenemos sería en este caso T'*F=T*F/K.

De esta forma, al estar reduciendo el tiempo de frame, o tiempo de muestreo K veces, estaríamos aumentando la respuesta en frecuencia del sistema en la misma proporción. Sin embargo también estaríamos introduciendo ruido en la reconstrucción. Al multiplicar por un factor K para corregir el oscurecimiento de la imagen, dado que el numero de eventos que ocurren entre dos instantes de tiempo es un valor entero, el numero de niveles representable se reduciría en ese mismo factor K. Por otra parte, sobre todo para píxels cuya frecuencia sea muy baja, del orden de pocas veces la frecuencia de frame original, en el tiempo que dura un frame aparecerían pocos eventos de estos píxels. Al dividir un frame en varios subframes, puede ocurrir que para algunos subframes aparezca un eventos, q al multiplicarse por K se aumenta indebidamente su luminosidad, mientras que en otros frames, al no aparecer ningún evento, se supondría sin actividad. El efecto observado al integrar para un píxel fijo poco brillante, sería el mismo que si se observara un un píxel de mayor luminosidad, que se enciende y apaga.

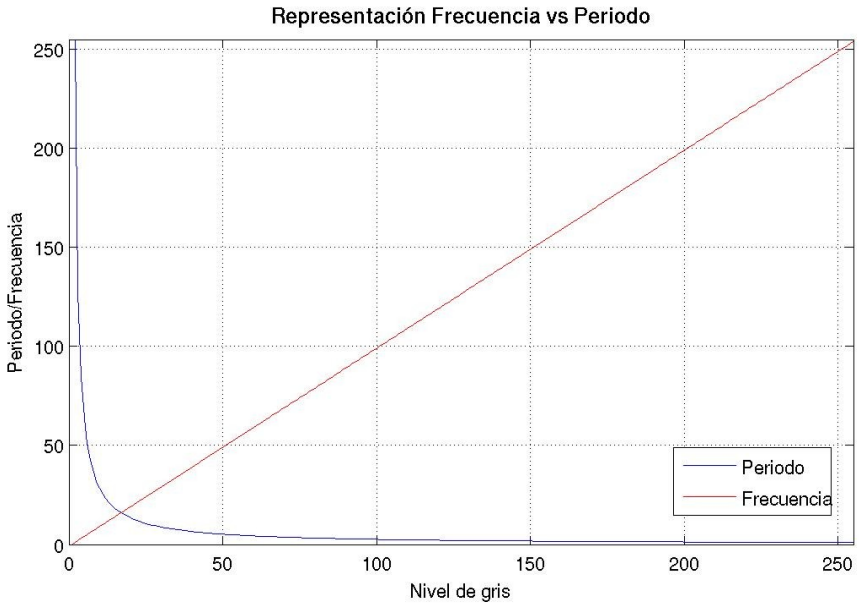

Ilustración 5.1: Frecuencia frente al periodo

El tiempo de frame adecuado viene limitado fundamentalmente por la cantidad de eventos por segundo que pueden emitir y recibir los elementos involucrados. El tiempo de frame máximo, cuando todos los píxeles emiten con frecuencia máxima, por lo que la cantidad de eventos será máxima sería:

$$T_{FRAME\,MAX} = Num_{pixels} * N_{MAX} * T_{evento} = Num_{pixels} * Freq_{MAX} \tag{5.4}$$

5.1.2 El problema del olvido en el receptor integrador

En el proceso de integrar por frames, al iniciar un nuevo frame, se cuentan el número de ocurrencias que ocurren para cada píxel y de esta manera obtener el nivel de gris correspondiente. Esta cuenta de eventos comienza de nuevo desde cero con el nuevo frame. Si no fuese así la suma de eventos iría creciendo indefinidamente, al irse acumulando los nuevos eventos a todos los recibidos con anterioridad. Para evitar esto es necesario un sistema de "olvido". El mecanismo más sencillo, partiendo de la idea de dividir el tiempo en frames, es inicializar todos los contadores al comienzo de cada frame. Una vez ha transcurrido el tiempo de frame, que sería equivalente al tiempo de integración, se da por válido el frame y se comienza desde el principio.

Sin embargo, la división en frames es algo externo al AER, que limita sus posibilidades. Por ejemplo, un movimiento más rápido de el tiempo de frame determinado no podría ser detectado por el sistema, o aparecería "movido" al ocupar distintas posiciones a lo largo del periodo de integración. Si la fuente AER fuese una imagen sintética a partir de vídeo capturado por una cámara de vídeo, la dinámica de la secuencia AER viene fijada por las limitaciones de los sistemas de vídeo tradicionales basados en frames. El funcionamiento de una retina AER es muy distinto que trata la sucesión de imágenes como un continuo y no como una sucesión de imágenes estáticas. Los píxels muy luminosos producirán una frecuencia de eventos altas. Teniendo en cuenta lo tratado en el capitulo 3, de manera teórica podemos predecir que para un determinado píxel, el sistema será capaz de detectar cambios en dicho píxel para frecuencias inferiores a la mitad de la frecuencia del VCO para el nivel de luminosidad actual.

La reconstrucción mediante frames no es adecuada para este tipo de fuentes AER. Ajustar el tiempo de frame no corrige el problema. Cuanto mayor es el tiempo de frame, menor es la respuesta temporal del sistema (se ganaría tolerancia al ruido, al promediar durante más tiempo, pero se pierde resolución temporal). Disminuir el tiempo de frame permitiría llegar a una solución de compromiso entre dinamismo y tolerancia a errores, sin embargo no resuelve el problema. Para poder detectar el movimiento más rápido detectable por el emisor, el tiempo de integración debe ser del orden de la mitad de la frecuencia más rápida posible, lo cual es demasiado pequeño para el resto de los píxel (muchas de las frames no recibirían eventos relativos a píxels poco brillantes).

Una posible solución es hacer el proceso de captura de frames no destructivo, y en lugar de que todos los integradores se reinicien al principio de cada frame, el proceso de olvido sea progresivo en el tiempo, eliminando el concepto de frame del proceso de integración. El estado de los integradores (captura de la imagen) puede comprobarse de manera independiente a la integración sin modificar el sistema. La dinámica del sistema viene entonces determinada por la función de olvido de cada píxel.

Este procedimiento sin embargo presenta varios problemas. La función de "carga" del integrador, es lineal al incrementarse su contenido en uno cada evento. Un evento muy brillante que durase poco tiempo no daría tiempo al integrador a acumular un número suficiente de eventos para hacerlo significativo. Por otra parte, el implementar en hardware el proceso de olvido progresivo presenta algunos problemas. Al hacerlo de manera independiente a la captura de las frames, no puede utilizarse el mismo acceso de lectura para reiniciarlo, sino que paulatinamente debe ir decrementando el valor de todos los píxels. Podemos recorrer periódicamente toda la imagen e ir actualizando sus valores. La tasa a la que habría que recorrer la memoria es varias veces superior al tiempo de integración del que partíamos (la descarga era instantánea mientras que ahora es progresiva). La función de descarga no debe ser lineal. Una función lineal premia a los valores muy altos al perder menor magnitud en términos relativos comparados con los píxels bajos. Si $K_{descarga}$ es pequeña, el sistema se saturaría al crecer los integradores indefinidamente. Si $K_{descarga}$ es grande, los píxels poco luminosos pierden toda su carga a los pocos ciclos.

Otra solución sería utilizar una matriz de olvido de forma que no todos los píxels tuvieran que cumplir la misma función de olvido, sino que pudiese seleccionarse para distintas zonas de la imagen, en función de sus características, asignarle un código, que indique cual de entre todos los posibles olvidos hay que aplicar. Utilizando 2 o 3 bits únicamente permitirían distinguir 4 o 8 clases de olvido distintas.

5.2 Método de la frecuencia instantánea

5.2.1 Introducción

Un segundo método para la reconstrucción consistiría en calcular el valor de la frecuencia midiendo el periodo que transcurre entre dos eventos del mismo píxel en lugar de integrar el número de eventos que ocurren entre dos instantes de tiempo. De esta forma se necesita únicamente dos eventos de un mismo píxel para conocer su frecuencia. Tenemos sin embargo que para píxels muy luminosos, el número de veces que podemos obtener esta información de frecuencia por unidad de tiempo es muy superior que para frecuencias bajas. Por tanto, para valores altos, la velocidad con que el valor del píxel puede cambiar es mayor que para píxels poco luminosos. Cuanto más luminosa sea una imagen, más rápido podrán ser sus cambios de luminosidad (movimiento). Esta limitación no se debe al método de reconstrucción sino a las características intrínsecas del bus AER, o más concrétamente, a la modulación en frecuencia que se utiliza.

Realizando la reconstrucción mediante el cálculo de la frecuencia intermedia presenta algunas ventajas e inconvenientes. La principal ventaja es su alta respuesta dinámica a los cambios en el espaciado de los eventos. Si un píxel poco brillante recibe durante un breve periodo de tiempo mucha intensidad de luz, el píxel alcanzará su valor máximo transcurridos tan solo dos eventos, sin que la luminosidad que tuviera en un instante anterior aporte nada a su valor. En la reconstrucción mediante integradores, el píxel va ganando su valor progresivamente.

Por otra parte, no existe ninguna división entre frames, y a priori no es necesario tener en cuenta el "olvido" propio de la integración, dado que cada nuevo evento determina por si mismo el valor del píxel calculando la separación con el evento anterior. No existe "acumulación" de eventos. El sistema sería sin embargo más rápido a los cambios que impliquen un aumento en la frecuencia de eventos. Cuando un píxel pierde luminosidad, los eventos se van distanciando. Al no actualizarse el valor de un píxel hasta que no llega un evento relativo a ese píxel, al incrementar la separación entre eventos, si este espaciado no se produce de manera progresiva, el píxel mantendrá durante todo este periodo un valor erróneo muy elevado.

En los sistemas biológicos se observa un comportamiento parecido. Cuando una célula

óptica recibe mucha iluminación de golpe y después desaparece, similar a un flash, los receptores responden rápidamente a la iluminación, mientras que tardan en ir apagándose.

Para minimizar este problema, se puede calcular la separación entre el momento en el que se está muestreando la señal y el último evento, si esta separación es mayor a la del último evento con el anterior (si el nivel de gris estimado si el siguiente evento llegara justo en ese momento, caso más optimista), quedarse con el estimado, en caso contrario el valor del píxel. De esta forma, para eventos con frecuencias en descenso, conseguimos una atenuación progresiva. Sin embargo, seguimos sujetos a las limitaciones impuestas por el AER. El píxel no se vería apagarse en el momento que realmente lo hace, sino irse atenuando en lo que dura el periodo T del valor de gris final.

5.2.2 Implementación

El valor de luminosidad de un píxel particular viene determinado por la frecuencia de los eventos correspondientes a ese píxel. Para calcular esta frecuencia existen dos aproximaciones. Calcular la frecuencia media para cada píxel en un intervalo de tiempo determinado, contando el número de ocurrencias de cada evento durante el tiempo de frame, de esta forma:

$$F_{media} = \frac{Num\_eventos}{T_{frame}} \tag{5.5}$$

Otra aproximación consiste en calcular la frecuencia instantánea asociada a cada píxel calculando el periodo o separación entre dos eventos y calculando su inversa:

$$F_{instantanea} = \frac{1}{T_{instantáneo}} \tag{5.6}$$

Para implementar el conversor AER-Frames por el método de la frecuencia instantánea, habrá que medir el periodo que transcurre entre dos eventos correspondientes a un mismo píxel. Para ello, se utiliza una memoria RAM donde se almacena una tabla con los periodos medidos, en lugar del número de eventos recibidos. Para calcular la frecuencia de un píxel, hay que almacenar el instante de tiempo en que se recibió el último evento correspondiente a ese píxel. Al recibir un nuevo evento, se puede calcular la separación con el anterior restando la información de tiempo del evento recién recibido con la información de tiempo de la ultima ocurrencia almacenada. El periodo calculado de esta manera debe almacenarse en otra tabla en RAM con el fin de poder ser recuperado desde el ordenador en cualquier momento para mostrar la imagen reconstruida. Por lo tanto se necesitan dos tablas en RAM, una que almacene el último periodo calculado para cada evento, y otra con la información de tiempo del último evento recibido de cada píxel para poder calcular el nuevo periodo cuando reciba un nuevo evento.

Como reloj utilizado para medir la separación entre eventos, se utiliza un contador de 32 bits que se incrementa automáticamente con cada ciclo de reloj. El valor de este contador se utiliza como marca de tiempo, que se restarán para calcular el número de tics de reloj transcurridos entre dos eventos. Sabiendo la frecuencia de reloj del sistema, puede convertirse la información de tiempo expresada en tics de reloj a nanosegundos.

El funcionamiento del integrador sería el siguiente:

1. *Recibimos un evento. Se accede a la tabla de time-stamps indexando por el número de evento.*
2. *Se resta el valor del reloj actual con el valor almacenado para calcular el periodo.*
3. *Se almacena el valor del periodo calculado en la tabla de periodos.*
4. *Se almacena el instante actual como nueva información de tiempo para el evento recibido en la tabla de time-stamps.*

El diagrama de flujo correspondiente sería el mostrado en la ilustración 5.2:

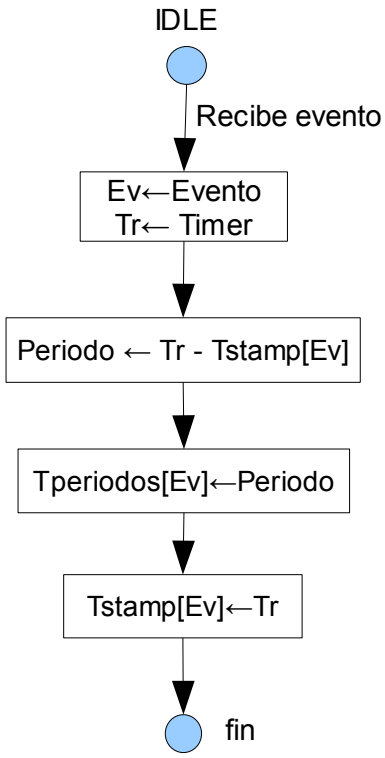

Ilustración 5.2: Funcionamiento framegrabber pseudoinstantáneo

Cuando el software solicita un nuevo frame para mostrar en pantalla, se envía por el USB el contenido de la tabla de periodos completos. Dado la complejidad hardware de calcular una división

para obtener la inversa del periodo, y por tanto la frecuencia, de cada píxel, a diferencia del integrador por frames que almacenaba la información de frecuencias propiamente dicha, el integrador instantáneo almacena la información del periodo, y será tarea del software el calcular la inversa, y calcular la equivalencia entre periodo-frecuencia y valor de gris.

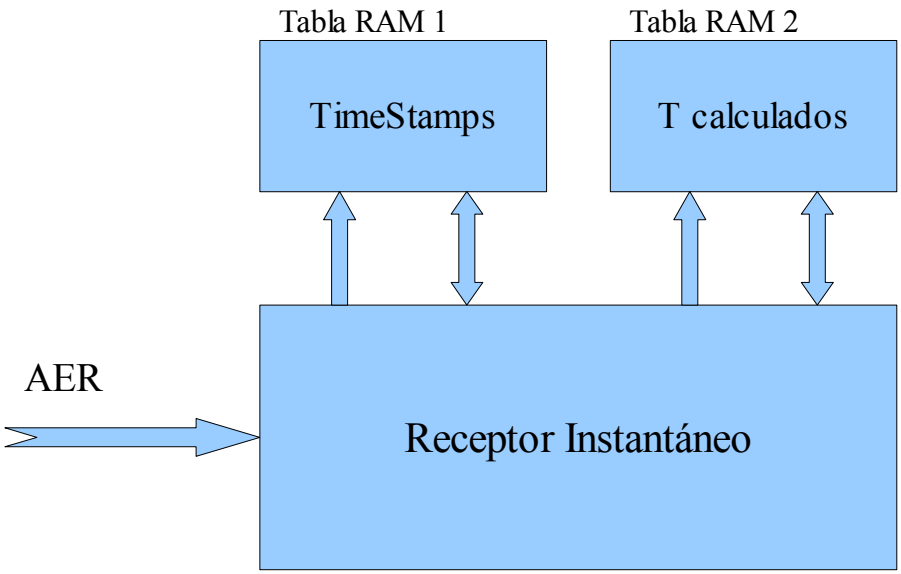

Ilustración 5.3: Diagrama bloques receptor instantáneo

Debido a que el tamaño de las tablas almacenadas en RAM es mayor que el equivalente basado en un integrador, el número de celdas de RAM internas de la FPGA es insuficiente para contener estas tablas, por lo que es necesario utilizar los chips de SRAM externos presentes en la placa USB-AER. El ancho del bus de datos de la memoria externa es de 32 bits, al igual que el contador-timer, por lo que es posible escribir un dato completo en un único acceso. Sin embargo, por cada evento recibido, hay que modificar el contenido de dos tablas, por lo que el número de accesos totales por cada evento recibido será un acceso de lectura (time-stamp antiguo) y dos accesos de escritura, para actualizar el nuevo time-stamp y el nuevo periodo calculado para dicho píxel.

Al leer el contenido del receptor desde el PC, la información suministrada por el receptor se refiere al periodo entre dos eventos, en lugar de la frecuencia propiamente dicha, por lo que será función del software el calcular la inversa. Dado que el dispositivo emisor puede ser de diversa naturaleza, la equivalencia entre la "frecuencia base" y el nivel de gris más bajo puede diferir en función de la velocidad del emisor, por lo que a priori, para un emisor desconocido, no puede establecerse cual será la frecuencia máxima o mínima con la que aparecerá un evento representando

los niveles máximos y mínimos de luminosidad. En el receptor integrador, este ajuste se realiza escogiendo un tiempo de frame adecuado. Un tiempo de frame muy alto provocaría que los contadores se saturaran al valor máximo, de forma que la imagen reconstruida se asemejaría a una foto sobre-expuesta. Un periodo de integración demasiado bajo provocaría el efecto contrario, una imagen muy oscura o infraexpuesta. En el receptor instantáneo, no existe el concepto de frame, por lo que los valores leídos desde el PC serán independientes del tiempo transcurrido entre operaciones de lectura. En este caso, el ajuste correcto se realiza en el software, al calcular la inversa, aplicando un factor multiplicador de escala. Dicho de otra forma:

$$Valor_{pixel} = \frac{Valor_{MAX} * T_{BASE}}{T_{medido}} \tag{5.7}$$

Siendo $Valor_{MAX}$ el valor máximo que puede tomar el píxel (normalmente 255), y T_{BASE} el periodo correspondiente a ese valor máximo. Representa la relación entre la frecuencia observada y valor del nivel gris. Será función de la velocidad del sistema.

5.2.3 El problema del olvido en el receptor instantáneo

A diferencia del Receptor-Integrador, donde era necesario establecer algún mecanismo de olvido, de manera que el valor de los contadores no creciese indefinidamente debido a la acumulación de eventos a lo largo de todo el tiempo. En el Receptor-Instantáneo no es necesario establecer ningún mecanismo de olvido que periódica o paulatinamente elimine parcial o totalmente el valor acumulado. Al calcular el valor del píxel utilizando únicamente los dos últimos eventos de dicho píxel, sin tener en cuenta la actividad de dicho píxel con anterioridad, el algoritmo no presenta ningún comportamiento acumulativo.

Observando el comportamiento del sistema se puede comprobar que cuando un píxel pasa de cualquier valor de gris al nivel 0, si el emisor asocia dicho nivel de gris a ausencia completa de eventos ($F_{PIXEL}=N_{GRIS}*F_{BASE}$), al no recibir más eventos relativos a ese píxel, el receptor conservaría el último valor calculado, sin considerar el tiempo que lleva sin recibir más eventos, cuyo valor calculado en el momento de la lectura sería muy elevado.

Este sería el caso extremo. Generalizando para niveles de gris intermedios, se puede deducir dado el funcionamiento del receptor, que un determinado nivel de gris queda congelado en el receptor hasta la recepción de un nuevo evento y por lo tanto el cálculo del nuevo valor (ilustración 5.4). Cuando un píxel evoluciona a un nivel de gris con más eventos, este mecanismo es correcto dado que mientras el píxel presenta un nivel de gris fijo, el espaciado entre eventos debe ser homogéneo, por lo que el valor almacenado coincidirá con el nuevo valor calculado manteniéndose el nivel reconstruido estable. Al evolucionar a un nivel de gris más luminoso, aumentará la frecuencia de eventos, disminuyendo el periodo entre eventos, actualizándose con mayor frecuencia aún el valor del píxel.

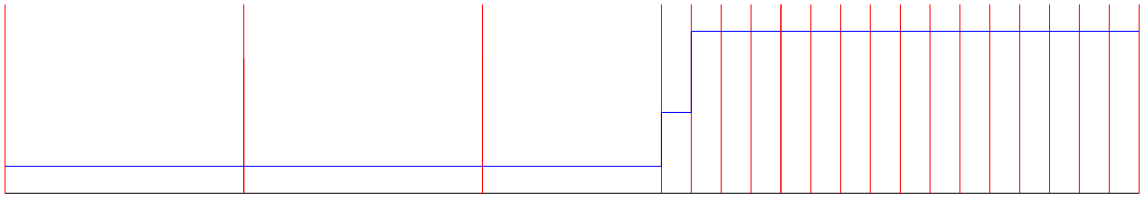

Ilustración 5.4: Receptor instantáneo(aumento luminosidad)

Sin embargo, al pasar de un nivel más brillante a uno muy oscuro (ilustración 5.5), el espaciado entre eventos aumenta, manteniéndose el periodo calculado hasta la recepción del siguiente evento, a pesar de que el tiempo ya transcurrido sin aparecer este evento implicaría que dicho píxel tiene un valor de gris inferior al calculado, que se presenta erróneo. El caso extremo sería cuando el nivel de gris es 0, que al no recibirse más eventos, mantendría un nivel de gris elevado, e incorrecto, dado la ausencia total de eventos.

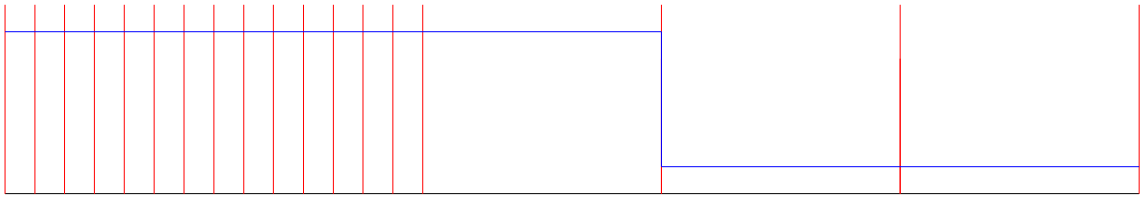

Ilustración 5.5: Receptor instantáneo (reducción luminosidad)

Una posible solución sería que el emisor mantuviera una frecuencia mínima de eventos incluso para el nivel de gris, o dicho de otra forma, añadiendo un offset a la frecuencia de eventos ($F_{PIXEL}=N_{GRIS}*F_{BASE}+Offset$) como se comenta en el capitulo 3 . Esto garantizaría una frecuencia mínima para todos los píxels, y por tanto, una frecuencia mínima para la actualización de todos los píxels. Además, al aumentar en conjunto la frecuencia para todos los niveles de gris, mejora también la respuesta dinámica del receptor en ambos sentidos (cambios de menor a mayor frecuencia y viceversa). Sin embargo, no resuelve el problema de que en un cambio de más a menos eventos, al aumentarse el periodo, el nivel antiguo se mantiene más brillante que su valor real, oscureciéndose bruscamente al llegar el nuevo evento. Este proceso de oscurecimiento podría interpolarse y convertirse en progresivo, comparándose con el tiempo que ha transcurrido sin recibirse ese evento con el periodo almacenado. En el caso de llevar más tiempo sin recibirse dicho evento del almacenado en la memoria, puede suponerse que el píxel se está oscureciendo, y considerar como periodo interpolado el tiempo que lleva sin recibirse dicho evento (ilustración 5.6).

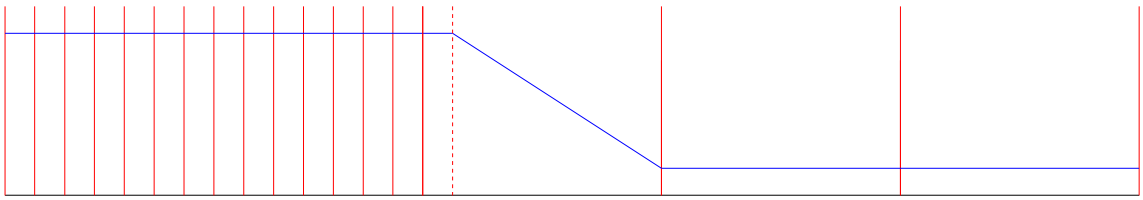

Ilustración 5.6: Receptor instantáneo (atenuación progresiva luminosidad)

Cuando el píxel cambia de un valor con pocos eventos a otro con mucha actividad, el nivel de gris crecería instantáneamente con la recepción del nuevo evento muy próximo al anterior. En este caso no existe interpolación posible dado que no puede predecirse la actividad que tendrá un píxel en un instante futuro.

El funcionamiento del receptor al solicitarle desde el PC una imagen sería el siguiente para cada píxel:

1. *Se captura el timer, con el fin de que sea el mismo para toda la imagen*
2. *Se calcula el periodo hasta el momento, restando el valor actual del timer con la marca de tiempo del último evento.*
3. *Se compara el valor recién calculado con el valor de periodo almacenado.*
4. *En el caso de que el valor interpolado sea mayor que el valor almacenado, se devolverá el valor interpolado. En caso contrario (lo usual), se devolverá el valor almacenado.*

El diagrama de flujo correspondiente se muestra en la ilustración 5.7:

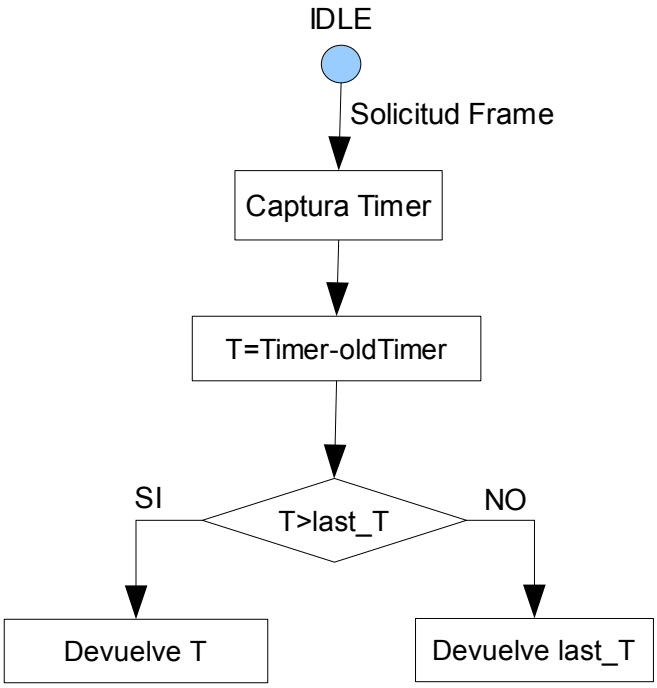

Ilustración 5.7: Diagrama flujo receptor pseudoinstantáneo

5.3 Resultados experimentales

Para comparar ambos mecanismos de reconstrucción, se plantea un escenario sencillo con únicamente dos dispositivos AER, un generador AER y un receptor. Se transmitirán dos imágenes distintas, una con la imagen patrón de la giralda, y posteriormente una imagen con la información del contorno de la misma (derivativa en el espacio), con el fin de comprobar el funcionamiento de ambos métodos de reconstrucción con dos imágenes de distintas características. Sabemos por [DELORME01][THORPE01][THORPE02] que es posible la caracterización de imágenes utilizando un número reducido de spikes. Con un 10% de los eventos iniciales sería suficiente para empezar a caracterizar la imágen.

Ilustración 5.8: Imagen original

Ilustración 5.9: Reconstrucción integrador

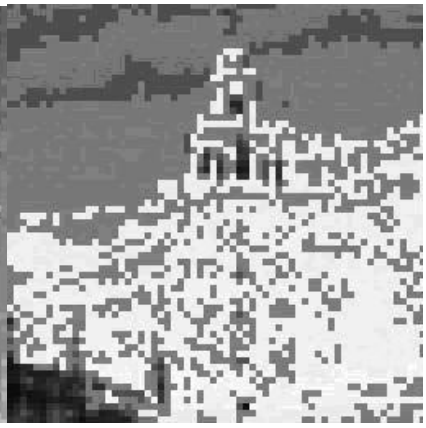

Ilustración 5.10: Reconstrucción Instantáneo

En las ilustraciones 5.9 y 5.10 se muestra el resultado de reconstruir con ambos métodos la imagen original mostrada en la ilustración 5.8. Vemos como la imagen reconstruida utilizando el integrador tiene menos ruido que la que se obtiene en la reconstrucción instantánea. El integrador integra el número de eventos que se recibe durante un periodo de tiempo (tiempo de frame), por lo que el valor de un píxel viene dado por múltiples eventos. Para el receptor instantáneo, sólo los dos últimos eventos recibidos determinan dicho valor, más concrétamente la separación en el tiempo entre ambos. Por lo tanto, este segundo método es más sensible a los errores en la distribución de los eventos. Para que la imagen se reconstruya correctamente sería necesario que el emisor mantuviera una distribución completamente homogénea de cada evento.

Sin embargo, la velocidad con la que un píxel puede cambiar de valor, en el caso del receptor con integrador, viene dado por el tiempo de frame, que suele ser del orden de decenas o centenas de milisegundos, y es el mismo para todos los puntos de la imagen, mientras que en el integrador instantáneo, cada píxel tiene su propia frecuencia de cambio, que viene dada por la frecuencia del evento correspondiente, pudiendo ser del orden de décimas de milisegundo, consiguiéndose una frecuencia de muestreo efectiva cientos de veces mayor.

A continuación repetimos el experimento con la imagen del contorno. Los resultados se muestran en las ilustraciones 5.11 a 5.13. En este tipo de imágenes, el número de eventos generados es mucho menor que en las imágenes en luminosidad. En este caso la información de los bordes es correctamente reconstruida por ambos métodos. Ajustando correctamente la frecuencia base de los eventos y su valor, podemos reconstruir la imagen en un tiempo de unos 500us, lo que daría un tiempo de refresco de 2000 frames por segundo.

Ilustración 5.11: Imagen original

Ilustración 5.12: Reconstrucción integrador

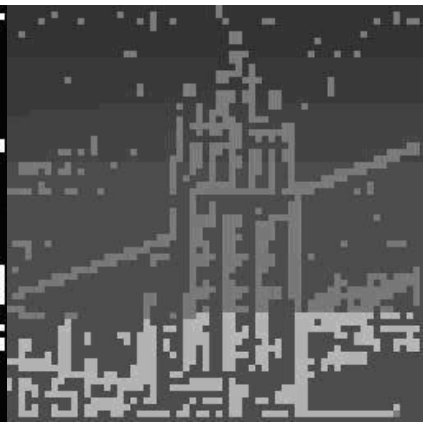

Ilustración 5.13: Reconstrucción instantáneo

5.4 Olvido

Cuando se quiere averiguar el valor que tienen una o más neuronas, hay que reconstruir un valor numérico, que en el caso ideal coincide exactamente con el valor que contiene el emisor, a partir de los eventos AER que dicho emisor vuelca en el bus. El valor de una neurona es función directa del número de ocurrencias de dicha neurona en el bus, por lo que para obtener un valor numérico es necesario un proceso de acumulación, integración o suma de eventos. Los eventos que aparecen con mayor frecuencia corresponderán a neuronas con mayor actividad, por lo que el valor reconstruido en este proceso de integración será mayor. El problema, es que necesitamos de alguna manera limitar la forma de que este valor acumulado crece para evitar que vaya creciendo continuamente y de manera indefinida, debido a que todos los eventos del bus tienen un carácter positivo, e irían continuamente incrementando el valor a reconstruir.

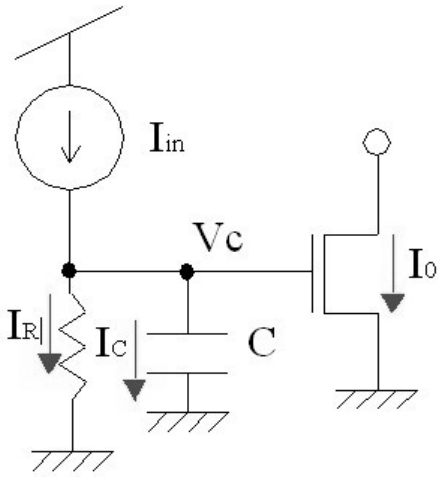

Ilustración 5.14: Integrador analógico

En un sistema analógico de reconstrucción, el integrador, basado en un condensador que va acumulando cierta carga cada vez que recibe un evento, el olvido vendría representado por una resistencia en paralelo por la que el condensador descarga parte de su carga acumulada.

En los sistemas digitales basados en FPGA, es necesario al reconstruir una señal AER limitar el valor de la integración de eventos. En diseños con pocas neuronas, cada neurona podría modelarse como un pequeño bloque con un contador, cuyo valor se incrementa con cada evento recibido, y que periódicamente se devalúa.

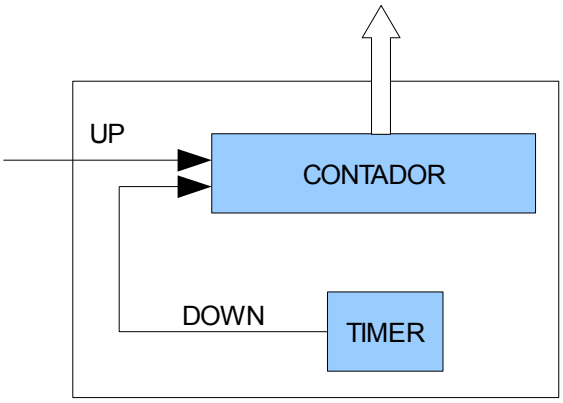

Ilustración 5.15: Integrador digital con olvido

Como veremos más adelante, el decrementar periódicamente una cantidad fija no resulta adecuado para la reconstrucción AER debido a que no soluciona el problema de que el valor de la integral crezca indefinidamente. Periódicamente hay que sustraerle al contador una porción del valor del contador, por lo que un contador up/down no sería suficiente, sino que haría falta un módulo multiplicador, que calculase el valor del contador antes de incrementarlo con el nuevo evento. Además, para compensar el olvido, el incremento por cada evento será superior a la unidad. Es necesaria una pequeña ALU en cada bloque en lugar de un simple contador.

Cuando el número de neuronas es elevado, como es el caso de las imágenes, cada una de las neuronas no puede implementarse en un bloque con un contador y una lógica de olvido relativamente compleja. El valor de los integradores tiene que almacenarse en bloques de RAM de la FPGA, y es necesario recurrir a un proceso que comparta el acceso a la memoria con el integrador y vaya devaluando los contadores. Alternativamente, junto al valor de cada integrador en RAM puede almacenarse una marca de tiempo con la antigüedad de dicho valor, de forma que cuando deba ser leído para mostrarse o para incrementar su valor como resultado de un nuevo evento, pueda en función de dicha marca de tiempo calcular el valor real tras un periodo de tiempo "olvidando".

Para ello se van a estudiar distintos mecanismos posibles de olvido, para tres valores posibles de una neurona 50 (rojo),100 (verde) y 200 (azul), como muestra de tres valores posibles dentro del rango de 0 a 255 dado.

5.4.1 Olvido total o por frames

Para limitar que los valores a reconstruir crezcan indefinidamente, en el integrador por frames, todo el proceso de olvido ocurre de manera instantánea una vez ha terminado el tiempo de frame y se ha leído dicho valor. Se toma como valor del píxel el total de la suma, y se inicializa a cero el contador para comenzar un nuevo frame:

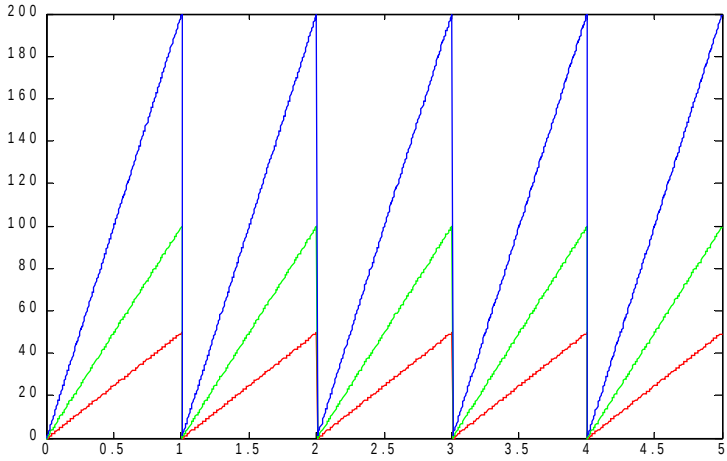

Ilustración 5.16: Olvido total o por frames

Como puede observarse en la ilustración 5.16, cada vez que se recibe un evento se incrementa el contador en uno. Al final del periodo, se lee el valor, y se reinicia a cero. El integrador tiene el valor correcto al final del periodo, pero el valor medio a lo largo del periodo va variando desde el valor cero inicial, hasta la mitad del valor final.

5.4.2 Olvido progresivo

Como mecanismo alternativo al olvido total que se describe anteriormente, es posible que cada neurona vaya perdiendo el valor que va integrando de manera progresiva, de forma similar a como lo va obteniendo. Cada vez que se recibe un evento, el valor del integrador se incrementa. Este valor que se va acumulando, va atenuándose de manera progresiva en el tiempo, de forma que el valor reconstruido correspondería al valor promedio del integrador, en la posición de equilibrio en la cual la cantidad de magnitud integrada que se repone que con los eventos recibidos coincida con la que se elimina debido al olvido.

Este método elimina la necesidad de definir un tiempo de frame o tiempo de integración, al repartir de manera homogénea en el tiempo la labor de olvidar. Sin embargo, como veremos a continuación, el valor real reconstruido oscila alrededor del valor final. La amplitud de esta oscilación, que se presenta como ruido al reconstruir una imagen, depende fundamentalmente de la frecuencia de los eventos de entrada y de la velocidad de olvido del integrador.

5.4.2.1 Olvido constante o lineal

Una función de olvido lineal define como mecanismo de olvido el substraer del valor integrado un valor constante k proporcional al tiempo. Si entendemos el proceso de olvido como una tarea que se produce a intervalos regulares de tiempo, en la cual substraemos una cantidad k constante cada vez.

$$V_{n+1} = V_n - k_{olvido} \tag{5.8}$$

$$V_{n+1} = V_n - k_{olvido} = [V_{n-1} - k_{olvido}] - k_{olvido} = V_{n-1} - 2 * k_{olvido} \tag{5.9}$$

$$V_n = V_{n-c} - c * k_{olvido} \tag{5.10}$$

Si en lugar de a intervalos regulares, se produce al cabo de un determinado tiempo t, a partir de la ecuación anterior podemos generalizar que el olvido al cabo de un tiempo t a partir del instante inicial t_0 sería el siguiente, siendo t_{base} el periodo entre dos actualizaciones para el cual se fijo k_{olvido} en la ecuación anterior:

$$V(t) = V(t_0) - k_{olvido} * \frac{t}{t_{base}} \tag{5.11}$$

Como vemos en las gráficas de las ilustraciones 5.17 y 5.18, el olvido lineal no nos resulta útil debido a que para frecuencias superiores a F, el olvido no es suficiente para contrarrestar a los eventos de entrada y el valor crece indefinidamente, mientras que para frecuencias inferiores a F, el olvido es superior a los eventos de entrada y el valor reconstruido sería próximo a cero.

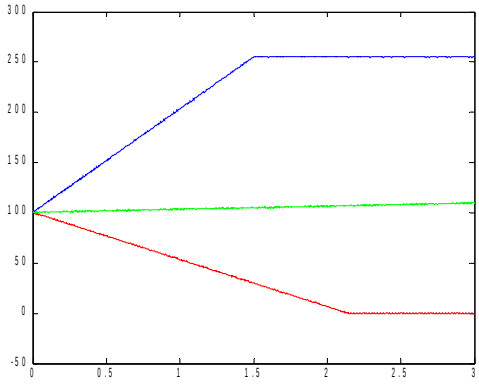

Ilustración 5.17: Olvido constante k=0.35

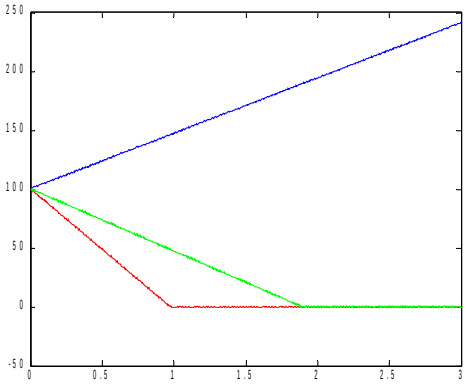

Ilustración 5.18: Olvido constante k=0.6

En la gráfica de la izquierda tenemos una constante de olvido de 0.35 mientras que en la gráfica de la derecha es de 0.6. Como vemos, el variar el valor de k solo modifica la pendiente de las rectas, pero no nos permite reconstruir adecuadamente el valor: se incrementaría para valores superiores a V' hasta alcanzar el valor máximo que puede tener una neurona que es de 255, o se incrementa para valores inferiores a V' hasta alcanzar el valor mínimo posible 0.

$$V' = k_{olvido} * V_{max} = k_{olvido} * \frac{t_{base}}{t_{slice}} \tag{5.12}$$

5.4.2.2 Olvido proporcional o exponencial

El olvido proporcional consiste en que el valor a substraer de la integración por unidad de tiempo no es un valor constante, sino que es proporcional al valor acumulado. Cada cierto tiempo, la función de olvido debe decrementar el valor de la integral, en una cantidad que se calcula multiplicando el valor que se está integrando por un factor de olvido, como se muestra en la siguiente expresión:

$$V_{n+1} = V_n - f_{olvido} * V_n = (1 - f_{olvido}) * V_n \tag{5.13}$$

Podemos generalizar el olvido, cuando en lugar de decrementarse periódicamente, se retrasa el decremento un número de veces el periodo anterior.

$$V_{n+1} = (1 - f_{olvido}) * V_n = (1 - f_{olvido}) * [(1 - f_{olvido}) * V_{n-1}] = (1 - f_{olvido})^2 * V_{n-1} \tag{5.14}$$

$$V_n = (1 - f_{olvido})^c * V_{n-c} \tag{5.15}$$

Generalizando para un tiempo t cualquiera, tenemos:

$$V(t) = (1 - f_{olvido})^{\frac{t}{t_{base}}} * V(t_0) \tag{5.16}$$

A partir de esta expresión, podemos actualizar el valor del integrador, sin la necesidad de ir calculando el olvido periódicamente. Esto elimina la necesidad en el integrador de un proceso que de manera concurrente a la integración vaya decrementando periódicamente la integral. De esta forma, en el instante previo a incrementar el valor de la integral, puede calcularse en función del tiempo transcurrido desde la última actualización, el valor de la integral teniendo en cuenta el olvido. Es útil de cara a la implementación en hardware del olvido.

Las dos gráficas siguientes representan la evolución del valor reconstruido para una determinada neurona con tres valores de excitación distintos, 50, 100 y 200. En la gráfica de la

izquierda el valor, en cada tiempo de slot, la neurona pierde el 1% de su valor, mientras que en la gráfica de la derecha, pierde el 20% de su valor. En ambas gráficas se ha ajustado el valor que añade al total cada evento, de manera que se compense adecuadamente la pérdida del olvido. La ganancia de cada evento viene determinada por la fórmula:

$$g_{neurona} = valor_{max} * f_{olvido} \tag{5.17}$$

Hay que multiplicar por el valor máximo (255) debido a que el tiempo de slot, que sería el periodo entre dos actualizaciones consecutivas lo hemos definido como el tiempo de frame dividido por el valor máximo posible:

$$T_{slot} = \frac{T_{frame}}{Valor_{max}} \tag{5.18}$$

Para la ilustración 5.20, la ganancia por cada evento es 20 veces superior a la primera para compensar el factor de olvido, que es veinte veces superior a la ilustración 5.19:

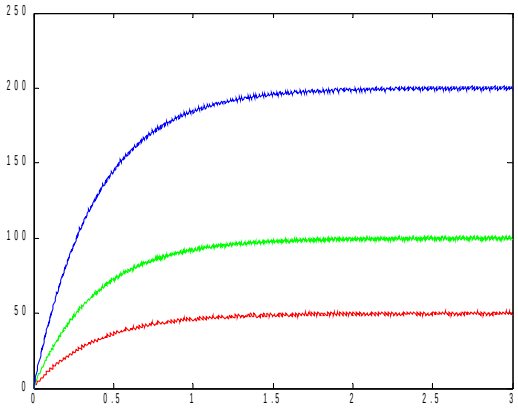

Ilustración 5.19: Olvido proporcional 1%

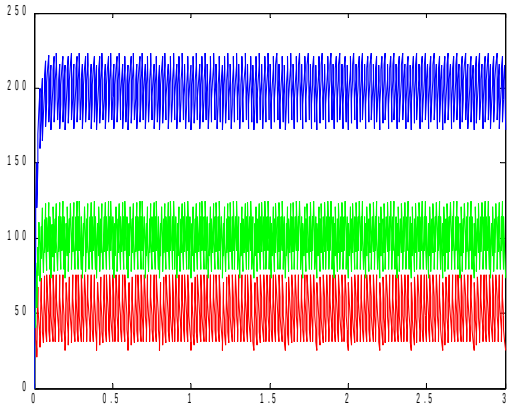

Ilustración 5.20: Olvido proporcional 20%

Puede observarse que en ambos casos la respuesta del sistema es lineal a la frecuencia, de forma que el valor reconstruido es proporcional a la frecuencia. Ajustando adecuadamente la ganancia del evento, el valor reconstruido sería el mismo con ambos factores de olvido. Sin embargo, para un factor de olvido mayor, el valor de la neurona oscila con mayor amplitud alrededor de su valor medio. Escoger sin embargo un factor de olvido demasiado bajo reduce la

respuesta dinámica del sistema al necesitar más tiempo en ir aumentando su valor hasta alcanzar el valor final. En la gráfica de la izquierda se ve que es necesario un tiempo similar al tiempo de frame hasta que el valor reconstruido alcanza el valor deseado, mientras que en el caso de la derecha, el valor se alcanza en menos de una décima parte de un frame. Se entiende aquí por frame el tiempo necesario para que el emisor coloque en el bus un numero de eventos igual al valor de la neurona.

5.4.3 Otros mecanismos de olvido

En todos los mecanismos de transmisión/reconstrucción AER debe aparecer el olvido que limite el proceso de integración. Además de los dos mecanismos de olvido (por frames y continuo) descritos anteriormente, puede haber otros mecanismos que induzcan el olvido. A continuación se describe el olvido explicito, asociado a un tipo de eventos especiales, y el olvido intrínseco, que aparece cuando se intenta reconstruir una imagen por el mecanismo de la frecuencia instantánea.

5.4.3.1 Olvido explícito

El olvido explicito ocurre cuando se recibe un evento de signo negativo, que en lugar de incrementar el valor de la integración lo disminuya. Los eventos negativos aparecen, como se comenta en el capítulo 7, al utilizar AER diferencial, con eventos negativos y positivos, como resultado por ejemplo de calcular una convolución sobre una determinada imagen de entrada utilizando el mapeador probabilístico.

En el caso de imágenes con eventos negativos como resultado del cálculo de una convolución, el olvido explícito coexiste junto a un olvido progresivo o por frames, dado que los eventos negativos aparecen como resultados intermedios de la convolución que se está calculando. Sin embargo, en la transmisión AER diferencial pura, el olvido explícito es el único mecanismo de olvido que aparecería, con el inconveniente de que los errores cometidos en la reconstrucción se irían acumulando. Se comenta más detenidamente en la transmisión AER diferencial (capítulo 7).

5.4.3.2 Olvido intrínseco o implícito

El último mecanismo de olvido, el olvido intrínseco no es un mecanismo de olvido como tal, pero cumple la función del olvido cuando se realiza la reconstrucción. En el caso del framegraber instantáneo, no aparece ningún elemento que vaya substrayendo el valor que vamos a reconstruir. Dada la naturaleza del framegraber instantáneo, el valor reconstruido se calcula a partir del periodo

entre dos eventos consecutivos.

$$Freq = \frac{1}{T_{medido}} \tag{5.19}$$

Desde el punto de vista teórico, la frecuencia de eventos asociada a una neurona con un valor determinado viene dada por la expresión:

$$Freq = Valor * F_{base} \tag{5.20}$$

Por lo tanto, para reconstruir

$$Valor = \frac{Freq}{F_{base}} = \frac{1}{T_{medido} * F_{base}} \tag{5.21}$$

El parámetro F_{base} relaciona el periodo T, que es el inverso del número de eventos por unidad de tiempo, con el valor a reconstruir. En este caso la constante F_{base} implícitamente fija la relación entre el la cantidad de eventos en el bus, a partir de su separación en el tiempo, y el valor reconstruido, función que desempeñaría la magnitud del olvido en el caso de otros casos de reconstrucción.

6. Fuentes de error en transmisiones AER

En todo proceso de transmisión de información hay una señal adicional no deseada de ruido, que se mezcla con la información que deseamos transmitir, de forma que la señal recibida se vería en mayor o menor medida alterada por ese ruido no deseado. Si el nivel de ruido es muy elevado puede hacer imposible la reconstrucción de la señal original o que se reconstruya muy distorsionada. La potencia en una transmisión AER, para un determinado píxel, es proporcional al número de eventos para ese píxel. Cuando se asocia a cada valor de luminosidad un número de eventos en la generación AER estamos precisamente determinando algo equivalente a la potencia de la señal, de forma que aumentando el número de eventos en dicha relación consumimos más ancho de banda pero aumentamos la potencia de la señal (por tanto, su relación señal ruido).

Existen diversas fuentes de error. Algunas de estas fuentes de error son intrínsecas a las características del sistema AER, a la discretización producida por los sistemas digitales, o introducidas por las particularidades de la implementación de un método de conversión en concreto. Este tipo de fuentes de error no pueden ser evitados al ser inherentes al sistema, aunque conviene conocerlas para utilizar el mecanismo de generación, más adecuado en cada momento.

Otro tipo de fuentes de error, que llamamos errores externos, son los introducidos por fenómenos ajenos a la conversión y reconstrucción AER. En un bus AER, físicamente implementado como un cable plano ATA-40, se supone que la impedancia entre emisor, receptor y medio se encuentran perfectamente adaptadas, por lo que no hay ondas reflejadas que imposibilitarían la comunicación. Entre líneas de datos debería haber líneas de masa que minimizaran el efecto de crosstalking. Sin embargo, no por ello la transmisión esta libre de posibles alteraciones debidas fundamentalmente a ruido electromagnético externo. Este ruido electromagnético puede ser tan fuerte en algunos momentos que altere el valor de un bit.

6.1 Fuentes internas

Como fuentes internas de error se consideran aquellas no producidas por interferencias externas al sistema, sino los errores introducidos por elementos implicados en la transmisión AER. Si se supone un medio de transmisión ideal libre de errores y en el cual las señales se transmiten sin ningún retraso entre emisor y receptor, al realizar una implementación de un sistema AER pueden cometerse errores en la ubicación de los eventos que provocarían que la reconstrucción de la imagen difiriese ligeramente de la imagen original transmitida.

Aunque el método repartiera homogéneamente los eventos para un píxel determinado, de manera que la separación entre dos eventos cualesquiera fuese siempre la misma, al existir más de un píxel conformando la imagen, con diferentes frecuencias, existe la posibilidad de que dos eventos de dos píxels deban ser enviados en el mismo instante de tiempo, de manera que uno de ellos deba ser adelantado o retrasado en el tiempo, difiriendo su separación respecto a su antecesor y sucesor de la del resto de los eventos. En el receptor se interpretaría estas variaciones en la frecuencia instantánea del píxel como pequeñas fluctuaciones en su nivel de gris. Adicionalmente, simplificaciones introducidas en el reparto de eventos realizadas en implementaciones hardware o software introducirían errores similares [MORGADO01].

6.1.1 Retrasos del protocolo

El sistema AER se basa en un bus asíncrono, basado en un protocolo de handshake de 4 estados. El tiempo que un evento está en el bus es función de la velocidad del emisor y del receptor. Debido a la multiplexión en el tiempo que se realiza en el bus, cuando dos eventos se emiten muy próximos, el segundo de ellos se vería retrasado hasta haber completado la transmisión del primero. Si se conectan dos dispositivos de velocidades muy diferentes, el rendimiento se reducirá hasta adaptarse a la velocidad del más lento. La resolución temporal de un dispositivo es función directa de su velocidad, por lo que en un dispositivo lento es menor que en uno rápido.

6.1.2 Errores introducidos por la implementación

Estos errores son introducidos por limitaciones al implementar un sistema de naturaleza analógica como los sistemas AER en dispositivos digitales. En los sistemas AER analógico de visión, cada elemento fotoreceptor emitiría impulsos con una frecuencia que varía de manera continua a lo largo del tiempo en función de la luminosidad recibida. Si se considera un único píxel aislado, se puede despreciar la influencia de otros eventos en la ubicación de los eventos del píxel en estudio.

Existen errores introducidos por los sistemas digitales a la hora de emular el comportamiento de sus equivalentes analógicos.

6.1.2.1 Error de discretización del nivel de gris

Este error no es debido al bus AER propiamente dicho, sino a los sistemas digitales en general. Se produce al convertir un nivel de luminosidad continuo, en valores discretos. Para un número de bits elevado, el error normalizado que se produce es pequeño:

$$E_{disc} = \frac{1}{2^{nbits}} \tag{6.1}$$

Hay que tener en cuenta, que para un número de bits alto, el número de eventos crece exponencialmente, por lo que también lo hará el ancho de banda necesario para transmitir la imagen. En una retina AER, donde una tensión proporcional a la luz captada por un fototransistor modula un VCO, de forma teórica son posibles infinitas frecuencias dentro de un rango determinado. Un generador digital, discretizará por el contrario el número de niveles de gris, y por lo tanto los valores que la frecuencia podrá tomar para un píxel.

6.1.2.2 Error de discretización en frames

Este error se presenta cuando se utiliza como origen para la emisión AER una fuente de vídeo que divide el tiempo en frames. El número de frames por segundo (FPS) indica la frecuencia con la que se muestrea la luminosidad para un píxel determinado. Por el teorema del muestreo se podrán reconstruir correctamente variaciones de luminosidad cuya frecuencia sea inferior a la mitad de la frecuencia de frame.

$$F_{MAX} \leq \frac{1}{2 * T_{FRAME}} \tag{6.2}$$

Sin embargo, esta limitación no se debe al sistema AER sino a la fuente de vídeo utilizada para generar las secuencias AER. Si se utilizara una fuente de vídeo continua, la frecuencia máxima vendría determinada por el tiempo de slot.

$$T_{SLICE} = \frac{T_{FRAME}}{Nivel_{MAX}} \tag{6.3}$$

6.1.2.3 Error por discretización en 'slot'

Este tipo de error se produce al dividir el tiempo de un frame en slices, y asignar a cada píxel un determinado slot dentro de ese slice. Al dividir un frame en varios slices o scans, y asignar una posición fija a cada evento, que puede ser o no emitido, estamos discretizando el periodo entre eventos a un múltiplo de T_{slice}. Por lo tanto el error máximo que cometeremos al aproximar la posición de un evento a su slice más próximo será la mitad del tiempo de slice:

$$E_{SLOT} = \frac{T_{SLICE}}{2} \tag{6.4}$$

6.1.2.4 Error introducido por el método de conversión

Para un píxel determinado, se tiene un slot reservado dentro del 'slice' que representaría a un barrido completo de la imagen. El criterio por el cual un slot debe ser ocupado o no por un evento depende del método de conversión implementado. La luminosidad de un píxel determina la frecuencia o numero de ocurrencias por unidad de tiempo con que debe aparecer su evento, o lo que sería lo mismo, la proporción de slots a usar del total asignado. Esto debe cumplirse en todos los

métodos. Además, la distribución de los slot usados debe ser homogénea en el caso ideal. En función de la complejidad del método de generación, la distribución realizada por los diferentes métodos variará, introduciendo errores cuya importancia depende de la irregularidad de la distribución. Este caso se ha estudiado con más detenimiento en el capítulo dedicado a la generación AER hardware.

6.2 Fuentes externas

Las fuentes externas de errores más frecuente en un sistema electrónico es el ruido electromagnético generado por el propio sistema, sistemas cercanos, ruidos en la alimentación, etc. Los campos electromagnéticos generados por los dispositivos eléctricos y electrónicos pueden inducir corrientes en conductores situados en sus proximidades. Si las líneas de transmisión no cuentan con sus impedancias bien adaptadas, para altas frecuencias y cables de longitud elevada pueden aparecer reflexiones en las líneas que se sumarían a la señal, produciendo errores en la transmisión.

Sus efectos pueden minimizarse teniendo en cuenta ciertas consideraciones básicas como utilizar cables de una longitud lo más corta posible, de forma que se minimizan los efectos de inducción. La impedancia del emisor, receptor y cable de transmisión correctamente adaptada elimina ondas reflejadas que se superpondrían a la señal original. Utilizar un cable con líneas conectadas a GND entre las señales de datos para reducir el crosstalking. Utilizar una fuente de alimentación bien estabilizada. Evitar cables largos en la alimentación que actuarían como inductancias reduciendo la respuesta de la fuente a alta frecuencia, etc.

6.2.1 Errores en las líneas de datos

Los errores en las líneas de datos implican que durante la transmisión de un evento, el contenido de al menos una línea de datos es alterado de forma que el evento se recibe pero para un píxel inadecuado. Al poder ser alterada cualquier línea de datos, el píxel resultante no tiene por que presentar proximidad espacial con el píxel original. Sin embargo, la dirección alterada no es aleatoria. El nuevo código es muy próximo al original atendiendo a la distancia de bits de ambas palabras. El nuevo píxel no tiene por que estar cerca del original espacialmente, pero sí en su código binario. Para un código de n bits, existen n posibles eventos a distancia 1, n*n para distancia 2, etc. Como regla general:

$$N_{eventos} = numbits^{distancia} \tag{6.5}$$

Aunque la probabilidad de que se produzcan más de un error sería muy pequeña. De manera teórica, la probabilidad de que haya un error en n líneas sería:

$$Prob_n = P_{error}{}^n \tag{6.6}$$

Esto sin embargo no tiene por que ser cierto. Si la causa del error se debe a la inducción de un campo externo debido a un mal aislamiento, esta alteración puede ocurrir simultáneamente en varias líneas próximas físicamente. El fenómeno de crosstalking puede aparecer como resultado acumulativo de las inducciones provocadas por los cambios en líneas adyacentes. Cada línea influirá por igual a las líneas más próximas a ambos lados. Algunos métodos de generación pueden propiciar más este tipo de errores.

Debido a que teóricamente este error afecta por igual a todos los eventos, para una imagen determinada, en promedio, todos los píxel se verán afectados de la misma manera por este error. Los píxels muy activos perderían eventos, que serían recibidos por otros, independientemente de la actividad que presente. Al ser un valor constante, independiente de la actividad del píxel receptor, los píxels con poca actividad serán más vulnerables a estos eventos añadidos.

6.2.2 Errores en las líneas de protocolo

El protocolo de handshake del bus AER consiste en un protocolo asíncrono básico de cuatro estados (ver ilustración 6.1). El emisor emite un evento, y espera que el receptor lo valide activando ACK, entonces desactivará REQ y espera a que el receptor desactive ACK antes de repetir el proceso para enviar el siguiente evento.

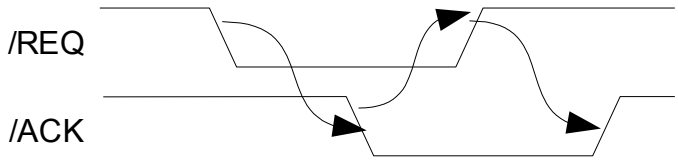

Ilustración 6.1: Protocolo AER req/ack

Un error introducido en alguna de estas dos líneas puede producir en ciertas circunstancias

errores en en intercambio de eventos, provocando que se generen eventos aleatoriamente, que se pierdan o dupliquen eventos durante su envío.

6.2.2.1 Falso request

El falso request es de los errores en las líneas de protocolo el más importante debido a que es el que tiene mayor probabilidad de darse. Puede producirse en cualquier momento estando el bus en reposo (ver ilustración 6.2).

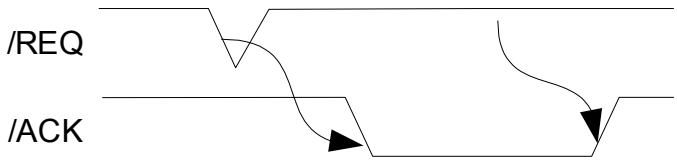

Ilustración 6.2: Falso request

Se produciría porque el receptor detecta erróneamente una activación de la señal request. El receptor activaría ACK por un breve espacio de tiempo como respuesta a la activación de REQ. El emisor ignoraría la activación de ACK, al no estar emitiendo ningún evento y mantendría REQ desactivado. El receptor, una vez finalizada la perturbación, al observar REQ desactivado lo interpreta como una respuesta a su activación de ACK, y desactivaría ACK. Habiéndose recibido un evento ficticio.

La dirección del evento recibido dependería de la implementación del emisor en concreto. De encontrarse el bus de datos en alta impedancia (protocolo SCX) el valor recibido puede ser cualquiera.

6.2.2.2 Doble request

El error “doble REQ” se produce por la desactivación por consecuencia del ruido de la señal REQ durante el envío de un evento. Suponiendo que la probabilidad de que se produzca un error es constante a lo largo del tiempo, hay más probabilidad de que este error se produzca cuanto más tiempo este la señal REQ activa (más lento sea el emisor) y más rápido sea el receptor (más sensible al ruido de alta frecuencia). Se muestra en la ilustración 6.3.

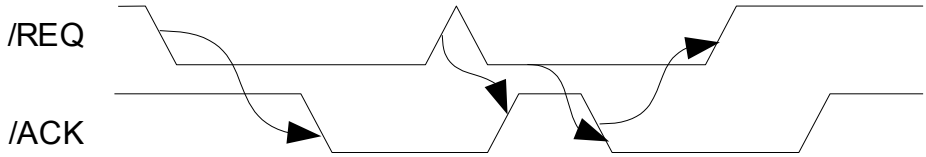

Ilustración 6.3: Errores AER: doble request

El emisor colocaría el evento en el bus, activando REQ. El receptor activaría ACK en respuesta. Si antes de que el emisor detecte la activación de ACK, la señal REQ se desactiva brévemente como consecuencia de ruido, el receptor interpretaría la desactivación de REQ como respuesta a ACK, desactivando ACK. Finalizada la perturbación, al estar REQ activada aún (ACK ha sido desactivada), volvería a activarse ACK en el receptor, interpretándolo como un nuevo evento. Como consecuencia, el mismo evento se recibiría duplicado.

6.2.2.3 Falso ACK

El falso ACK se produce cuando el ruido afecta en este caso a la señal ACK, durante la emisión de un evento. Tiene más probabilidad de producirse cuando el receptor es lento comparado con el emisor. El emisor activaría REQ, antes de que el receptor haya recibido esta petición, el emisor detecta ACK activada como consecuencia de alguna perturbación exterior, e interpreta que el receptor ha recibido el evento, y pasa a desactivar REQ (ver ilustración 6.4).

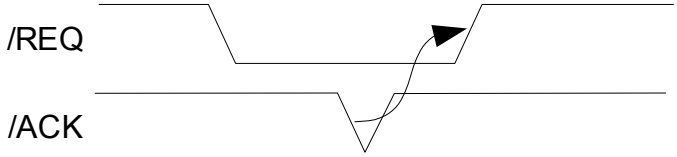

Ilustración 6.4: Errores AER: falso ACK

En este caso el evento se perdería y nunca sería recibido por el receptor. Cuanto más lento sea el receptor, más tiempo estará ACK desactivada, pudiendo una repentina activación como consecuencia del ruido confundir al emisor.

6.2.2.4 Doble ACK

Este error puede darse cuando el receptor es muy lento comparado con el emisor. Tras haber activado ACK como respuesta a la activación de REQ, el emisor desactiva la señal de REQ y espera que el receptor desactive ACK antes de enviar un nuevo dato. La desactivación durante un instante debido a una interferencia externa puede provocar que el emisor emita el siguiente evento. Como el receptor no había desactivado aún ACK, el receptor desactiva REQ al interpretar ACK activa como el reconocimiento del nuevo evento. Por esto, el segundo evento se pierde (ilustración 6.5).

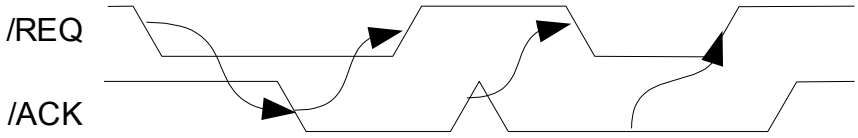

Ilustración 6.5: Errores AER: doble ACK

6.3 Consideraciones para reducir el ruido

Teniendo en cuenta todo lo comentado anteriormente, hay ciertas consideraciones de diseño a destacar para minimizar estos efectos. Es deseable utilizar cables AER lo más cortos posible. Utilizar una frecuencia de reloj adecuada a las necesidades de ancho de banda del bus, reduciéndose la frecuencia en las aplicaciones que lo permitan.

Para minimizar los errores en las líneas de protocolo, además de las consideraciones en el diseño del PCB (ajustar las impedancias adecuadamente, aislar convenientemente las lineas de datos para evitar crosstalking, prestando especial atención a las líneas de protocolo, etc.), en el diseño VHDL de las máquinas de estado de envío/recepción de eventos, pueden filtrarse los pulsos de pequeña duración (menos de un ciclo de reloj) para detectar falsas activaciones.

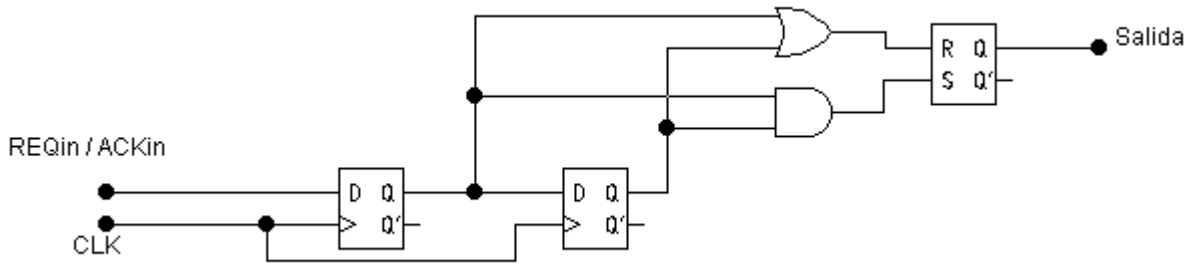

Ilustración 6.6: Circuito biestables para eliminar glitches

Respecto a las líneas de datos, hay que intentar minimizar las posibles interferencias con otras líneas de datos para evitar el crosstalking. Los errores debidos a interferencias electromagnéticas esporádicas, no debidas al crosstalking, afectan por igual a todos los eventos. Debido el carácter redundante del bus AER estos errores no son preocupantes para los píxels con mucha actividad. Incluso para píxels con poca actividad, al generar pocos eventos, sufrirán también pocos errores (la probabilidad del error se supone igual para todos los eventos). Sin embargo, son más vulnerables a este tipo de errores, sobre todo como receptor de eventos erróneo correspondientes en realidad a otros píxels de mucha actividad con dirección muy similar. El protocolo AER no implementa ningún mecanismo que permita detectar este tipo de errores.

Para minimizar este error pueden tomarse varias medidas:

- Reducir la frecuencia del reloj en la medida que las necesidades del bus lo permitan.
- Añadir información adicional al evento (bit de paridad, o CRC), que permita desechar los eventos erróneos.

7. AER diferencial

Como se ha visto en capítulos anteriores en esta tesis, en algunas situaciones es necesario recurrir a la utilización de eventos con signo. Si nos vamos a un caso más general de transmisiones AER, al margen de la transmisión de imágenes, tenemos que la transmisión AER puede emplearse para transmitir cualquier otro tipo de información, proveniente de diversos tipos de sensores. En el caso de una imagen, la actividad de una neurona, y por tanto la frecuencia de aparición de su dirección en el bus dependía directamente de la cantidad de luz recibida por el sensor. En otro caso de sensores, de temperatura, presión, velocidad, etc., el número de eventos dependería respectivamente del valor de temperatura, presión, o velocidad medido por cada sensor determinado.

Si nos fijamos por ejemplo en una señal sonora transmitida por el aire como una onda de presión que es captada por un oído artificial que convertiría estos niveles de presión en un flujo de eventos. Para el caso del sonido, la información no se encuentra codificada en la presión que captamos con el oído, sino en las frecuencias de los cambios de presión. Lo interesante a transmitir aquí no es el valor de presión captado, sino las variaciones de dicha presión. Las variaciones de una magnitud, expresada como la diferencia entre el valor actual del sensor respecto al último valor medido, pueden por tanto tomar valor positivo o negativo. Una neurona tiene que transmitir por lo tanto la magnitud de la variación además de su signo. El valor de la magnitud en valor absoluto viene dado por la frecuencia de aparición de la dirección de la neurona en el bus. Existen oídos AER, también llamados cochleas [CHAN01][OTNES01], que los eventos representan la intensidad del sonido en una determinada espectral, en este caso no sería necesario recurrir a eventos negativos.

Para transmitir el signo o dirección de la derivada se recurre por lo tanto a los eventos con signo. Para que haya eventos con signo, cada neurona tiene asociada entonces dos direcciones distintas, una que representa su estímulo positivo y otra para representar su estímulo negativo. Las dos direcciones asignadas a cada neurona pueden ser cualesquiera, pero por simplicidad se les asigna dos direcciones iguales a excepción de un bit que indicaría el signo. Este bit adicional se traduce en una línea extra en el bus AER que representa el signo positivo o negativo. Cada neurona receptora ignoraría este bit de signo en la decodificación, y en el caso de que la decodificación fuese positiva, actuaría aumentando o disminuyendo su valor en función del bit de signo.

Los eventos con signo pueden ser útiles en los siguientes escenarios:

- Transmisión de variaciones en lugar de valores absolutos
- Impulsos inhibitorios al realizar algún tipo de procesado (ej. convoluciones)
- Olvido explícito (mejorar respuesta temporal).

El escenario que acaba de ser descrito en el cual se va a transmitir una señal sonora sería un caso del primer tipo de escenarios donde se utilizan eventos negativos. Si nos centramos de hecho en los circuitos tradicionales que trabajan con señales de audio, las diferentes etapas se acoplan entre ellas utilizando condensadores que eliminan la información de continua, dejando pasar únicamente la parte de alterna, que es donde se encuentra la información sonora.

Para el caso de imágenes, los valores de luminosidad recibidos son siempre positivos, por lo que no se emplearían eventos negativos. Si lo que nos interesa no es la imagen en si, sino los cambios o movimientos que se observan en ella, se puede recurrir entonces a una retina derivativa cuya salida sea función de la variación de luminosidad. De forma teórica se podría integrar esta información de variación y como resultado de la integración, obtener los valores de luminosidad de la imagen. Sin embargo, en este modelo, los errores en la transmisión se irían acumulando, por lo que resulta difícil reconstruir adecuadamente los valores de luminosidad. La principal ventaja de la retina derivativa sería que el trafico generado es mucho menor generalmente que la retina de intensidad.

El segundo tipo de escenario correspondería cuando en lugar de ser la etapa sensora la que emite eventos de ambos signos, es una segunda capa, que realiza un procesado sobre una señal de entrada, aunque los eventos recibidos sean todos positivos. Para cambiar el signo de un evento basta con un elemento mapeador de eventos que sustituya un evento positivo por su equivalente negativo. Hay que tener en cuenta que si a la entrada del procesado aparecen eventos negativos, el cambio de signo consiste en convertirlos en positivo. El maper trata los eventos en función de su dirección, sin distinguir este bit de signo con un significado especial, sino que cada uno de ambos eventos tiene su propia tabla de mapeado separada. Bastaría con programar convenientemente el maper que genere el evento de salida correcto para ambos tipos de eventos de entrada. En otro tipo de dispositivos, o incluso en el maper, con el fin de reducir los requerimientos de memoria o poder asignar más eventos de salida a cada evento de entrada, el bit de signo puede sacarse de la decodificación, de forma que se decodifiquen únicamente los bits de dirección del evento. El bit de signo recibido debe combinarse entonces con el generado internamente, para realizar la multiplicación de los signos. En

hardware esta operación se reduce a una sencilla puerta XOR. Al utilizar la misma información de mapeo para ambos eventos, reducimos la necesidad de memoria a la mitad. En el caso del maper esto significa que pueden realizarse convoluciones con kernels mayores.

El tercer tipo de escenario, en cual nos vamos a centrar en este capítulo, únicamente participarían dos elementos, una retina proporcional y un receptor basado en un integrador, con olvido progresivo. Si el receptor integrase por frames, el tiempo de frames determina el intervalo mínimo de tiempo en que puede apreciarse una variación de la luminosidad. En el caso de un integrador con olvido progresivo, vimos que era el factor de olvido el parámetro que fijaba la respuesta dinámica del sistema. Para factores de olvido elevados, el número de eventos en el bus debe ser mayor que para factores de olvido pequeño para transmitir el mismo valor de intensidad. Por contra, un factor de olvido reducido hacía que el sistema evolucionara muy lentamente, sobre todo cuando el sistema cambia de un píxel muy luminoso a otro poco luminoso. La pérdida de intensidad en el receptor recaería toda en el olvido progresivo. Para intentar reducir estas limitaciones, se propone el envío de eventos negativos para incrementar la velocidad con la que el receptor pierde "carga".

Como se ha visto en capítulos anteriores, el mecanismo de transmisión AER tiene ciertas limitaciones, a la hora de reconstruir una señal variable en el tiempo. Si nos centramos en un único píxel tendremos una señal continua variable que evoluciona con el tiempo. Cuando solo trabajamos con eventos positivos, el valor de dicho píxel viene dado por la frecuencia de dichos eventos. Un incremento en la frecuencia de eventos indica un aumento en la luminosidad de dicho píxel, mientras que una reducción de dicha frecuencia lo contrario. En este caso, es el factor de olvido el que atenúa el valor de dicho píxel. La pendiente máxima con la que puede reducirse la luminosidad de un píxel viene por tanto limitada por el factor de olvido.

Esto se representa en la ilustración 7.1. Como comentábamos en el capítulo 5.4.La pendiente tanto en la subida como en la bajada viene determinada por el factor de olvido y la ganancia asociada a cada evento. Un mayor factor de olvido viene asociado a una mayor ganancia por evento, a unos tiempos de subida y bajada más rápido, pero a su vez, con un mayor rizado de la señal reconstruida.

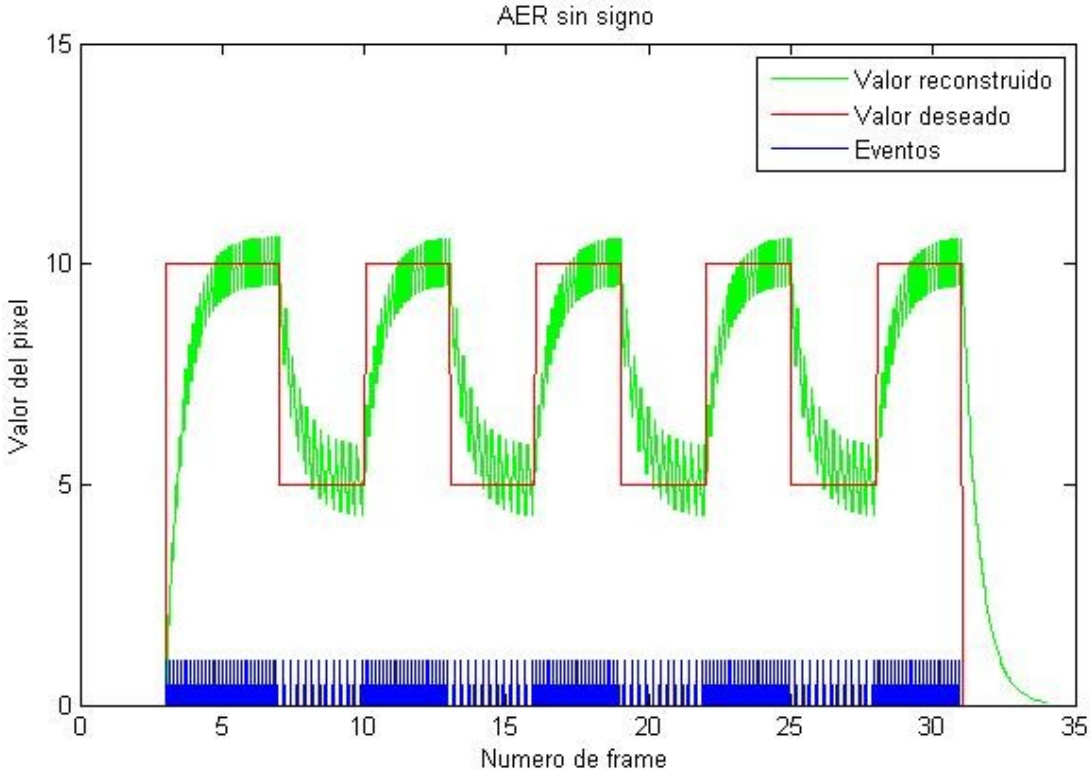

Ilustración 7.1: AER sin signo

Al introducir eventos negativos, los eventos positivos aumentarían la magnitud de la luminosidad, mientras que los eventos negativos la disminuirían. Si eliminamos el olvido, los eventos negativos serían los únicos responsables de la disminución de dicha magnitud, por lo que el valor reconstruido no sería función ya de la frecuencia de pulsos, sino la derivada o variación de dicha magnitud la que va en función de la frecuencia y tipo de eventos.

En una transmisión AER diferencial, los eventos AER, no codifican el valor de la magnitud a representar, sino su variación. En una transmisión ideal, en el lado del receptor bastaría con integrar la información recibida para reconstruir la señal original. El integrador sería un integrador sin pérdidas, y corresponde a los eventos negativos decrementar el valor de la integral acumulada. El diagrama de bloques que lo representa sería:

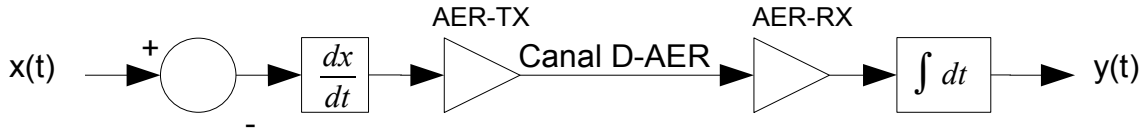

Ilustración 7.2: Transmisión Delta AER pura

La ventaja principal de este diseño diferencial respecto al AER clásico donde se transmite el valor de la variable (luminosidad del píxel) radica en que el número de eventos que se transmite por el bus es mucho menor, si se supone que la variación de luminosidad de la mayoría de los píxels será pequeña. Sin embargo, al no ser el canal de transmisión AER un canal ideal, y estar influenciado por diversos tipos de errores que pueden alterar los eventos que se reciben, la imagen reconstruida se verá afectada por dichos errores. En la transmisión AER no diferencial, este ruido se presenta como un ruido que se añade a la imagen original. En la transmisión AER diferencial, este error que se superpone, lo hace sobre la variación. Este error es acumulado en el integrador, y puede presentarse como un oscurecimiento o aumento de brillo progresivo con el tiempo (el error se presenta como una señal de continua que va variando en función de la distribución de los errores) sobre los distintos píxels de manera independiente.

En la ilustración 7.3 vemos el resultado de aplicar el esquema de transmisión AER diferencial sobre la misma señal de entrada del caso anterior. La gráfica verde indica el valor ideal de la reconstrucción, mientras que en rojo observamos el valor de la reconstrucción, pero suponiendo que el canal de transmisión no es ideal, por lo que está afectada por errores, y por tanto alguno de los eventos a transmitir se pierden debido a estos errores.

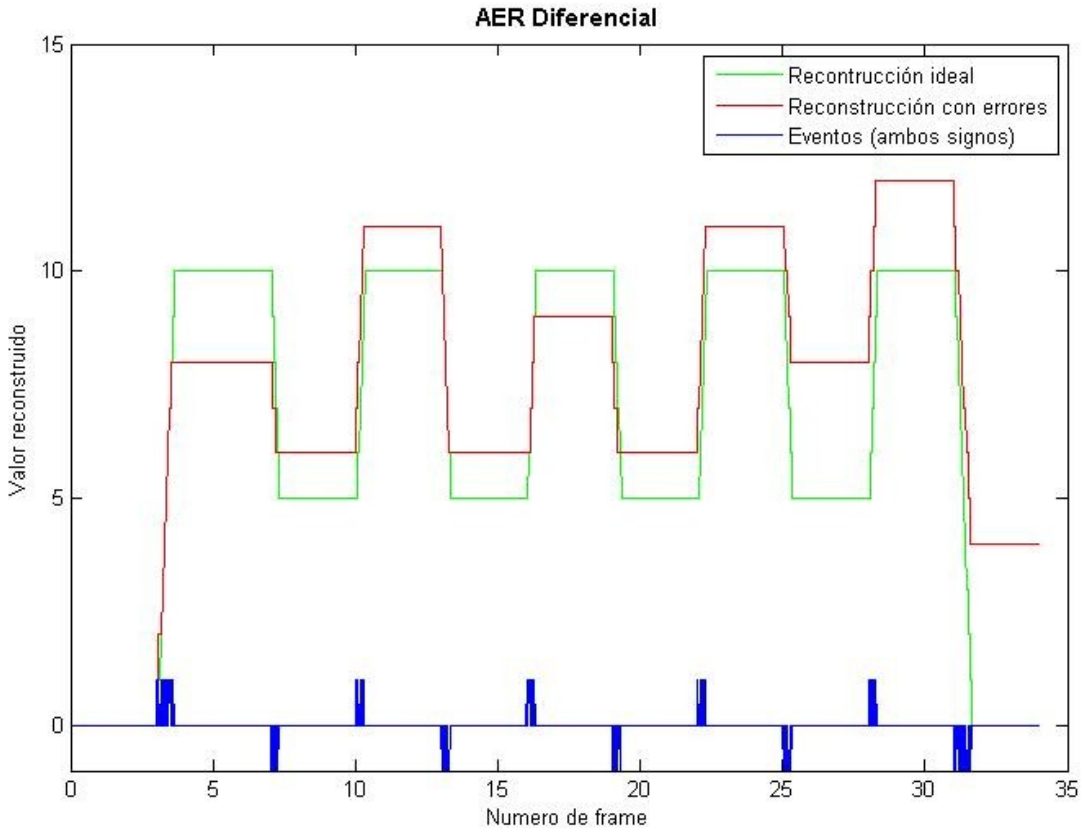

Ilustración 7.3: AER diferencial

Para estas simulaciones, con el fin de mejorar la claridad visual de las ilustraciones, se ha supuesto un valor máximo para cada píxel de 16, reduciéndose el número de slices, y por tanto de eventos necesarios para definir cada píxel. El error introducido en la transmisión se ha exagerado también fijándolo en una probabilidad del 20% de perder un evento con el fin de obtener mayor claridad a la hora de ver los efectos de dicho error sobre la señal reconstruida. En un montaje real, este error viene fijado por la velocidad de eventos, longitud y características físicas del cable, así como la ocupación del bus. En cualquier caso, el error real sería mucho menor (inferior al 1%)

En el caso de transmitir información del sonido, este error que se presenta como un nivel de continua se elimina en el acoplamiento capacitativo entre etapas amplificadoras, o incluso en el propio altavoz. La información sonora se encuentra codificada en las variaciones de presión de alta frecuencia (20Hz-20Khz), y nuestro propio oído es incapaz de percibirla. Sin embargo, para la transmisión de imágenes, la derivada de la luminosidad representa el movimiento que se produce en los diferentes píxels que conforman la imagen. El receptor podría percibir siempre el movimiento que se produce en la imagen. Sin embargo, con el tiempo podría ser incapaz de reconstruir la imagen estática que se está observando.

En los sistemas digitales de transmisión de video, más concrétamente en la compresión de las imágenes, con el fin de reducir el ancho de banda necesario para almacenar o transmitir una secuencia de video, se procede a comprimir no la información del frame, sino la diferencia entre un frame y el anterior. Sin embargo, en estos sistemas, para evitar la progresiva degradación de la imagen, se procede a transmitir la información completa de un frame periódicamente en lugar de la variación. Desde el punto de vista bio-inspirado, esto correspondería a realizar una especie de parpadeo, en el cual emisor y receptor se reinician con una imagen negra, y a continuación, al abrir los ojos, la derivada entre esta primera imagen y la imagen en negro corresponde a la imagen original.

Volviendo al bus AER diferencial, esta resincronización supondría:

- Codificar de alguna manera el impulso de “resincronización” para informar al receptor.
- Reenviar la información de luminosidad completa de la imagen.

El codificar el pulso de resincronización podría resolverse utilizando una dirección AER específica que conozcan tanto emisor como receptor. Habría que tener en cuenta que los errores del bus AER podrían provocar falsas resincronizaciones, por lo que debería codificarse como un tren de eventos, para que por medio de la redundancia,pueda garantizar que no se debe a errores en el bus.

El enviar la información completa de la imagen provoca un gran tráfico en el bus debido a que hay que enviar toda la información de luminosidad de toda la imagen.

Como alternativa a este mecanismo, se podría realimentar, con el valor reconstruido de forma que si éste se va separando del valor original, este error se realimenta de forma que pueda ser compensado, como se indica en el diagrama de bloques de la ilustración 7.4:

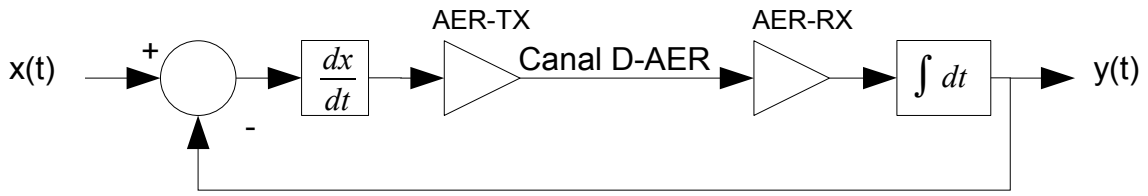

Ilustración 7.4: Esquema teórico Delta AER con realimentación

Este esquema sin embargo presenta el inconveniente de tener que realimentar, posiblemente mediante otro bus AER, al emisor con la información del error. Si se dispone de otro bus AER de retorno esto no plantearía ningún problema. A pesar de hacer más complejo el conjunto del sistema, el número total de eventos en el bus será menor al que se obtenían cuando se transmitía toda la información de luminosidad de la imagen constantemente.

El resultado es muy parecido a la codificación Sigma-Delta, muy utilizada en dispositivos digitales de sonido.

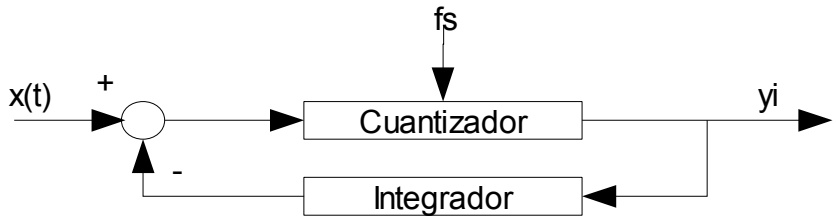

Ilustración 7.5: Modulación Delta-Sigma

La diferencia fundamental entre el AER diferencial, y la modulación Sigma-Delta utilizada en audio radica en que en la modulación Sigma-Delta, la existe una frecuencia de muestreo constante (fs), mientras que en el AER diferencial, no existe una frecuencia fija de muestreo, sino que la frecuencia a la que se enviarán los eventos positivos o negativos dependerá de la derivada de

la señal, o de la distancia entre el valor deseado y el retropropagado por el integrador. Si se duplica el integrador en el emisor, y se realimenta a partir de él, podríamos tener la realimentación que deseamos pero sin tener obtenerla desde el receptor, de forma similar a como funcionara el predictor de Smith utilizado en el control automático.

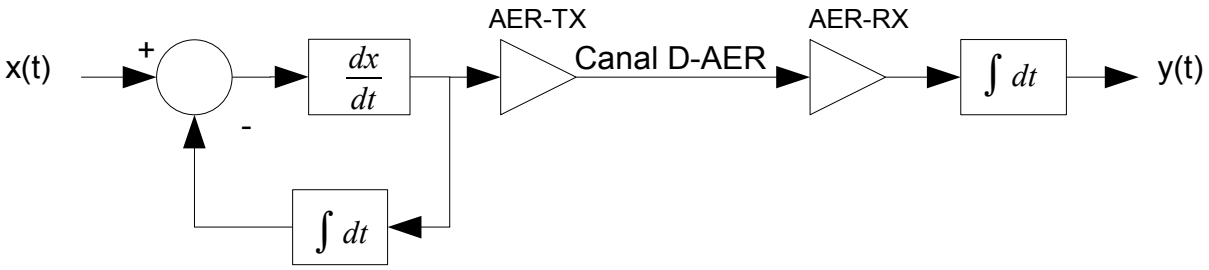

Ilustración 7.6: Esquema Delta AER con realimentación local

Este esquema sin embargo no es válido directamente puesto que en la realimentación no se contempla la posible influencia del error (ver ilustración 7.6), por lo que el integrador del emisor (para realimentación) y el del receptor acabarían distanciándose debido a que el error en la transmisión sólo afecta a uno de ellos. Si utilizamos, sin embargo, un integrador con perdida, en lugar de un integrador sin perdida como veníamos utilizando, podemos minimizar la influencia del error. Si consideramos la siguiente serie que describe el cálculo de una integral, tenemos:

$$Y_n = Y_{n-1} + X_n \tag{7.1}$$

$$Y_n = Y_{n-2} + X_{n-1} + X_n \tag{7.2}$$

$$Y_n = \sum_{i=0}^{n} X_i \tag{7.3}$$

Como se esperaba, Y_n es la suma de todos los eventos de entrada. Si sustituimos el integrador sin pérdidas en el esquema anterior por un integrador con pérdidas, la expresión que describe la serie se escribiría como:

$$Y_n = (1 - f_{olvido}) * Y_{n-1} + X_n \tag{7.4}$$

$$Y_n = (1 - f_{olvido}) * [(1 - f_{olvido}) * Y_{n-2} + X_{n-1}] + X_n \tag{7.5}$$

$$Y_n = \sum_{i=0}^{n} (1 - f_{olvido})^{(n-i)} * X_i \qquad (7.6)$$

Si tenemos en cuenta que Xi=(X'i+Ei), el error tiende a cero conforme va aumentando la antigüedad de la muestra.

En la ilustración 7.7 se ha representado para el mismo estímulo que en los casos anteriores, los eventos generados, el valor del integrador local, cuya función de olvido es idéntica a la del receptor, y el valor reconstruido con los errores introducidos durante la transmisión. Recordar que la influencia del error se encuentra exagerada (tasa de error del 20%), y que en un escenario real, este error será cientos o miles de veces menor. Vemos como aún así se suprime el efecto acumulativo del error que aparecía en la transmisión AER diferencial pura, y la respuesta en frecuencia mejora respecto al AER tradicional, con una mejor ocupación del bus.

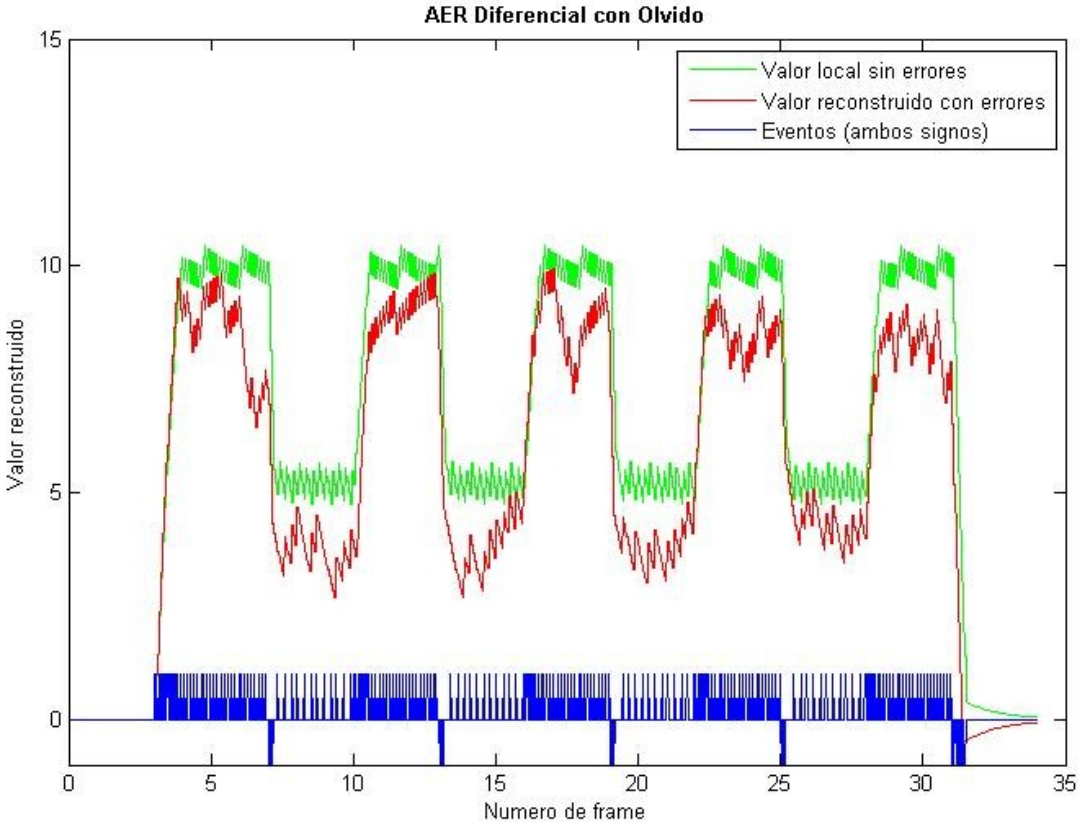

Ilustración 7.7: AER diferencial con olvido

8. Conclusiones (aportaciones)

- Se han implementado en hardware utilizando FPGA diversos dispositivos para la generación, transformación y reconstrucción de imágenes construyendo una cadena AER completa.
- En la conversión de imágenes a AER se ha diseñado en VHDL sobre la placa USB-AER dos métodos de generación, el método exhaustivo y el método random para generar eventos AER a partir de imágenes digitales.
- A partir de las implementaciones anteriores, se ha diseñado una retina sintética proporcional y otra derivativa que generarán eventos AER a partir de una señal de vídeo procedente de una cámara, para el posterior testeo de filtros. Esto permite tener fuentes generadoras de imágenes AER a un bajo coste, con elementos portátiles para aplicaciones empotradas.
- Se han probado distintos tipos de filtro utilizando un mapeador probabilístico sobre la placa USB-AER, así como sus limitaciones, configurando adecuadamente las tablas de mapeado con los eventos adecuados y sus probabilidades.
- Como transformación más significativa se realiza el cálculo de convoluciones utilizando eventos con signo. El mecanismo utilizado es novedoso, pues se realiza exclusivamente operando con pulsos al vuelo, sin cambiar la tipología de las señales recibidas, siendo su implementación digital.
- En la reconstrucción AER se han probado en hardware dos modos de reconstrucción, mediante un integrador y mediante el cálculo de la frecuencia de eventos. Este último se evalúa por primera vez.
- Se ha estudiado mediante simulaciones el problema del olvido en la reconstrucción. Se proponen y estudian varios modelos de olvido.
- Se han estudiado las diferentes fuentes de errores en las transmisiones AER, como la perdida de eventos, los retrasos introducidos sobre los eventos, o los introducidos en la generación por el método empleado, avanzando en la caracterización del bus AER.
- También se han estudiado las ventajas que aporta el empleo de eventos con signo, como forma de olvido explicito (AER diferencial), en el cálculo de convoluciones, o en la retina derivativa para indicar el signo de la derivada.

9. Trabajos futuros

Las posibilidades del procesado de imágenes basado en pulsos son mucho más amplias que las mostradas en este trabajo. Como mejoras futuras en las que investigar cabrían distinguir las siguientes:

- Los filtros AER diseñados se han obtenido por mecanismos deductivos. Las señales digitales son en muchos casos una representación del mundo analógico y existe una relación entre filtros analógicos y sus equivalentes digitales, estudiada y formalizada desde hace tiempo. De igual forma se podría sistematizar cómo deberían ser los filtros que actúan sobre las señales pulsantes para que estos hicieran funciones equivalentes a los filtros analógicos aplicados sobre sus correspondientes señales analógicas. Es decir, las señales pulsantes son una representación de la realidad con las que se puede operar, pero para la que no existe un formalismo.
- Las transformaciones de imágenes de vídeo en formato frame se han realizado en monocromo y con baja resolución. Esto sirve para sistemas de bajo coste en los que se requiera una elevada velocidad de procesado. El mecanismo propuesto no puede competir con las retinas AER, en cuanto a la velocidad de captación de imágenes, pues está limitado a 25 frames por segundo. Sin embargo, puede ser interesante aplicar estos mecanismos a secuencias de vídeo de mayor resolución y en color, puesto que las retinas actuales son de baja resolución y monocromas, lo cual permitiría los primeros estudios sobre imágenes AER en color.
- Los errores que se producen en la transmisión/transformación AER limitan la calidad de estos sistemas. Aunque el procesado es bastante robusto y eficaz el sistema no puede ser utilizado con fines de difusión de vídeo por estos errores o por las limitaciones de ancho de banda. La caracterización y eliminación de los errores es fundamental para que su uso pueda extenderse a otras aplicaciones. En este trabajo se han clasificado los errores, pero no se han cuantificado de forma definitiva, características tan importantes como la relación señal ruido quedan pendientes.

10. Bibliografía

[APSEL01] Alyssa B. Apsel, Andreas G. Andreou, "Analysis of data reconstruction efficiencyusing stochastic encoding and an integrating receiver", IEEE Transactions on Circuits andSystems II: Analog and Digital Signal Processing,Vol. 48, No. 10.Pág.890-897. 2001.

[ARIAS01] M. Arias-Estrada, D. Poussart, and M. Tremblay, "Motion Vision Sensor Architecture with Asynchronous Self-SignalingPixels", Proc. of the 7th Int. Workshop on Computer Architecture for Machine Perception (CAMP97),.Pág.75-83. 1997.

[AZADMEHR01] M. Azadmehr, J. Abrahamsen, and P. Häfliger, "A foveated AER Imager Chip", Proc. of the IEEE Int. Symp. on Circ. andSyst. (ISCAS2005),.Pág.2751-2754.Kobe, Japan 2005.

[BARBARO01] M. Barbaro, P.Y. Burgi, A. Mortara, P. Nussbaum, F. Heitger, "A 100x100 pixel silicon retina for gradient extraction with steering filter capabilities and temporal output codig", IEEE Journal of Solid-State Circuits,Vol. 37.Pág.160-172. 2002.

[BOAHEN01] Kwabena A. Boahen, "Communicating Neuronal Ensembles between Neuromorphic Chips", Neuromorphic Systems. Kluwer Academic Publishers,.Pág..Boston 1998.

[BOAHEN02] K.A. Boahen, "Point-to-Point Connectivity Between Neuromorphic Chips Using Address Events", IEEE Trns. on Circuits and Systems Part II,vol. 47 No. 5.Pág.416-434. 2000.

[BOAHEN03] K.A. Boahen, "A burst-mode word-serial address-event link-O: Transmitter design", IEEE Transactions on Circuits and Systems I - Regular Papers,.Pág.1269-1280. 2004.

[BOAHEN04] K.A. Boahen, "A burst-mode word-serial address-event link-II: Receiver design", IEEE Transactions on Circuits and Systems I - Regular Paper,Vol 51.Pág.1281-1291. 2004.

[BOAHEN05] K.A. Boahen, "A burst-mode word-serial address-event Link-III: Analysis and test results", IEEE Transactions on Circuits and Systems I - Regular Papers,Vol. 51.Pág.1292-1300. 2004.

[BOAHEN06] K. Boahen, "Retinomorphic Chips that see Quadruple Images", Proc. Int. Conf. Microelectronics for Neural, Fuzzy and Bio-Inspired Systems (Microneuro99),.Pág.12-20.Granada, Spain 1999.

[BOAHEN07] Kwabena A. Boahen, "Retinomorphic Vision Systems", Proceedings of FifthInternational Conference on Microelectronics for Neural Networks and Fuzzy Systems IEEE,Microneuro,.Pág.2-14.Laousanne 1996.

[BOAHEN08] Kwabena A. Boahen, "Retinomorphic vision systems I: Pixel design", Proc.IEEE International Symposium on Circuits and Systems, volume Supplement, ISCAS '96,.Pág.3-14. 1996.

[BOAHEN09] Kwabena A. Boahen, "Retinomorphic vision systems II: Communicationchannel design", Proceedings of the IEEE International Symposium on Circuits and Systems,ISCAS 1996, volume Supplement,.Pág.14-17. 1996.

[CAVIAR01] , "Proyecto CAVIAR", ,.Pág.. .

[CHAN01] V. Chan, S.C. Liu, A. Van Schaik, "AER EAR: A Matched Silicon Cochlea Pair with Address-Event-Representation Interface", IEEE Transactions on Circuits and Systems I,Vol 54, No. 1.Pág.48-59. 2007.

[CLEMENTE01] Mª Carmen Clemente Medina, Alberto Peinado Domínguez, "Caracterización de Secuencias Binarias Pseudoaleatorias generadas mediante LFSR", XVIIISimposium Nacional de la URSI (Unión Radio-Scientific Internationale,.Pág.. 2003.84-9749-081-9

[COSTAS01] J. Costas-Santos, et al, "A contrast retina with on-chip calibration for neuromorphic spike-based AER vision systems", IEEE Transaction Circuits Systems. ,vol 54, nº7.Pág.1444-1458. 2007.

[CULURCIELLO01] E. culurciello, R. Etienne-Cumming y K.A. Boahen, "A biomorphic digital image sensor", IEEE journal of Solid-State Circuits,Vol 38.Pág.. 2003.

[DELORME01] A. Delorme, S. J. Thorpe, "Face identification using one spike per neuron: resistance to image degradations", Neuronal Networks,.Pág.14:795-803. 2001.

[DELORME02] A. Delorme, G. Richard, M. Fabre-Thorpe, "Ultra-rapid categorisation of natural images does not rely on colour: A study in monkeys and humans", ,Vis. Res 40.Pág.2187-220. 2000.

[FABRE01] M. Fabre-Thorpe, A. Dlorme, C. Marlot y S. J. Thorpe, "A limit of the speed of processing in ultra-rapid visual categorization and novel natural scenes", J. Comp. Neurol. 13,Vol 2.Pág.171-180. 2003.

[FABRE02] M. Fabre-Thorpe, G. Richard y S. J. Thorpe, "Rapid categorization of natural images by rhesus monkeys", NeuroReport 9(2),.Pág.303-308. 1998.

[GOLDBERG01] David H. Goldberg, Gert C. Cauwenberghs, Andreas G. Andreou, "Address-Event Representation for Communication and Computation in Neuromorphic VisionSystems", Proceedings of the Fourth International Conferences on Cognitive and NeuralSystems, ICCNS00,.Pág.. 2000.

[GOLDBERG02] David H. Goldberg, Gert C. Cauwenberghs, Andreas G. Andreou, "Analog VLSIspiking neural network with address domain probabilistic synapses", The 2001

IEEEInternational Symposium on Circuits and Systems, 2001. ISCAS 2001,Vol. 2.Pág.241-244. 2001.

[GOLDBERG03] David H. Goldberg, Gert C. Cauwenberghs, Andreas G. Andreou, "ProbabilisticSynaptic Weighting in a Reconfigurable Network of VLSI Integrate-and-Fire Neurons", Neural Networks. NN01, Elsevier Science,Vol. 14, No. 6-7.Pág.781-793. .

[GOLOMB01] S.W. Golomb, "Shift Register Sequences", 1982

[GOMEZ01] Francisco Gomez Rodriguez, Alejandro Linares Barranco, Rafael Paz Vicente, Lourdes Miró Amarante, Gabriel Jimenez Moreno, Antonio Abad Civit Balcells, "Aer Image Filtering", Proceedings of SPIE, the International Society for Optical Engineering..6592.Pag659207-1 a 659207-102007

[GOMEZ02] Francisco Gomez Rodriguez, Rafael Paz Vicente, Lourdes Miró Amarante, Alejandro Linares Barranco, Gabriel Jimenez Moreno, Antonio Abad Civit Balcells, "Two Hardware Implementation of the Exhaustive Synthetic Aer Generation Method", Lecture Notes in Computer Science.3512.Pag534-5402005

[GOMEZ03] Francisco Gomez Rodriguez, Rafael Paz Vicente, Alejandro Linares Barranco, Manuel Rivas Pérez, Lourdes Miró Amarante, Saturnino Vicente Diaz, Gabriel Jimenez Moreno, Antonio Abad Civit Balcells, "Aer Tools for Communication and Debugging", IEEE International Symposium on Circuiits and Systems. ISCAS 2006IEEE,.Pág.3253-3256.Kos, Grecia 2006.0-7803-9390-2

[GOMEZ04] F. Gomez-Rodriguez, A. Linares-Barranco, L. Miró, S.C. Liu, A. Van Schaik, R. Etienne-Cumming, M.A. Lewis, "AER Auditory Filtering and CPG for Robot Control", ISCAS,.Pág.. 2007.

[GROSSBERG01] S. Grossberg, E. Mingolla, and W. D. Ross, "Visual Brain and Visual Perception: How Does the Cortex do Perceptual Grouping?"", Trends in Neuroscience,Vol. 20.Pág.106-111. 1997.

[HAFLIGER01] P. Hafliger, "Adaptative WTA with an Analog VLSI Neuromorphic Learning Chip", IEEE Transactions on Neural Networks,Vol. 18, No. 2.Pág.551-572. 2007.

[HARRIS01] X. G. Qi, X.; Harris, J,, "A Time-to-first-spike CMOS imager", Proc. of the 2004 IEEE International Symposium onCircuits and Systems (ISCAS2004),.Pág.824-827.Vancouver, Canada 2004.

[HIGGINS01] C.M. Higgins and S.A. Shams, "A Biologically Inspired Modular VLSI System for Visual Measurement of Self-Motion", IEEE Sensors Journal,Vol. 2, No. 6.Pág.508-528. 2002.

[HIGGINS02] Charles M. Higgins, Christof Koch, "A Modular Multi-Chip

NeuromorphicArchitecture for Real-Time Visual Motion Processing", Integrated Circuits and SignalProcessing, Kluwer Academic Publishers,Vol. 24.Pág.195-211. 2000.

[HIGGINS03] Charles M. Higgins, Christof Koch, "Multi-chip neuromorphic motionprocessing", Proceedings 20th Anniversary Conference on Advanced Research in VLSI,.Pág.309-323.Atlanta, GA 1999.

[HIGGINS04] Charles M. Higgins, Christof Kock, "Multichip Neuromorphic MotionProcessing", Conference on Advanced Research in VLSI,.Pág.. 1999.

[INDIVERI01] G. Indiveri, A.M. Whatley, and J. Kramer, "A Reconfigurable Neuromorphic VLSI Multi-Chip System Applied to VisualMotion Computation", Proc. Int. Conf. Microelectronics for Neural, Fuzzy and Bio-Inspired Systems (Microneuro99),.Pág.37-44.Granada, Spain 1999.

[JIMENEZ01] A. Jiménez-Fernández, R. Paz-Vicente, M. Rivas, A. Linares-Barranco, G. Jiménez, A. Civit., "AER-based robotic Closed-loop control system", ISCAS,.Pág.. 2008.

[JIMENEZ02] A. Jiménez-Fernández, A. Linares-Barranco, R. Paz-Vicente, C.D. Luján-Martínez, G. Jiménez, A. Civit, "AER and dynamic systems co-simulation over Simulink with Xilinx System Generator", ICECS,.Pág.. 2008.

[JIMENEZ03] A. Jiménez-Fernández, C.D. Luján, A. Linares-Barranco, F. Gómez-Rodríguez, M. Rivas, G. Jiménez, A. Civit, "Address-Event based Platform for Bio-inspired Spiking Systems", SPIE. Optical Engineering,.Pág.. 2007.

[KALAYJIAN01] Z. Kalayjian and A. G. Andreou, "Asynchronous communication of 2D Motion Information using Winner-Takes-All Arbitration", International Journal Analog Circuits Signal Processing,vol. 13.Pág.103-109. 1997.

[KRAMER01] J. Kramer, "An On/Off Transient Imager with Event-Driven, Asynchronous Read-Out", Proc. of the IEEE Int. Symp. onCirc. and Syst. (ISCAS02),.Pág.165-168.Phoenix, USA 2002.

[LAZZARO02] J. P. Lazzaro and J. Wawrzynek, "A Multi-Sender Asynchronous Extension to the Address-Event Protocol", 16th Conference on Advanced Research in VLSI,.Pág.158-169. 1995.

[LICHTSTEINER01] P. Lichtsteiner, T. Delbruck and J. Krammer, "Improved On/Off Temporally Differentiating Address-Event Imager", IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS,.Pág.211-214.Vancouver, Canada 2004.

[LICHTSTEINER02] P. Lichtsteiner, Tobi Delbruck, "64x64 Event-Driven Logarithmic Temporal Derivative Silicon Retina", Proc. IEEE Workshop on Charge-Coupled Devices and Advanced Image Sensors,.Pág.157-160.Nagano, Japan 2005.

[LICHTSTEINER03] P. Lichtsteiner, C. Posh, Tobi Delbruck, "A 128x128 120dB 15us Asynchronous Temporal Contrast Vision Sensor", IEEE Journal on Solid-State Circuits,Vol 43, No 2.Pág.566-576. 2008.

[LICHTSTEINER04] P. Lichtsteiner, C. Posch, T. Delbruck, "A 128x128 120dB 30mW Asynchronous Vision Sensor that Responds to Relative Intensity Change", IEEE ISSCC Digest of Technical Papers,.Pág.508-509.San Francisco 2006.

[LINARES01] Alejandro Linares Barranco, "Estudio y Evaluación de Interfaces para conexión de Sistemas Neuromórficos mediante Address-Event-Representation", 2003

[LINARES02] Alejandro Linares Barranco, Gabriel Jimenez Moreno, Antonio Abad Civit Balcells, Jose Luis Sevillano Ramos, Rafael Paz Vicente, "Software Generation of Address-Event-Representation for Interchip Images Communications", Proc. of the 28th Annual Conference of IEEE Industrial Electronics SocietyIEEE Press,.Pág.1915-1919.Sevilla, España 2002.

[LINARES03] A. Linares-Barranco, G. Jimenez-Moreno, A. Civit, B. Linares-Barranco, "On Algorithmic Rate-coded AER Generation", IEEE Transaction on Neural Network,Vol 17, No. 3.Pág.771-788. 2006.

[LINARES04] A. Linares-Barranco, M. Oster, D. Cascado, G. Jimenez, A. Civit y B. Linares-Barranco, "Inter-spike-intervals analysis of AER Poisson-like generator hardware", Neurocomputing Elsevier,.Pág.. 2007.

[LINARES05] A. Linares-Barranco, F. Gomez-Rodriguez, A. Jimenez-Fernandez, Tobi Delbruck, P. Lichtensteiner, "Using FPGA for visuo-motor control with a silicon retian and a humanoid robot", ISCAS,.Pág.. 2007.

[MAHOWALD01] Misha Mahowald, "VLSI Analogs of Neuronal Visual Processing: A Synthesis of Form and Function", 1992

[MAHOWALD02] M. Mahowald, "An Analog VLSI Stereoscopic Vision System", Kluwer Academic Publishers,.Pág.. 1994.

[MAHOWALD03] Misha A. Mahowald, "The Address-Event Representation CommuncationProtocol. AER 0.02", California Institute of Technology,.Pág.. 1993.

[MAHOWALD1] Misha Mahowald, "VLSI Analogs of Neuronal Visual Processing: A Synthesis of Form and Function", PhD Thesis,.Pág..California Institute of Technology. Pasadena, California 1992.

[MASSEY01] J.L. Massey, "Shift-Register Synthesis and BCH Decoding", IEEE Trans. on Information Theory,Vol IT-15.Pág.122-127. 1969.

[MEROLLA01] P. A. Merolla, J. V. Arthur, B. E. Shi, and K. A. Boahen, "Expandable Networks

for Neuromorphic Chips", IEEE Trans. onCircuits and Systems-I: Regular Papers,vol. 54, No. 2.Pág.301-311. 2007.

[MIRO01] Lourdes Miró Amarante, A. Jimenez, Alejandro Linares Barranco, Francisco Gomez Rodriguez, Rafael Paz Vicente, Gabriel Jimenez Moreno, Antonio Abad Civit Balcells, Rafael Serrano Gotarredona, "Lvds Serial Aer LINK Performance", IEEE International Symposium on Circuits and Systems,.Pág.1537-1540.New Orleans,Lusiana, USA 2007.1-4244-0921-7

[MIRO02] Lourdes Miró Amarante, Angel Francisco Jiménez Fernández, Alejandro Linares Barranco, Francisco Gomez Rodriguez, Rafael Paz Vicente, Gabriel Jimenez Moreno, Antonio Abad Civit Balcells, "A Lvds Serial Aer LINK", IEEE International Conference on Electronics, Circuits and Systems. Icecs ,.Pág.938-941.Nice, France 2006.1-4244-0395-2

[MITA01] R. Mita, G. Pulumbo, S. Pennisi y M. Poli, "Pseudorandom bit generator based on dynamic linear feedback topology", Electronics Letters,Vol 38, Nº 19.Pág.1097-1098. 2002.

[MORGADO01] Arturo Morgado Estévez, "Análisis y modelado de sistemas pulsantes bioinspirados basados en buses de altas prestaciones: Bus AER", 2004

[MORTARA01] A. Mortara and E.A. Vittoz, "A Communication Architecture Tailored for Analog VLSI Artificial Neural Networks: Intrinsic Performance and Limitations", IEEE Trans. Neural Networks,vol 5. No. 3.Pág.459-466. 1994.

[MORTARA02] A. Mortara, E.A. Vittoz and P. Venier, "A Communication Scheme for Analog VLSI Perceptive Systems", IEEE Journal of Solid-State Circuits,vol. 30, No. 6.Pág.660-669. 1995.

[MORTARA03] Alessandro Mortara, "A Pulsed Communication/Computation Framework forAnalog VLSI Perceptive Systems", Analog Integrated Circuits & Signal Processing, KluwerAcademic Publishers,Vol.13 No. 1.Pág.93-101.Boston 1997.

[MORTARA04] Alessandro Mortara, "A Pulsed Communication/Computation Framework forAnalog VLSI Perceptive Systems", In Tor Sverre Lande ed., Neuromorphic SystemsEngineering. Neural Networks in Silicon, Kluwer Academic Publishers,Vol. 9.Pág.201-215.Boston 1998.

[OTNES01] Hans Kristian Otnes Berge, "An AER Analog Silicon Cochlea Model using Pseudo Floating Gate Transconductors", 2005

[OZALEVLI01] E. Özalevli and C.M. Higgins, "Reconfigurable Biologically Inspired Visual Motion System Using Modular NeuromorphicVLSI Chips", IEEE Trans. Circ. Syst. I,Vol

52, No. 1.Pág.79-92. 2005.

[RICCI01] Gian Marco Ricci, "Selective Attention VLSI Tracking System", Semester Project,Courses and Seminars for the Ph.D. Program in Neuroscience,.Pág.. 2001.

[RIVAS01] Manuel Rivas Pérez, Francisco Gomez Rodriguez, Rafael Paz Vicente, Alejandro Linares Barranco, Saturnino Vicente Diaz, Daniel Cascado Caballero, "Tools for Address-Event-Representation Comunication Systems and Debugging.", Lecture Notes in Computer Science.696.Pag289-2962005

[RIVAS02] Manuel Rivas Pérez, Alejandro Linares Barranco, Francisco Gomez Rodriguez, Rafael Paz Vicente, Gabriel Jimenez Moreno, Antonio Abad Civit Balcells, "Aer Tools for Aer Bio-Inspired Spiking Systems", ..Pag341-3482005

[ROUSSELET01] G. A. Rousselet, M. Fabre-Thorpe, and S. J. Thorpe, "Parallel processing in high level categorisation of natural images", Nat. Neurosci., 5,.Pág.629-630. 2002.

[ROUSSELET02] G. A. Rousselet, S. J. Thorpe, and M. Fabre-Thorpe, "Processing of one, two or four natural scenes in humans: the limits ofparallelism", Vis. Res., 44,.Pág.877-894. 2004.

[RPAZ01] Rafael Paz Vicente, Alejandro Linares Barranco, Daniel Cascado Caballero, Saturnino Vicente Diaz, Gabriel Jimenez Moreno, Antonio Abad Civit Balcells, "Time-Recovering Pci-Aer Interface for Bio-Inspired Spiking Systems", Optical Engineering.5839.Pag111-1182005

[RPAZ02] Rafael Paz Vicente, Alejandro Linares Barranco, Daniel Cascado Caballero, Miguel Angel Rodriguez Jodar, Gabriel Jimenez Moreno, Antonio Abad Civit Balcells, Jose Luis Sevillano Ramos, "Pci-Aer Interface for Neuro-Inspired Spiking Systems", IEEE International Symposium on Circuiits and Systems. ISCAS 2006IEEE,.Pág.3161-3164. 2006.0-7803-9390-2

[RPAZ04] Rafael Paz-Vicente, Angel Jiménez-Fernández, Alejandro Linares-Barranco, Gabriel Jimenez Moreno, Francisco Gomez-Rodriguez, Lourdes Miró-Amarante, Anton Civit-Ballcels, "Image convolution using a probabilistic mapper on USB-AER board", ISCAS,.Pág..Seattle 2008.

[RUEDI01] P.F. Ruedi, et al., "A 128x128 pixel 120-dB dynamic-range vision-sensor chip for image contrast and orientation extraction", IEEE Journal of Solid-State Circuits,Vol. 38.Pág.2325-2333. 2003.

[SERRANO01] Rafael Serrano Gotarredona, Bernabe Linares Barranco, Teresa Serrano Gotarredona, Antonio Jose Acosta Jimenez, Alejandro Linares Barranco, Rafael Paz Vicente, Francisco Gomez Rodriguez, Gabriel Jimenez Moreno, Antonio Abad Civit Balcells, "High-Speed Image Processing With Aer-Based Components", IEEE

International Symposium on Circuiits and Systems. ISCAS 2006IEEE,.Pág.951-954. 2006.

[SERRANO02] Rafael Serrano Gotarredona, Matthias Oster, P. Lichtsteiner, Alejandro Linares Barranco, Rafael Paz Vicente, Francisco Gomez Rodriguez, H. Kolle Riis, T. Delbruck, S. C. Liu, S. Zahnd, A. M. Wathley, R. Douglas, P Halfliger, G. Jiménez Moreno, A Civit, Teresa Serrano Gotarredona, Antonio Jose Acosta Jimenez, Bernabe Linares Barranco, "Aer Building Blocks for Multi-Lawyer Multi-Chip Nueromorphic Vision Systems", Proceeding of the Neural Information Processing Systems Conference. NIPS 2005Neural Information Processing Systems Conference,.Pág.1217-1225.Vancouver, Canada 2005.

[SERRANO03] R. Serrano-Gotarredona, T. Serrano-Gotarredona, A. Acosta-Jiménez, C. Serrano-Gotarredona, J.A. Pérez-Carrasco, B. Linares-Barranco, A. Linares-Barranco, G. Jiménez-Moreno, A. Civit-Ballcels, "On Real-Time AER 2D Convolutions Hardware for Neuromorphics Spike Based Cortical Processing", IEEE Trans. Neural Networks,.Pág.. 2008.

[SERRANO03] R. Serrano-Gotarredona, T. Serrano-Gotarredona, A. Acosta-Jiménez, C. Serrano-Gotarredona, J.A. Pérez-Carrasco, B. Linares-Barranco, A. Linares-Barranco, G. Jiménez-Moreno, A. Civit-Ballcels, "On Real-Time AER 2D Convolutions Hardware for Neuromorphics Spike Based Cortical Processing", IEEE Trans. Neural Networks,.Pág.. 2008.

[SERRANO04] Teresa Serrano-Gotarredona, Andreas G. Andreou, Bernabé Linares-Barranco, "AER Image Filtering Architecture for Vision-Processing Systems", IEEE Transactions on Circuits and Systems. Fundamental Theory and Applications,Vol 49, No. 9.Pág.. 1999.

[SERRANO05] Rafael Serrano Gotarredona, "Arquitectura Aer Bio-Inspirada para Convolución de Imágenes en Tiempo Real", 2007

[SHEPHERD01] G. M. Shepherd, "The Synaptic Organization of the Brain", 1990

[SHOUSHUN01] C. Shoushun and A. Bermak, "A Low Power CMOS Imager based on Time-to-First-Spike Encoding and Fair AER", Proc. IEEE Int. Symp. Circ. Syst. (ISCAS),.Pág.5306-5309. 2005.

[SIVILOTTI01] M. Sivilotti, "Wiring Considerations in Analog VLSI Systems with Application to Field-Programmable Networks", 1991

[TELLURIDE01] , "Report to the National Science Foundation: Workshop on NeuromorphicEngeineering", 2001

[THORPE01] S.J. Thorpe, A. Delorme, R. VanRullen, "Spike-based strategies for rapid processing", Neuronal Networks,.Pág.14:715-725. 2001.

[THORPE02] S.J. Thorpe, "Ultra-rapid scene categorisation with a wave of spikes", BMCV,.Pág..

2002.

[THORPE03] S. J. Thorpe, K. R. Gegenfurtner, M. Fabre-Thorpe, and H. H. Bulthoff, "Detection of animals in natural images using farperipheral vision", Eur. J. Neurosci., 14,.Pág.869-876. 2001.

[TOBI01] Toby Delbrück (group leader), "Neuromorphic Hardware group at theInstitute for Neuroinformatics (INI)", ,.Pág..Zurich, Switzerland .

[VANRULLEN01] R. VanRullen y S. J. Thorpe, "Is it a bird? Is it a plane? Ultra-rapid visual characterization of natural and artificial objects", Perception, 30,.Pág.655-668. 2001.

[WHATLEY01] A. Whatley, R. Douglas, T. Delbrück, M. Fischer, Misha A. Mahowald, T. Matthews, "The Silicon Cortex Proyect: Address-Event Protocol", http://www.ini.unizh.ch/~amw/scx/aeprotocol.html,.Pág.. .

[WHATLEY02] Adrian M. Whatley, Rodney J. Douglas, "Silicon Cortex (SCX) Proyect", Institut für Neuroinformatik der Uni./ETH,.Pág.ETH Zürich. 1997.

11. Apéndices

11.1 Tarjeta USB-AER

La tarjeta USB-AER es una tarjeta basada en una FPGA para permitir probar diferentes tipos de dispositivos AER. Tiene un bus AER de entrada y otro de salida, de forma que puede usarse como dispositivo receptor AER, o como emisor AER, o ambas simultáneamente, introduciéndose en una cadena AER para realizar algún tipo de procesado. Dispone de 2MB de memoria RAM estática de 12 ns, estructurada en palabras de 32 bits. Una tarjeta SD/MMC puede almacenar el firmware que se cargara en la FPGA, y el contenido inicial de la memoria RAM, para permitir que el dispositivo pueda configurarse de forma autónoma sin necesidad de estar conectado a un PC. Adicionalmente, por medio de un puerto USB puede reconfigurarse la tarjeta desde un PC, con fines de depuración, e intercambiar datos con el PC. Usando esta funcionalidad puede usarse la tarjeta USB como puente.

El ancho de banda del puerto USB no permite el tratamiento individual de eventos. Con un ancho de banda de 11 Mbps, si cada evento ocupase un total de 32 bits (16 bits para el evento, 16 bits para la información temporal), el numero total de eventos por segundo seria de 340 Keventos/s. El valor exacto depende del tipo de transferencia que se use, el overhead del protocolo, etc. Sin embargo, considerándola como cota superior queda patente que es insuficiente para aplicaciones AER de vídeo. Una imagen de 64x64 con 256 niveles de gris, en el peor de los casos harán falta 1048 Keventos, 524 Keventos de promedio, para una única imagen. Sin embargo, si en lugar de enviar los eventos de manera independiente por el bus USB, enviamos las imágenes como mapas de bits, el ancho de banda necesario se reduce considerablemente. La misma imagen de 64x64x256 necesitaría 4096 eventos (4Keventos). Incluso para vídeo en tiempo real, para 25 fps se necesitaría 100Keventos/s. Por esto, los diferentes métodos tanto de reconstrucción de imágenes AER, como de generación de eventos a partir de imágenes, se puede usar la tarjeta USB, que permite conectarse a ordenadores portátiles, a diferencia de la PCI, que requiere instalarla previamente en el interior de un ordenador de sobremesa.

Dado que la tarjeta se basa en una FPGA, el sistema es reconfigurable, cargando diferentes firmwares diferentes para la FPGA, que permitirían diferentes usos para el mismo hardware. Inicialmente fue diseñada para actuar como un mapeador de eventos, de manera que para cada

evento que recibiera por su puerto de entrada, buscara en unas tablas en RAM el evento o eventos equivalentes (un evento de entrada puede dar lugar a ninguno, uno o varios eventos de salida, diferentes o no) y los enviará por el puerto de salida.

Sin embargo, modificando el firmware de la FPGA, se puede utilizar como generador o receptor de eventos. Como generador de eventos puede situarse al comienzo de una cadena AER para generar patrones de eventos de manera similar a como lo haría una retina AER, a partir de imágenes fijas que se irían cargando desde el USB, o tarjeta MMC, o secuencias de vídeo (actualizando la imagen cargada de forma periódica a través del USB o de una cámara de vídeo añadiendo un convertidor ADC a la entrada). Como receptor puede colocarse al final de una cadena, o insertarse de forma transparente dentro de una cadena para visualizar resultados intermedios.

En la siguiente figura puede verse una fotografía de la placa USB2AER, con los principales componentes resaltados:

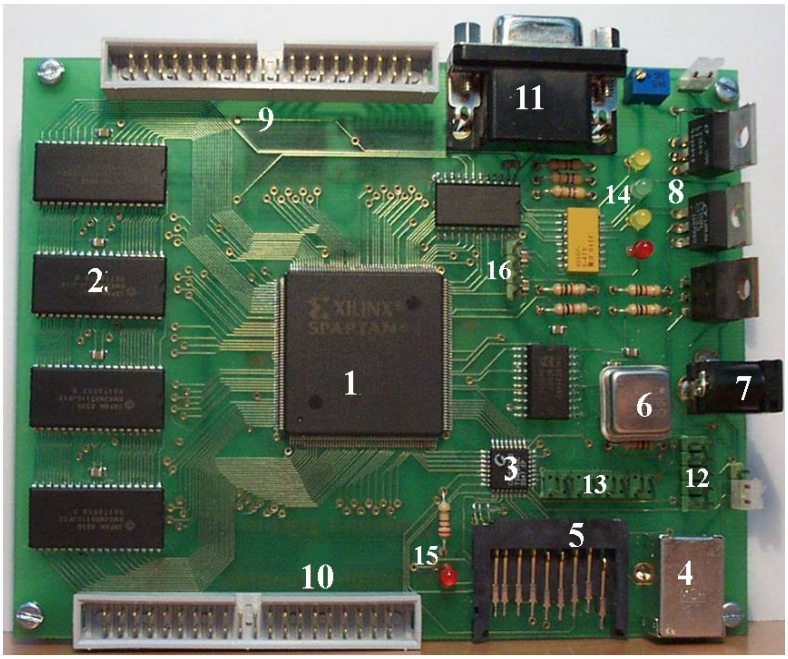

Ilustración 11.1: Tarjeta USB-AER

En la fotografía aparecen identificados los diferentes componentes que componen la tarjeta USB-AER:

1. FPGA Spartan II 200
2. Memoria SRAM. 512Kx32bits
3. Microcontrolador Cygnal C8051F320.

4. Conector USB.
5. Conector para tarjeta de memoria MMC/SD
6. Reloj de cuarzo de 50 Mhz
7. Conector de alimentación (6V-12V)
8. Fuente de alimentación (2.5V, 3.3V y 5V)
9. Puerto AER de entrada
10. Puerto AER de salida.
11. Puerto VGA de salida (sin DAC).
12. Puerto de programación del Cygnal. (Actualización firmware)
13. Conector JTAG de la FPGA. (Depuración solo).
14. Diodos LED de FPGA (led de Done + 3 de usuario)
15. Diodo Cygnal.
16. Pin reset FPGA y Cygnal (Debug).

11.1.1 Estructura de la tarjeta USB-AER

La tarjeta USB-AER se basa fundamentalmente en una FPGA Xilinx Spartan 2, conectada a 4 chips de memoria RAM conformando un total de 512Kx32 bits, con 4 señales de escritura independiente de forma que pueden escribirse palabras de 32 bits, o bytes independiente. Esta memoria puede utilizarse para almacenar eventos, eventos junto a su time-stamp (16 bits de evento y 16 bits de time-stamp), tablas de mapeado en el caso de funcionar como remapper, o cualquier información en general que el diseño a prueba necesite. Con un reloj de 50Mhz, es posible realizar un acceso cada ciclo a estas memorias (tiempo de acceso de 12 ns).

Por otra parte, un microcontrolador se encarga de cargar el contenido del firmware de la FPGA simulando una eeprom serie conectada a la FPGA. El firmware podrá cargarse utilizando un fichero almacenado en la tarjeta MMC o bien por el puerto USB. El microcontrolador Cygnal C8051F320 basado en un núcleo 8051 con un interfaz USB 1.1 Full Speed, se encarga del interfaz con la MMC (protocolo SPI) y con un PC a través del puerto USB. Adicionalmente, un bus asíncrono compuesto por 8 líneas de datos, además de algunas señales adicionales de control, interconectan la FPGA con el microcontrolador para el intercambio de datos. Por medio de este bus, el microcontrolador, una vez cargado el firmware en la FPGA, cargará el contenido de la RAM, desde la MMC/SD o el USB. El firmware cargado ira recibiendo los datos desde el microcontrolador, y almacenarlo en la RAM externa de la placa, en la interna de la FPGA o realizar la operación que se necesite. Las señales por las cuales se carga el firmware en la FPGA son distintas a las que se utilizan posteriormente para transferir datos.

Un reloj de cuarzo de 50 Mhz genera una señal de reloj estable para la FPGA, mientras que el microcontrolador utiliza un oscilador interno de 12 Mhz, que mediante un PLL interno se multiplica por 4 para conseguir 48Mhz para el USB, y 24 Mhz para el procesador 8051. La interconexión entre el microcontrolador y la FPGA se sincroniza por medio de dos señales: strobe y busy. El microcontrolador activa strobe para enviar o recibir datos, mientras que la FPGA puede activar la señal busy para congelar el envió de datos, aunque dado la mayor velocidad de la FPGA sobre el microprocesador esto raramente es necesario.

Tanto el puerto AER de entrada como el de salida tienen unos buffers alimentados a 5V o a 3.3V para ser compatible con dispositivos AER a ambas tensiones, aunque la mayoría de los

dispositivos AER actuales trabajan ya con 3.3V.

En el siguiente diagrama puede observarse un esquema de bloques de la tarjeta:

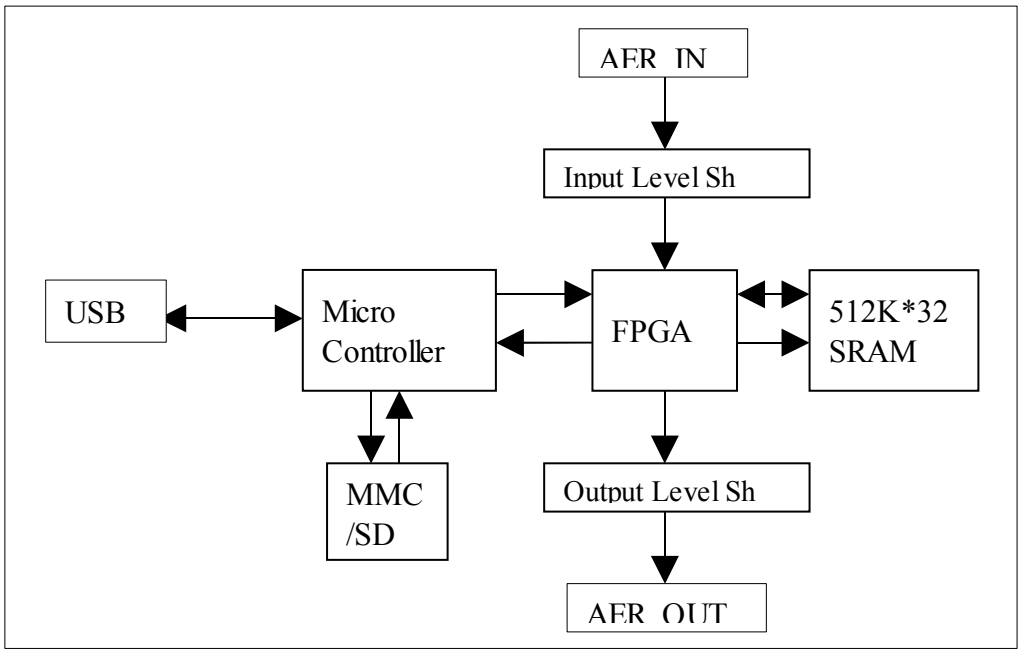

Ilustración 11.2: Diagrama de bloques tarjeta USB-AER

11.1.2 Proceso de arranque de la tarjeta USB

Cuando la tarjeta USB se alimenta, la FPGA se encuentra sin programar. El código contenido en el microcontrolador se encargará de buscar el firmware que debe cargarse en la FPGA. Para ello buscara en la tarjeta SD/MMC un fichero llamado "firmware.bin". En caso de encontrarlo, empezará la programación de la FPGA. Una vez se ha programado la FPGA, el microcontrolador buscará un fichero llamado "ram.bin" y en caso de encontrarlo, lo enviará a la FPGA usando el bus de 8 bits según un protocolo descrito más adelante.

Tanto si se ha encontrado la tarjeta SD/MMC como si no, a continuación se entra en el bucle principal en el cual se esperan comandos enviados a través del bus USB. Un led conectado al microcontrolador se encenderá durante las transferencias de datos. En este bucle se espera a recibir un paquete de control a través del USB, y proceder según el comando que se reciba. De esta forma se puede reprogramar en cualquier momento la FPGA y cambiar su funcionamiento de forma dinámica,

11.1.3 Protocolo USB

Cuando la tarjeta USB-AER se encuentra conectada a un PC por medio del USB, desde el PC se podrá modificar el firmware contenido en la FPGA, además de enviar o recibir datos desde esta. Puesto que las aplicaciones que pueden cargarse en la FPGA son muy diversas, el protocolo debe ser lo bastante genérico para no tener que se modificado al desarrollar nuevos diseños.

El protocolo esta orientado en forma de paquetes o comandos. Antes de cualquier transferencia hay que enviar desde el PC un paquete de control de 64 bytes al microcontrolador (escribir en el dispositivo). El formato de dicho paquete es el siguiente:

Offset	Bytes	Campo	Descripción
00h	3	'ATC'	"Magic Number" para identificar un paquete de control.
03h	1	Command	Comando.
04h	4	Longitud	Numero de bytes que se van a transferir
08h	16	Parámetros	16 bytes de parámetros que se enviaran a la FPGA.

Tabla 11.1: Descripción de trama de comandos USB-AER

Actualmente hay definidos 3 comandos para el control de la FPGA. Podría haber comandos para la comunicación entre el PC y el microcontrolador que no implique a la FPGA si fuera necesario. Los comandos actuales son los siguientes

Código	Comando	Descripción
00h	Update Firmware	Se utiliza para subir un nuevo firmware a la FPGA
01h	Write FPGA	Se utiliza para enviar datos a la FPGA.
02h	Read FPGA	Se utiliza para recibir datos de la FPGA.
03h	SetDevName	Se utiliza para darle un nombre al dispositivo.

Tabla 11.2: Códigos de los diferentes comandos USB-AER

El campo longitud indica el tamaño de los datos que se van a transferir. En el caso del comando '00' indica el tamaño del firmware que se va a transferir. En los comandos '01' y '02' el tamaño de los datos que se van a transferir.

A continuación se transfieren 16 bytes que serán parámetros del comando que se le enviarán a la FPGA. Hay una señal de control entre el microcontrolador y la FPGA que se activará para distinguir si los datos que se transfieren son parámetros o datos. El significado es dependiente del diseño cargado en la FPGA. Si no se necesitan parámetros pueden ser ignorados por la FPGA,

aunque el microcontrolador siempre los reenviará a la FPGA. En el caso del remapper,los 4 bytes de estos parámetros indican la dirección de comienzo para lectura o escritura. Estos parámetros siempre son enviados por el PC a la FPGA, independientemente de si se trata de un comando de lectura o de escritura.

Una vez transferido este paquete de control, dependiendo de la dirección del comando especificado, se transferirán desde el PC a la FPGA o viceversa tantos datos como se indicaron en el campo longitud. Estos datos serán troceados y enviados por el usb en fragmentos de 64 bytes.

11.1.4 Protocolo Cygnal-FPGA

El microcontrolador Cygnal se comunica con la FPGA utilizando un bus de 8 bits. Adicionalmente a estas 8 líneas de datos, hay cinco lineas de control:

Nombre	Dirección	Descripción
Data[8]	Bidir.	Lineas de datos
DataValid	uP a FPGA	A uno indica dato válido.
Program	uP a FPGA	A uno indica se van a intercambiar datos. A cero normalmente.
Control	uP a FPGA	A uno indica paquete de control. A cero datos.
ReadWrite	uP a FPGA	A uno indica que el uP escribe en la FPGA. A cero lee de la FPGA.
Busy	FPGA a uP	A uno cuando la FPGA no pueda transferir datos.

Tabla 11.3: Líneas de protocolo interconexión microcontrolador-FPGA

El protocolo entre la FPGA y el microcontrolador se ha intentado hacer lo suficientemente general para que permita ser válido independientemente de las características del diseño contenido en la FPGA, de forma que pueda modificarse fácilmente el firmware de la FPGA sin tener que reprogramar el microcontrolador, y permita la intercomunicación en ambos sentidos con una aplicación que se ejecute en el PC e intercambie datos usando el bus USB. Este protocolo esta implementado en el microcontrolador, y debe ser tenido en cuenta al diseñar algún dispositivo sobre esta tarjeta. Estas señales sin embargo podrían cambiar de dirección y funcionalidad si fuese necesario modificando el código del microcontrolador.

Durante la operación normal, la señal Program esta desactivada, el microcontrolador no va a transferir datos, por lo que el resto de las señales no tienen que tenerse en cuenta. Cuando llega un paquete de control con un comando de transferencia de datos, el microcontrolador activa la señal Program indicando el comienzo de una transferencia. La señal de Control también se activa para indicar que primero va a transferirse los parámetros del comando. Estos parámetros siempre irán

desde el microcontrolador a la FPGA, independientemente del valor de ReadWrite. Una vez se ha terminado de transferir los parámetros, se desactiva control y se procederá a la transferencia de datos. Durante este periodo, la señal ReadWrite indica la dirección de los datos.

La señal DataValid se utiliza como reloj para la transferencia de datos y es activada por el microcontrolador. Si la FPGA necesitara detener al microprocesador puede activar la señal Busy y el microcontrolador quedara en un bucle a la espera a que se desactive antes de continuar.

11.2 La tarjeta PCI-AER

La tarjeta PCI-AER [RPAZ01][RPAZ02]es un dispositivo que hace de puente entre los buses AER y PCI. Está basado fundamentalmente en una FPGA Xilinx Spartan II por lo que su funcionalidad puede cambiar en función de las necesidades de cada aplicación en concreto. Además de la FPGA, encargada del interfaz con el bus PCI, emisión y recepción de eventos AER, cuenta con otros componentes adicionales. El diseño en VHDL de la FPGA puede almacenarse en una memoria EEPROM de Xilinx desde la que se cargará al alimentar la placa PCI. El contenido de esta EPROM y de la FPGA es accesible desde un puerto de programación JTAG. Una memoria RAM de 512 KB estructurada en palabras de 16 bits (256Kx16). En el diseño VHDL actual de la tarjeta PCI-AER se utiliza como FIFOS la memoria interna de la FPGA que es de mayor velocidad que la memoria RAM externa. Sin embargo puede ser útil en futuros diseños puesto que la cantidad de memoria RAM dentro de la FPGA es muy reducida.

Los puertos AER, tanto de entrada como de salida se conectan con los conectores AER por medio de unos buffers bidireccionales, dos de 8 bits por puerto, para cubrir los 16 bits de tamaño de evento. Por defecto están configurado como salida los dos buffers del bus AER de salida, y como entrada los del bus AER de entrada, aunque el la dirección de los buffers es controlada desde la FPGA, por lo que podría modificarse. Las líneas de protocolo (REQ y ACK) pasan por otro buffer diferente, y su dirección es fija y no puede modificarse.

La tensión de alimentación se selecciona mediante un jumper entre 3.3V y 5V para hacer compatible la tarjeta PCI con ambas tensiones del bus AER, aunque actualmente la mayoría de chips AER desarrollados por los otros socios del proyecto CAVIAR funcionan en la actualidad a 3.3V, manteniéndose los 5V como compatibilidad con dispositivos antiguos.

El regulador de tensión que se observa en la fotografía, proporciona 2.5V utilizados para la alimentación interna de la FPGA, al no estar presentes en el bus PCI. El reloj externo, proporciona una señal de reloj adicional de 50 Mhz, para ser utilizadas en la FPGA, además de los 33Mhz proporcionados por el bus PCI. Normalmente será suficiente con el reloj de la PCI, que puede además multiplicarse internamente en la FPGA utilizando un PLL interno de la FPGA, que permitirá obtener relojes de 66Mhz (2xCLKPCI) o 133Mhz (4xCLKPCI), que al derivar del reloj de

33Mhz irán en fase, por lo que es más recomendable su utilización a la del reloj externo, que presenta la dificultad de tratar con dos señales de reloj independientes.

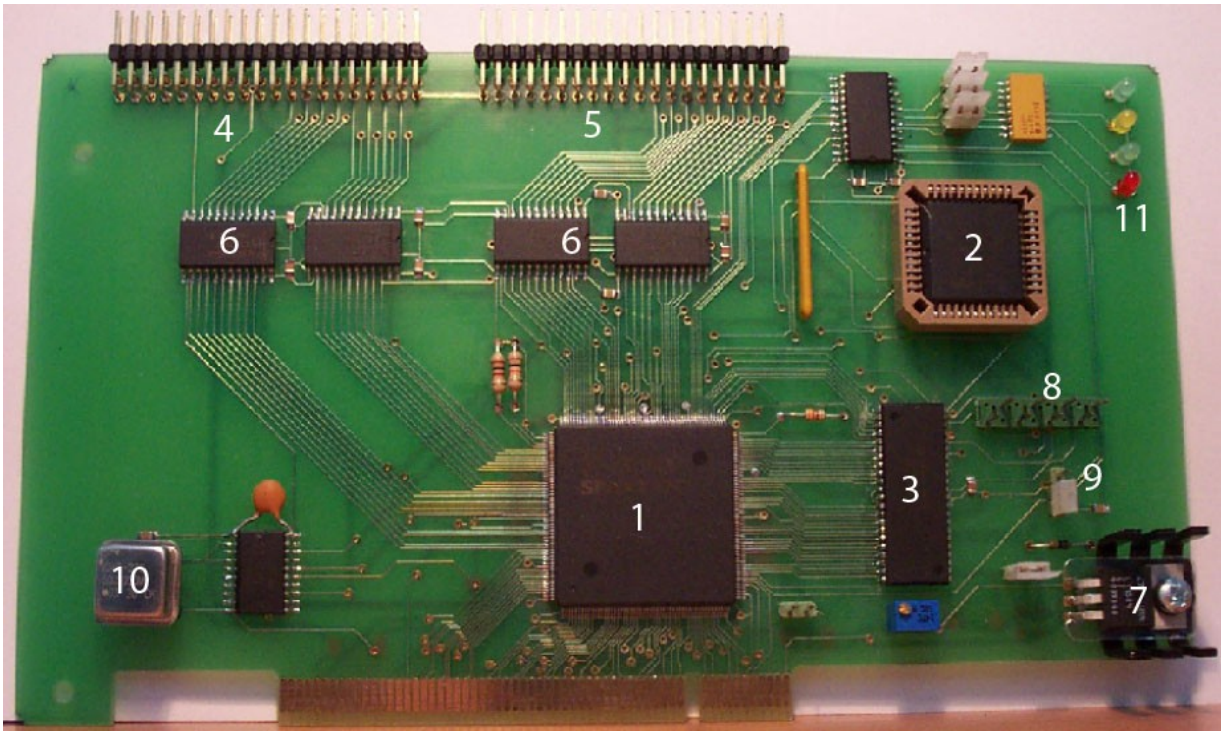

Ilustración 11.3: Tarjeta PCI-AER

En la fotografía aparecen identificados los diferentes componentes que componen la tarjeta PCI-AER:

1. FPGA Spartan II 200
2. EEPROM firmware FPGA
3. Memoria SRAM. 256Kx16 bits
4. Puerto AER de entrada
5. Puerto AER de salida
6. Buffers AER de entrada/salida. Ajustan niveles tensión AER.
7. Regulador tensión ajustable, para 2.5V
8. Conector programación JTAG
9. Selector voltage AER 3.3v o 5v
10. Reloj de cuarzo de 50 Mhz
11. Leds de usuario y DONE.

11.2.1 Introducción

Si nos detenemos a mirar las especificaciones PCI, vemos que el bus PCI no es un bus sencillo como podían ser buses como el ISA, etc., en el sentido de que las señales que se describen tienen diversas funcionalidades en función de la fase de la transacción en la que nos encontremos. Incluso, varias señales pueden usarse en conjunto para indicar una determinada condición. Un dispositivo target, usando las señales DEVSEL#, TRDY# y STOP# puede indicar diferentes condiciones de terminación, y el master o iniciador debe ser capaz de reconocer estas condiciones para actuar en consecuencia, y bien reintentar el comando o devolver al driver una condición de error. Si un target necesita insertar más de un determinado número de estados de espera, que superen la latencia máxima especificada en el estándar, no podrá mantener ocupado el bus, sino que debe abortar la transacción actual y esperar a que el master vuelva a iniciarla para responder satisfactoriamente, etc. Se definen diferentes comandos, en función de la acción a realizar, y en función del comando utilizado, habrá señales cuyo significado variará. Por ejemplo, las líneas de dirección no toman el mismo sentido en un acceso a memoria, a entrada/salida, a configuración, en un reconocimiento de interrupción, etc.

Cuando se quiere diseñar un dispositivo que sea compatible con el bus PCI, hay que tener en cuenta que hay que abordar dos cuestiones diferentes, la primera sería diseñar el circuito de aplicación concreto que se necesite, pero además adecuar su interfaz para adaptarla a las especificaciones PCI, para que sea capaz de comunicarse con el microprocesador o con los demás dispositivos utilizando el bus PCI, siendo capaz de reconocer transacciones iniciadas por el master, o ser capaz de generar transacciones en el caso de ser un dispositivo master.

Para simplificar la tarea de interconexión con el bus PCI, existen diferentes tipos de soluciones, ofrecidas por diversos fabricantes. Para ello, encapsulan las funciones de interconexión con el bus PCI, en una caja negra, que por un lado se conecta con el bus PCI, y por otro, ofrecen un interfaz más sencillo al cual interconectar el circuito que se esta diseñando. De esta forma, se delegan las tareas de interconexión con el bus PCI en esta "caja negra", cuyo interfaz que ofrece para la aplicación de usuario es mucho más sencilla.

Para ello existen principalmente dos soluciones. Una de ella consiste en un circuito integrado

al que se conecta el resto del circuito, y al bus PCI. Otra solución consiste en ofrecer un circuito, pero en lugar de en forma de circuito integrado, como entidad en algún lenguaje de descripción de hardware como el VHDL, ya sea en forma de código fuente VHDL que se une al resto del código VHDL y se sintetiza de forma conjunta, para grabar sobre la FPGA en la que se basará el sistema, o bien, como entidad presintetizada para una FPGA concreta, que aunque oculta su implementación, conociendo su interfaz podemos incorporar al proyecto.

Empresas como QuickLogic, PLX, etc., manufacturan familias de circuitos integrados para interconexión PCI, que soportan funcionamiento bien como target, o como master/target, en buses de 32/64 bits, y 33/66 Mhz según la versión de circuito integrado particular. Por otra parte empresas como Xilinx, PLDA, que ofrecen una solución "software" en forma de core para incorporar al proyecto VHDL que se este desarrollando para sintetizarlos juntos.

En este caso particular, el diseño consiste en un core VHDL que implementa una máquina de estados finita, encargada de la interconexión con el bus PCI por una parte, y que por otra parte ofrece un interfaz más sencillo al que conectar la aplicación de usuario que se quiera implementar.

11.2.2 Core PCI

El core PCI consiste en una entidad definida en VHDL que realiza la tarea de interconexión con el bus PCI. El core PCI se basa fundamentalmente en una máquina de estados finitos, que dividirá una transacción en diversos estados o etapas, y en función del estado en que se encuentre, actuará siguiendo las especificaciones del bus PCI. En la parte de la aplicación de usuario, el core ofrece un interfaz mucho más sencillo, pero sobre todo, que el significado de las señales no cambia como ocurre en el bus PCI, por lo que la aplicación cliente, únicamente tiene que atender al estado de estas señales, sin tener que llevar internamente el estado de la transacción en que se encuentre.

Como hemos visto en la descripción del estándar PCI, la especificación 2.2 contempla dos anchuras para datos, de 32 y 64 bits, con una frecuencia de reloj de 33 o 66 Mhz. Inicialmente el core PCI debía soportar las cuatro versiones del estándar en previsión de futuras necesidades. El PCI-X, de similares características al PCI pero con una velocidad de reloj de 133 Mhz, 266 Mhz e incluso 533 MHz con 16, 32 y 64 bits se vislumbraba como sucesor del PCI al alcanzarse anchos de

banda por encima del gigabit. Sin embargo la fuerte popularización de buses series de alta velocidad (firewire, usb 2.0) y posteriormente el bus PCI-espress que se presenta como sucesor serie al PCI, ha dejado obsoletos tanto al PCI-X como las versiones de 64 bits y/o 66 Mhz del PCI 2.0 que no han llegado a popularizarse más allá de algunos ordenadores de gama alta (servidores, etc.) y de propósito industrial.

Debido a que únicamente se ha popularizado la versión "domestica" del bus PCI, a 33 Mhz y 32 bits, finalmente el core PCI se ha centrado en esta versión, sin embargo podría modificarse para extenderse a 64 bits y 66 Mhz. La diferencia entre un bus a 33 Mhz o a 66 Mhz es únicamente una diferencia de tecnológica, pero no existe ningún tipo de diferencia a nivel de protocolo o de señales. Tan sólo la señal M66EN le indicaría al dispositivo que se encuentra en un bus a 33 Mhz o a 66 Mhz, por lo que podría usarse para conocer si deben insertarse estados de espera adicionales en el caso de que la aplicación de usuario concreta no fuese capaz de suministrar datos a la velocidad necesaria. Una tarjeta que sólo pudiese operar a 33 Mhz, usaría esta línea, conectándola de forma permanente a nivel lógico 0 para indicar su imposibilidad de operar a frecuencias superiores, en cuyo caso, el bus completo reduciría su frecuencia de reloj. Una tarjeta no puede cambiar de forma dinámica esta línea para reducir la velocidad del bus cuando le fuese necesario.

Debido a que los procesadores actuales Intel basados en la familia X86 poseen un ancho de palabra de 32 bits, ninguna instrucción de E/S ni ningún acceso a memoria requeriría datos de 64 bits. La única posibilidad de aprovechar los 64 bits de datos serían bien que el puente agrupara dos transferencias de 32 bits en una única de 64 bits en los accesos a memoria (los accesos a memoria se hacen a líneas completas de caché, no a datos individuales), o bien que la tarjeta iniciara como master una transferencia hacia memoria de 64 bits.

Sin embargo, dado que estos buses apenas han visto la luz y su sucesor el PCI-Express empieza a popularizarse, carece de interés el abordar estos buses. El core PCI se centrará en PCI 33Mhz/32 bits. Para ello se ha diseñado una tarjeta PCI basada en una FPGA de la familia Xilinx. En una primera generación se utilizaron FPGAs de la familia VIRTEX, aunque debido el alto coste de la familia Virtex debido en parte a estarse abandonando la producción por parte de Xilinx en favor de Virtex 2E y Virtex 4, de mayor capacidad y velocidad (y menor precio), no recomendaban su utilización. Familias Virtex superiores no pueden utilizarse para interconectarse al bus PCI debido a que el bus PCI 33/32 trabaja con señales de 5V, mientras que las Virtex trabajan con niveles de 3.3V, a excepción de la Virtex 1.

En su lugar, en versión definitiva de la tarjeta PCI se utiliza una FPGA de la familia Spartan II de menor capacidad y velocidad que las Virtex. La FPGA Spartan II 200, tiene un tamaño suficiente para albergar al core PCI (8% CLBs ocupadas), junto a la aplicación de la tarjeta propiamente dicha. La menor velocidad de la familia Spartan II, obliga a un diseño cuidadoso para evitar caminos combinacionales demasiado largos que incumplirían los tiempos de Setup/Hold, provocando un mal funcionamiento del hardware. Operar a 66 Mhz queda fuera del alcance de esta FPGA, aunque el diseño del core sería completamente válido, necesitando sintetizarse para otra FPGA de velocidad superior. Como dato anecdótico, una FPGA Xilinx Virtex XCV300, la más pequeña de la familia Virtex cuesta alrededor de 300€ en un distribuidor oficial de Xilinx, mientras que la Spartan II XCS200, la más grande de las Spartan II, alrededor de 30€. El coste de fabricación de una tarjeta PCI es inferior a 100€ lo que la presenta como una opción atractiva por económica aunque rápida y eficiente para interconexión con ordenadores PC.

11.2.2.1 Características del core PCI

La utilidad del core reside en ocultar al desarrollador de un dispositivo PCI precisamente la parte de interconexión con el bus. Internamente implementa una máquina de estados finitos que se encarga de capturar las diferentes señales del bus, y en función del estado en que se encuentre, interpretarlas de una forma u otra. Como hemos vistos, en el bus PCI se definen tres espacios de direccionamiento diferente: memoria, entrada/salida y configuración. Durante el funcionamiento de un dispositivo, los accesos se realizan bien con accesos a memoria, bien con accesos a entrada/salida, mientras que el espacio de configuración se utiliza para la configuración de la tarjeta, normalmente por parte del sistema operativo, para ubicarla en posiciones de memoria o entrada/salida no utilizadas por otros dispositivos, pero no para el funcionamiento normal posterior de la tarjeta.

El core PCI, implementa los comandos de lectura/escritura para memoria, entrada/salida y configuración. Los registros de configuración, están implementados dentro de la entidad core-PCI, por lo que los accesos a estos registros son gestionados internamente por el core, mientras que los accesos a memoria o a entrada/salida son reenviados a la aplicación de usuario. De esta forma, la aplicación de usuario no tiene que preocuparse de la parte de configuración. El core-PCI

implementa los 6 registros BAR (Base Address Register), aunque cada aplicación puede utilizar únicamente aquellos que necesite. Cuando el core recibe una transacción desde el bus PCI, capturará la dirección de comienzo y la comparará con todos los BAR, teniendo en cuenta que algunos BAR definirán espacios de direccionamiento en memoria y otros en entrada salida, por lo que los comandos PCI utilizados serán diferentes. Si existe un acierto en la dirección, la máquina de estados pasaría a la fase de datos, activando una de las líneas de selección hacia la aplicación de usuario. La aplicación de usuario únicamente deberá comprobar si hay algún CS activado, si se trata de una operación de lectura o escritura para actuar en consecuencia.

Una primera versión del core PCI únicamente implementaba el comportamiento como target PCI. Posteriormente se ha ampliado la máquina de estados para implementar el comportamiento como master. Para mantener el interfaz con la aplicación VHDL específica lo más sencillo posible se ha mantenido en todo lo posible la compatibilidad con el core target. En el core PCI, además de los registro de configuración que son gestionados por el core de forma transparente a la aplicación VHDL, en el BAR6 se han añadido dos registros de control para el master. En estos registros se almacena la dirección hacia o desde la cual se realizará la transferencia, además de la longitud a transferir en palabras de 32 bits. En los accesos de la tarjeta como master, la dirección de comienzo debe estar alineada a 32 bits, y las transferencias son siempre múltiplos de 4 bytes para mantener el diseño lo más sencillo posible. Los accesos como master son interesantes para realizar transferencias de bloques grandes, por lo que esta limitación no es significativa.

Adicionalmente a las señales existentes para el target, se han añadido dos lineas adicionales mRW y mENABLE, que permiten a la aplicación VHDL conocer la dirección de la transferencia, y mediante la activación y desactivación de mENABLE controlar la transferencia de datos e interrumpirla si no hay datos para transmitir y posteriormente continuarla una vez haya más datos para transferir (o haya espacio en los buffers de la aplicación para recibir más datos). Durante el funcionamiento como target, la señal BUSY permite insertar estados de espera. Como master la señal BUSY sigue siendo util para insertar estados de espera (desactivaria TRDY# o IRDY# según correspondiese). La desactivación de mENABLE sin embargo, no inserta estados de espera sino que interrumpe la transferencia dejando el bus libre para que se use por otros dispositivos, y volviendolo a solicitar al arbitrador una vez haya datos bastantes para continuar con la transferencia, relanzándola con la posición de memoria siguiente a donde se interrumpió.

La señal mRW es diferente a la señal ReadWrite. La señal ReadWrite indica si se está accediendo a la tarjeta PCI como lectura o escritura en los accesos como target. Mientras que mRW

indica la dirección de las transferencia programada en los registros de control del master ubicados en el BAR6. Hay que tener en cuenta que durante una transferencia en modo master, si el master libera el bus temporalmente por falta de datos, el driver podría acceder a la tarjeta como target para por ejemplo cancelar la transferencia, o acceder a algun registro de estado, al margen de la transferencia master que podrá relanzarse en cualquier momento de forma automática.

El interfaz con el bus PCI del core-PCI es la siguiente en modo target sería la siguiente:

Señal	Dirección (core)	Significado
AD_Bus	**Entrada**	**Bus datos/direcciones**
CBE_Bus	**Entrada**	**Comando / Byte-enable**
PAR	**Entrada/salida**	**Paridad**
FRAME_N	**Entrada**	**Señal FRAME# (inicio transacción)**
TRDY_N	**Salida**	**Target Ready**
IRDY_N	**Entrada**	**Initiator Ready**
STOP_N	**Salida**	**STOP# (terminar transacción)**
DEVSEL_N	**Salida**	**Dispositivo seleccionado**
IDSEL	**Entrada**	**Selección ciclos configuración**
PERR_N	**Entrada/salida**	**Indicador error de paridad**
SERR_N	**Entrada/salida**	**Indicador error del sistema**
CLK	**Entrada**	**Señal de reloj**
RST_N	**Entrada**	**Señal de reset**

Tabla 11.4: Descripción señales bus PCI modo target

Para añadir las funcionalidades de master, se han añadido dos señales adicionales de interfaz con el PCI, REQ# y GNT# para la arbitración del bus. Además algunas señales cambian de dirección al operar como master (por ejemplo, IRDY# es salida como master, entrada target, etc). El interfaz con el PCI completo quedaria entonces de la siguiente forma:

Señal	Dirección (core)	Significado
AD_Bus	**Entrada/salida**	**Bus datos/direcciones (32bits)**
CBE_Bus	**Entrada/Salida**	**Comando / Byte-enable (4 señales). Entrada como target, salida como master**
PAR	**Entrada/salida**	**Paridad del bus AD[31..0] y CBE[3..0].**
FRAME_N	**Entrada/Salida**	**Señal FRAME# (inicio transacción). Entrada como target, salida como master.**
TRDY_N	**Salida/Entrada**	**Target Ready (Salida como target, entrada como master).**
IRDY_N	**Entrada/Salida**	**Initiator Ready. Entrada como target,**

		salida como master.
STOP_N	**Salida/Entrada**	**STOP# (terminar transacción). Salida como target, entrada como master.**
DEVSEL_N	**Salida/Entrada**	**Dispositivo seleccionado. Salida como target, entrada como master.**
IDSEL	**Entrada**	**Selección ciclos configuración.**
PERR_N	**Entrada/salida**	**Indicador error de paridad**
SERR_N	**Entrada/salida**	**Indicador error del sistema**
CLK	**Entrada**	**Señal de reloj**
RST_N	**Entrada**	**Señal de reset**
REQ_N	**Salida**	**Solicitar mastering del bus**
GNT_N	**Entrada**	**Concesión del bus**

Tabla 11.5: Descripción señales bus PCI modo master

Estas señales son las definidas en las especificaciones PCI y su funcionamiento se ha visto en el capítulo anterior. De cara a la aplicación de usuario, se ha intentado ofrecer un interfaz que limite lo menos posible la flexibilidad ofrecida por el bus PCI, oculte detalles carentes de relevancia para la aplicación que va a diseñarse, pero que a la vez ofrezca un interfaz sencillo de utilización y sobre todo, que permita realizar aplicaciones de usuario de forma sencilla.

Teniendo en cuenta estas consideraciones, se propone un interfaz del core-PCI que, por una parte, oculta todo el proceso de decodificación (y obviamente, todo lo relativo a la configuración de la tarjeta PCI), ofreciendo únicamente 6 señales de habilitación, de forma que la aplicación de usuario no tendría que preocuparse de en que espacio de direccionamiento se ubica, o del comando utilizado (la señal ReadWrite indica si es lectura o escritura), aunque por otra parte permite sin más que activar la señal de control correspondiente, insertar estados de espera, o terminar la transacción en curso (con o sin datos, provocando que el core-PCI activara la señal STOP# forzando la terminación de la transacción actual).

Otras señales, como ADDRESS, permiten en el caso de ser necesario, utilizar los bits bajos para conocer a que registro o a que posición de memoria se esta accediendo dentro de nuestro dispositivo.

El interfaz que se propone entre el core-PCI y la aplicación de usuario sería el siguiente. El sentido de las transferencias se indica desde el punto de vista del core-PCI:

Señal	Dirección (core)	Significado
ADDRESS	Entrada	Dirección de la transacción (32 bits)
DATA	Entrada/salida	Bus de datos para entrada/salida (64 bits)
ReadWrite	Entrada	Indica si es una operación de lectura (1) o de escritura (0).
INICIA	Entrada	Indica al core-master que solicite el bus para iniciar una transacción
FINALIZA	Salida	Indica que ha concluido la transacción (target-abort, etc)
LastTX	Entrada	Indica que el siguiente dato será el ultimo a transferir. Solo target.
BUSY	Entrada	Insertar estados de espera
DENABLE	Salida	Señal de habilitación de datos
SCLK	Salida	Reloj del sistema
RESET	Salida	Reset del sistema
mRW	Salida	Dirección de la transferencia master programada en registros de control de DMA.
mENABLE	Entrada	Regula las transferencias en modo master

Tabla 11.6: Descripción señales core PCI

El bus ADDRESS se utiliza para facilitarle a la aplicación de usuario la dirección a la que se está accediendo. En transacciones en ráfaga se iría incrementando en múltiplos de 4 por cada pulso de transferencia de datos. Aunque el core-PCI activa la señal DEVSEL# cuando la comparación de la dirección de comienzo de la transacción coincide con la configurada en alguno de los registros BAR, activando la señal CSn correspondiente para indicarle a la aplicación que esta siendo accedida, la parte baja de este bus ADDRESS puede usarse para conocer dentro de la zona de memoria definida en el BAR, a que posición exacta se esta accediendo.

El bus DATA es un bus de datos bidireccional, cuya dirección depende de si nos encontramos en una operación de lectura o escritura. La aplicación de usuario debe usar la línea ReadWrite para conocer la dirección de la transacción en curso, y volcar datos sobre este bus, o bien colocarlo en alta impedancia y leer los datos procedentes desde el bus PCI.

El significado de las líneas BE (Byte Enable) es similar a las líneas C/BE del bus PCI, y deben utilizarse para conocer el tamaño de los datos transferidos.

Cada línea CSn indica si la dirección de la transacción, y su tipo (entrada/salida o memoria) coincide con la configurada en cada uno de los BAR. Si CSn(0) se activa, indica que la dirección de la transacción coincide con la indicada en el BAR0, y es del mismo tipo. Si el bit 0 del BAR0 indicara que es una zona ubicada en el espacio de entrada/salida, sólo se activaría CSn(0) si la

transacción corresponde a un comando a entrada/salida, ya sea de lectura o de escritura. Como en un BAR no se define una única posición, sino un rango de direcciones, la comparación se realiza en función de una mascara de bits que indica que bits de la dirección se utilizan para decodificar el dispositivo, y cuales para direccionar dentro de él. El BAR6 se activa cuando el arbitrador nos concede el bus para su uso como master. Su funcionamiento es similar a cualquier otro BAR.

La señal ReadWrite indica la dirección de una transacción. A nivel alto indica una operación de lectura, y a nivel bajo una operación de escritura.

Las señales TWithData y TWithoutData se utilizan para indicar la master la terminación de la transacción actual. La diferencia entre una u otra es que si activamos TWithData, la transacción actual terminará después de completar la fase de datos actual, al activarse TRDY#, mientras que si activamos TWithoutData, la señal TRDY# se desactivará (independientemente del valor de BUSY), por lo que la fase de datos actual no se completa y se termina sin transferirse el ultimo dato. Si TWithoutData se activa en la primera fase de datos, sin que esta llegue a completarse, tendremos una condición de "Retry". Puede usarse cuando la aplicación de usuario no vaya a ser capaz de cumplir los tiempos de latencia máximo, y el master volverá a intentar el comando más tarde.

La señal Busy es una señal de entrada al core (salida de la aplicación de usuario) que puede ser utilizada por esta última para insertar estados de espera adicionales. La aplicación de usuario deberá cuidar no mantener esta señal activa demasiado tiempo de forma que sobrepasase la latencia máxima permitida por el bus PCI.

La señal DENABLE es una señal de habilitación de datos. Cada flanco de subida de la señal de reloj en las que DENABLE está a nivel alto indica que se ha transferido un dato. Cuando se activa la señal Busy para insertar estados de espera, la señal DENABLE se desactiva, al igual que cuando el dispositivo no esta siendo accedido (o se está accediendo a sus registros de configuración.

La señal SCLK se corresponde con la señal de reloj del bus PCI. Puede ser usada por la aplicación de usuario por ejemplo, para activar la señal Busy sólo por un número de ciclos de reloj determinado.

Para su uso como master, se añaden dos señales mRW y mENABLE. La señal mRW contiene la dirección en que van a ocurrir las transferencias en modo master. Esta señal puede

utilizarse para en función de si es una lectura o una escritura, comprobar si la aplicación puede transferir datos, y activar mENABLE en caso afirmativo para solicitar el bus al arbitrador. La desactivación de mENABLE produce la finalización de la transacción actual y se libera el bus hasta que mENABLE vuelva a activarse, o se alcance la longitud de transferencia programada. Es interesante distinguir con la señal ReadWrite, que debe seguir siendo utilizada para decodificar el acceso por parte de la aplicación VHDL cuando se haya activado el CS correspondiente al BAR6.

11.2.3 Máquina de estados finitos (FSM). Target.

El corazón del core-PCI reside en una máquina de estados finitos donde se modelan las diferentes fases o estados por las que puede pasar un dispositivo PCI. En primer lugar vamos a describir el funcionamiento como target, obviando la parte del master para mayor claridad. Más adelante se indicara el esquema de la máquina de estados resultante completa.

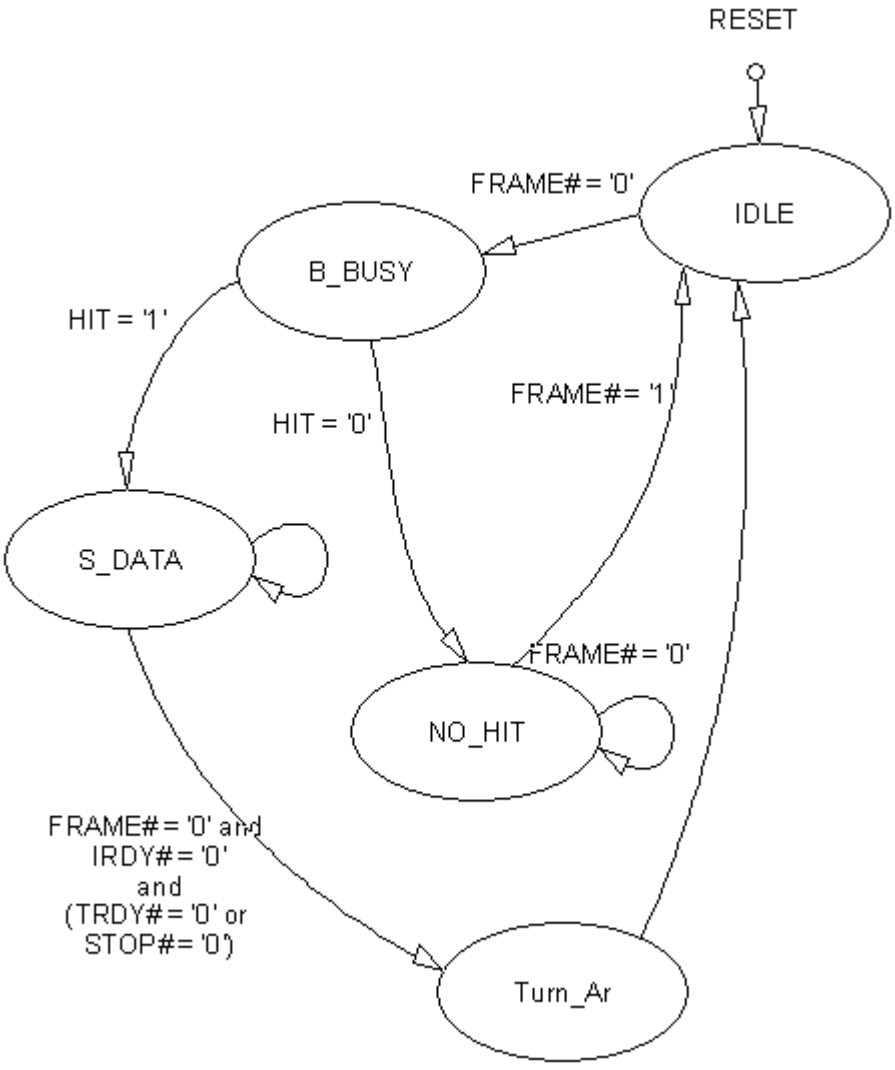

Ilustración 11.4: Diagrama estados FSM core PCI (target)

Para modelar el comportamiento de un dispositivo target se han utilizado cinco estados. El estado IDLE indica que el bus esta en reposo, por lo que el core-PCI tendrá todas sus salidas en alta impedancia en ese momento:

```
AD_Bus <= (others => 'Z');
TRDY_N<= 'Z';
DEVSEL_N <='Z';
PERR_N <='Z';
SERR_N <= 'Z';
STOP_N <='Z';
DATA <= (others => 'Z');
```

Cuando algún dispositivo activa la señal FRAME# indicando el comienzo de una transacción, se pasa al estado B_Busy, indicando que el bus pasa a estar ocupado. En este estado se decodifica la dirección de comienzo de la transacción, activándose la señal HIT en caso de que se determine que la tarjeta es el dispositivo de destino de la transacción.

La activación de HIT no depende únicamente de la dirección indicada en la fase de direccionamiento, sino también del comando utilizado,y de otras señales adicionales como IDSEL. Por ejemplo, en un acceso de entrada/salida o de memoria, la dirección especificada en la fase de direcciones debe coincidir con alguno de los registros BAR, pero además el comando debe coincidir con el tipo de dirección que codifica el registro BAR: si el bit 0 del BAR esta a 1, indica que es una dirección de entrda/salida, por lo que el comando utilizado debe ser una lectura o una escritura pero de entrada/salida. Si el bit 0 del BAR fuese 0, indicando una dirección de memoria, el comando debería ser cualquiera de los existentes para acceso en memoria.

Si por el contrario el comando utilizado es un comando de lectura o escritura en configuración, la dirección de la fase de direcciones no se utiliza para seleccionar nuestro dispositivo, sino que sería la señal IDSEL particular para este dispositivo.

Para activar HIT, se define que HIT se activa cuando se activa alguno de los chipselect correspondientes a los BAR, o cuando IDSEL esta activo con un comando de configuración. Esto puede verse en el código VHDL que lo describe:

```
HIT <= CSCFG or CS0 or CS1 or CS2 or CS3 or CS4 or CS5;

CSCFG <= LIDSEL and CFGCMD;

CS0 <= '1' when (((BAR0 and MASKAND0) = (LAddress and MASKAND0))
                    and ((BAR0(0)='1' and IOCMD ='1' and IOEnabled ='1')
                        or (BAR0(0)='0' and MEMCMD='1' and MemEnabled ='1'))
                    and BAR0Enabled = '1') else '0';
```

La activación de las señales CS1, CS2, CS3, CS4 y CS5 es similar a la de CS0 pero con BAR1, BAR2, BAR3, BAR4 y BAR5. El bit 0 de cada BAR indica si el correspondiente BAR se refiere a una dirección de entrada/salida o de memoria, por lo que hay que comprobar su valor y el tipo de comando utilizado antes de dar por válido un acceso.

Las señales IOEnabled y MemEnabled indican si están activados para este dispositivo los accesos a entrada/salida y los accesos a memoria. Esta condición se controla utilizando los bits 0 y 1 del registro de control. Normalmente se activan una vez han sido inicializados adecuadamente los registros BAR correspondientes:

```
IOEnabled <= ControlRegister(0);
MemEnabled <= ControlRegister(1);
```

En el estado B_Busy, se activa la señal DEVSEL# si nuestro dispositivo era el destino de la transacción (HIT activada) para hacerse cargo de la transacción, y pasará al estado S_Data. Si HIT no estaba activada, no se activa DEVSEL# y se pasa al estado NO_HIT hasta que termine la transacción en curso para pasar a IDLE. En caso de que la decodificación fuera afirmativa, al teminar de la transferencia se pasará al estado Turn_Ar (Turn arround) a la espera de que FRAME# se desactive indicando que ha terminado la transacción para pasar al estado de bus en reposo (IDLE). Como las especificaciones PCI permiten activar DEVSEL# en los tres ciclos de reloj siguientes a la activación de FRAME#, no era necesario DEVSEL# en este estado y podría haberse retrasado hasta el siguiente estado (S_Data).

El estado NO_HIT se diferencia del Turn_Ar en que en Turn_Ar, el core PCI antes de liberar el bus y pasar a IDLE, debe subir algunas señales de control antes de dejar el bus en alta impedancia, mientras que en el estado NO_HIT el bus esta en alta impedancia esperando a que concluya la transacción actual para pasar a IDLE y decodificar una nueva transacción.

En el estado S_Data se transferirá un dato en cada flanco de subida de la señal de reloj siempre que IRDY# y TRDY# estén ambas activas. Se llega a este estado a partir de B_Busy si la señal HIT estaba activada. Puesto que el dispositivo PCI ha sido diseccionado, las señales de salida al bus PCI dejarán de estar en alta impedancia y pasarán a ser controladas por el core PCI. Mientras la señal FRAME# este activada, la máquina de estados seguirá en este estado indicando que se encuentra en la fase de datos de una transacción. Cuando FRAME# se desactiva, se pasa al estado Turn_Ar.

El estado de Turn_Ar tiene una doble funcionalidad. Cuando no ha habido un acierto de direcciones, se utiliza para esperar a que termine la transacción, por lo que el bus debe permanecer en alta impedancia, y cuando FRAME# se desactive volver al estado IDLE. Por otra parte, cuando se ha estado transfiriendo datos en el estado S_Data, una vez se desactiva FRAME# indicando que ha terminado la transacción en curso, antes de volver al estado de reposo, en el que todas las líneas de salida están en alta impedancia, algunas líneas como TRDY#, STOP# y DEVSEL# tienen que devolverse a nivel alto de forma activa antes de dejarlas en alta impedancia.

A modo de ilustración del funcionamiento de la máquina de estados durante un acceso, podemos mostrar el valor del registro de estado, así como el siguiente estado proyectado, en función de los valores de las señales de entrada del bus PCI. Las transiciones de estados se producen en los flancos de subida de la señal de reloj, en los cuales, el registro de estado toma el valor del nuevo estado. El nuevo estado depende del estado actual, y del cumplimiento o no de las condiciones de transición mostradas en el diagrama de transición de estados anterior.

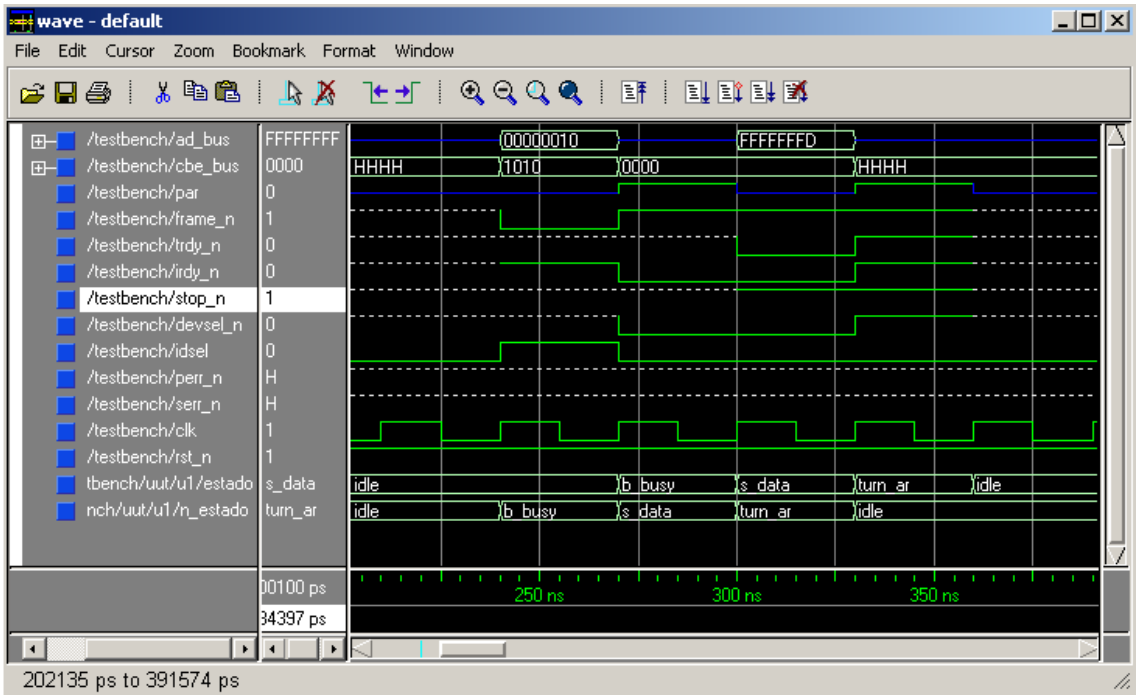

Ilustración 11.5: Cronograma ciclo bus PCI

Cuando el bus esta en reposo, la máquina de estados se encuentra en el estado IDLE, y mientras FRAME# siga desactivado permanecerá en este estado, por lo que el siguiente estado sigue siendo IDLE. Una vez se activa FRAME# indicando el comienzo de una transacción, se pasará al estado B_Busy (bus busy) en el siguiente flanco de subida de la señal de reloj. En este ejemplo,

como es una operación de lectura, durante B_Busy el bus esta en alta impedancia, sirviendo así de ciclo de turn-around para que el master libere el bus de datos/direcciones y lo tome el target. Si fuera una operación de escritura, al activar el target DEVSEL# en este estado, el master podría volcar el dato en este ciclo, aunque no se completará la fase de datos hasta el estado siguiente, S_Data, cuando TRDY# se activa.

En este caso, como únicamente habrá una fase de datos, la señal FRAME# se desactiva, para que el siguiente estado sea turn_ar, terminándose la transacción después de la primera fase de datos.

Los accesos a configuración son tratados íntegramente por el core, están compuestos por una única fase de datos, y no se inserta ningún estado de espera adicional. Los accesos bien a entrada/salida, bien a memoria, son decodificados por el core y reenviados a la aplicación de usuario. En accesos en ráfaga, el core se encarga de auto incrementar la dirección facilitada a la aplicación de usuario, de forma que en cada flanco de reloj, que tanto TRDY# como IRDY# estén activos se incrementará la dirección de destino en 4 bytes (transferencias de 32 bits).

En los accesos a memoria, hay dos métodos definidos en el estándar PCI, “linear incrementing” y “Cacheline Wrap mode”. El incremento linear es el descrito anteriormente. Para el modo “Cacheline Wrap”, habría que implementar el registro de configuración “Cache line size”. Sin embargo, este modo no es obligatorio implementarlo, por lo que en el caso de que el core detectara un acceso de ese modo, terminaría la transacción tras completar la primera fase de datos (Terminate with data) tal y como se recomienda en las especificaciones PCI. De esta forma, el master tendría que relanzar la transacción en la siguiente posición de memoria, por lo que la transferencia terminaría realizándose, aunque sin la optimización que aporta las transferencias en ráfaga.

Las Busy, TWithData y TWithoutData, utilizadas por la aplicación de usuario para insertar estados de espera o para terminar una transacción (con o sin datos) respectivamente, únicamente tienen funcionalidad en los ciclos de memoria y de entrada/salida. En los accesos a configuración, será el propio core PCI el que establezca los valores de TRDY# y STOP# independientemente del valor de estas señales.

11.2.4 Máquina de estados finitos (FSM). Master.

Una vez el core es capaz de responder a los comandos de configuración, y los accesos tanto de lectura como escritura, la operación como master es relativamente sencilla. Aunque el número de estados que se añaden es elevado, las señales de las que dependen las transiciones de las máquinas de estado son menos que en el comportamiento como target. La máquina de estados resultante sería la siguiente:

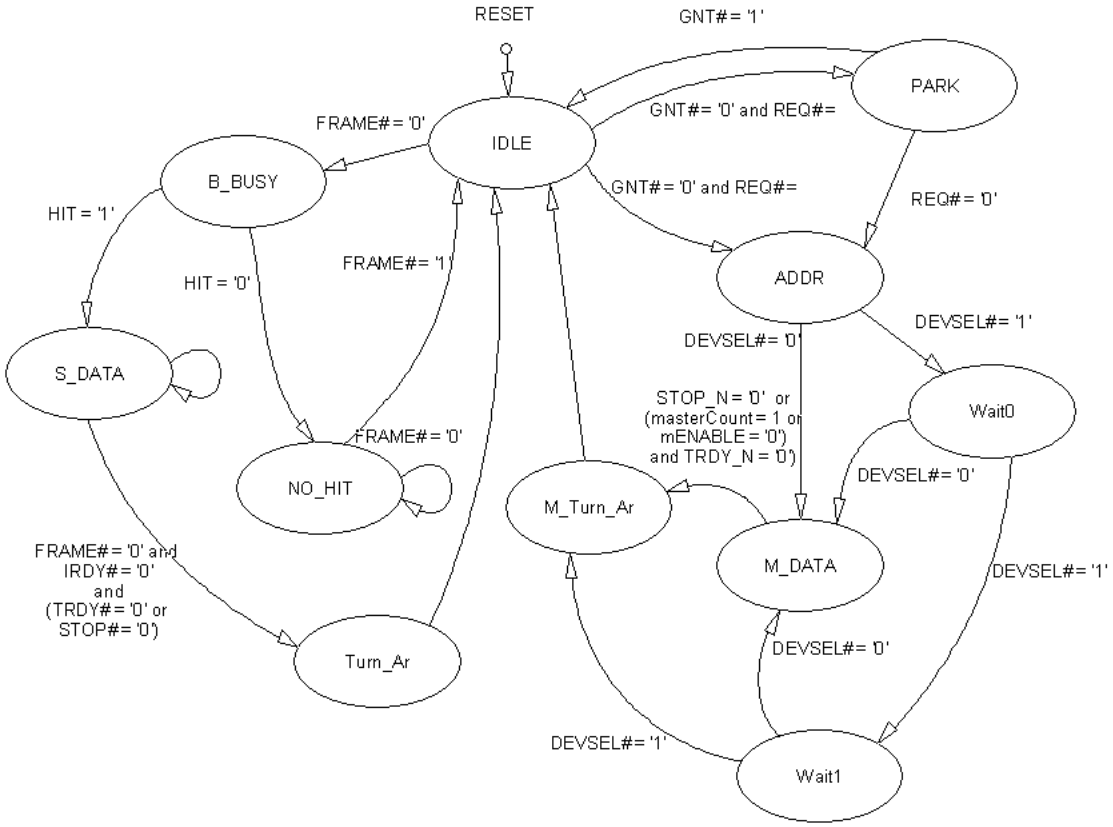

Ilustración 11.6: Diagrama estados FSM core PCI (master)

Registros de control del master

Para descargar a la aplicación VHDL de las tareas relacionadas con las transferencias master, el core PCI implementa un conjunto de dos registros de 32 bits que se utilizaran para configurar al core para que gestione las transferencias por DMA. Para ello hay que indicarle la dirección de comienzo, la longitud de la transferencia y la dirección de la transferencia. El acceso a estos registros ocurre de forma transparente para la aplicación VHDL. El CS correspondiente al

BAR6 (CSn(5)) no se activará cuando se acceda a estos registros. En su lugar, se activa cuando se van a transferir datos, de forma que la aplicación VHDL debe tratarlo de forma similar a cualquier otra señal de CS de los otros BAR en modo target. Si el driver del dispositivo se asegura de programar una longitud de transferencia no superior a los datos disponibles en la tarjeta, de forma que se asegure que se va a transferir en su totalidad, la señal mRW puede ignorarse y mENABLE conectarse fija a 1.

La descripción de los registros es la siguiente:

- Registro BASE+0: Dirección de comienzo
  - Bits 31-0 Dirección de comienzo de la transferencia. Como la dirección tiene que alinearse a palabras de 32 bits, los bits 1 y 0 deben valer 0.

- Registro BASE+4: Registro de control
  - Bit 31: Master Interrupt Enable. Activa la generación de interrupciones por parte del core-master.
  - Bit 30-18: Sin uso. Reservado para ampliaciones futuras
  - Bit 17: MasterTerminate. Se activa cuando MasterInicia esta activo, y el contador de transferencias llega a cero. Genera interrupción si MIE esta activo. Es de solo lectura. Para limpiar la interrupcion desactivar MasterInicia o programar una nueva transferencia.
  - Bit 16: MasterInicia: Se pone a uno para comenzar a transferir datos. Una vez se ha terminado, el software debe ponerla a cero para limpiar la interrupción
  - Bit 15-12: Comando. Estos 4 bits contienen el comando que se volcará en el bus. El bit 12 indica la dirección de la transferencia. Se recomienda utilizar los comandos ‘0110’ para leer de memoria y ‘0111’ para escribir. Por defecto vale ‘0110’, por lo que para cambiar el sentido basta cambiar el bit 12.
  - Bit 11-0: Longitud de la transferencia en palabras de 32 bits. El tamaño máximo serían 4096 palabras de 4 bytes, por lo que el tamaño máximo de la ráfaga serían 16KB. No se aprecia mejor de rendimiento importante para ráfagas mucho mayores, y el bus podria quedar ocupado demasiado tiempo.

11.2.5 Ejemplo sencillo de utilización del Core-PCI

Para ilustrar la utilización del Core-PCI descrito anteriormente, podemos utilizar un sencillo ejemplo de aplicación de usuario, que únicamente haga uso de un registro BAR (BAR0), y que implemente un registro de 32 bits que puede leerse a través de entrada/salida. Para ello, habrá que configurar el core de forma que BAR0 esté activado (BAR0Enabled = '1'), y que sea de entrada/salida:

```
constant MASKAND0: std_logic_vector(31 downto 0):=x"FFFFFFFC";
constant MASKOR0:    std_logic_vector(31 downto 0):=x"00000001";
constant BAR0Enabled: std_logic:='1';
```

La constante MASKAND0 es una máscara AND que se aplica sobre la dirección para separar los bits de la dirección que decodifican un dispositivo, y cuales lo direccionan internamente. En este caso, como únicamente queremos un registro de 32 bits, los dos bits menos significativos de la mascara AND están a cero para que no tengan significado alguno en la comparación de la dirección de comienzo de la transacción (almacenada en LAddress) y la indicada en el registro BAR0.

```
CSn <= '1' when (((BARn and MASKANDn) = (LAddress and MASKANDn)) else '0'
```

La constante MASKOR0 es una máscara OR que se aplica sobre el valor que quiere escribirse en el registro (al igual que MASKAND0), pero en este caso, con una operación lógica OR, de forma que los bits a 0 en esta máscara permanecerán inalterados, mientras que los que se coloquen a 1 se activarán al escribir en el BAR0. De esta forma, colocando el bit 0 a 1 en la máscara MASKOR0, forzamos a que el bit menos significativo sea 1, indicando que codifica una dirección de entrada/salida.

La expresión lógica que definiría una escritura sobre un registro BAR sería de manera simplificada la siguiente:

```
BARn <= (AD_Bus and MASKANDn) or MASKORn
```

Por último, BAR0Enabled se coloca a nivel lógico '1' para indicar que el BAR0 se va a utilizar en la decodificación. Los demás BARnEnabled, se colocarán a cero dado que para nuestro

ejemplo sólo se va a utilizar un registro BAR.

Las señales de la entidad “aplicación de usuario” que se va a implementar serán las definidas en el core-PCI para este fin:

```
entity UserApp is
Port (ADDRESS: in std_logic_vector(31 downto 0);
              DATA: inout std_logic_vector(31 downto 0);
              BE: in std_logic_vector(3 downto 0);
              CSn: in std_logic_vector(5 downto 0);
              ReadWrite: in std_logic;
              TWithData: out std_logic;
              TWithoutData: out std_logic;
              BUSY: out std_logic;
              DCLK: in std_logic;
              SCLK: in std_logic
       );
end UserApp;
```

Hay que tener en cuenta, que las señales que son de entrada en el core-PCI, serán salidas en la aplicación de usuario, y viceversa.

Para implementar el registro, y que este pueda leerse, se define un registro de 32 bits, y un proceso que se encargará de escribir o leer de el, en función de las señales facilitadas por el core:

```
signal miregistro: std_logic_vector(31 downto 0);

      mf: process(DATA, CSn, ReadWrite, DENABLE, SCLK, micont)
      begin
            if CSn(0) = '1' then
                  if ReadWrite='1' then
                        DATA <= miregistro;
                  else
                        DATA <= (others =>'Z');
                        if (SCLK'event and SCLK = '1' and DENABLE ='1') then
                              miregistro <= DATA;
                        end if;
                  end if;
            else
                  DATA <= (others => 'Z');
            end if;
      end process;
```

En este caso, si CSn(0) esta desactivado, no se estará accediendo a nuestro dispositivo. Si CSn(0) esta activo, en función de que sea una operación de lectura o de escritura (señal ReadWrite), se volcará en el bus de datos (DATA) el contenido de mi registro, o se colocará en alta impedancia, para el el flanco de subida de la señal DCLK se cargue el valor de dicho bus en el registro interno “miregistro).

Con esto sería suficiente para implementar una sencilla aplicación de usuario, manteniendo desactivadas las señales Busy, TWithData y TWithoutData:

```
Busy <= ‘0’;
TWithData <= ‘0’;
TWithoutData <= ‘0’;
```

Para demostrar el funcionamiento de estas tres señales, podemos suponer que la aplicación de usuario va a necesitar tres estados de espera antes de poder completar la fase de datos de la transacción, y que además, en previsión de que el master pudiera intentar más de una fase de datos, se terminará la transacción con una condición de “Terminate with data” para frustrar una posible segunda fase de datos:

Se define un proceso que contará el número de ciclos de reloj que lleva la transacción actual:

```
mf2: process(CSn, SCLK)
begin
       if CSn(0)='0' then
              micont <=(others => '0');
       elsif SCLK'event and SCLK ='1' then
              micont <= micont + 1;
       end if;
end process;
```

Mantendremos la señal busy activada durante los tres primeros ciclos de reloj de la transacción, manteniendo de esta forma TRDY# desactivada impidiendo que se termine la primera fase de datos, y una vez pasados estos tres ciclos de reloj, se desactiva BUSY para permitir la terminación de esta fase de datos, y se activa TWithData para que el target active STOP# y termine la transacción:

```
BUSY <= '0' when (micont >3) else '1';
TWithData <= '1' when (micont >3) else '0';
TWithoutData <= '0';
```

El funcionamiento de esta aplicación de usuario puede observarse en la siguiente simulación. Una vez definida la entidad “aplicación de usuario”, se conecta al core-PCI, y este a su vez a una entidad techbench que simula el comportamiento de un dispositivo master-PCI. Como vemos, aunque la señal DEVSEL# se active junto a IRDY# en el segundo ciclo de reloj (estado B_Busy), TRDY# va a permanecer desactivada durante tres ciclos de reloj, hasta la aplicación de usuario desactive BUSY y active TWithData como puede verse en la simulación.

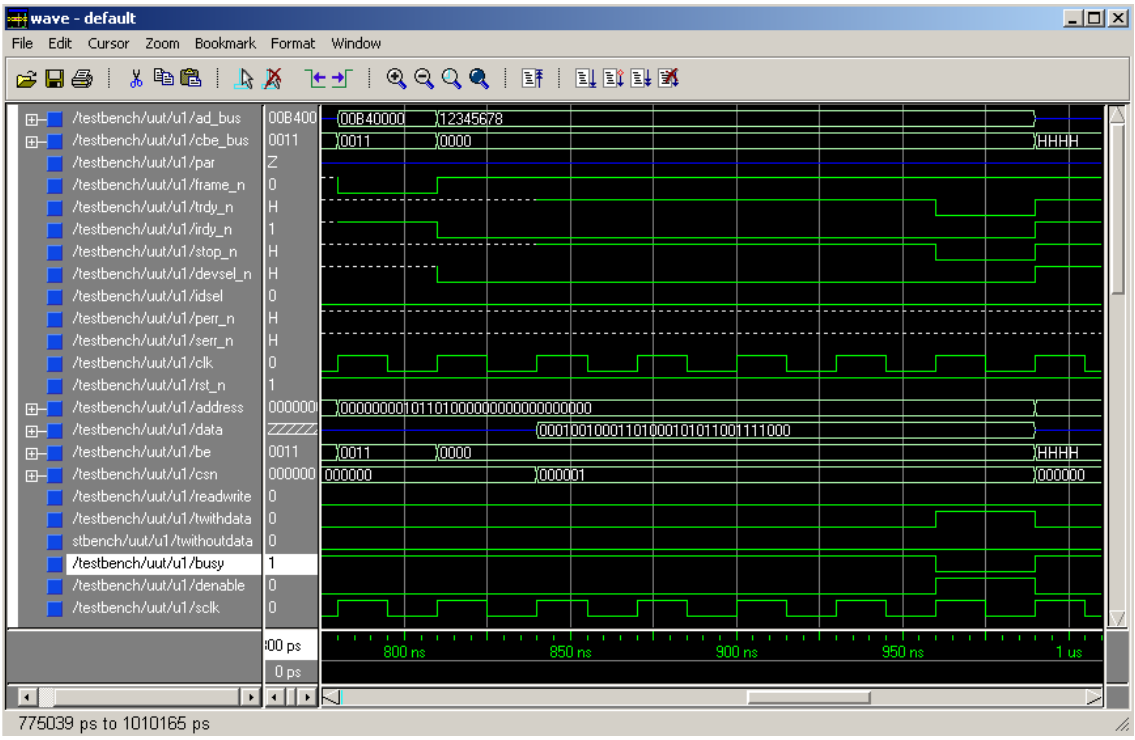

Ilustración 11.7: Condición de terminación PCI "Terminate With Data"

11.3 Código Matlab

11.3.1 Simulador Generador Exhaustivo

```
function genera_exaustivo (origen,destino)
    handle=fopen(origen,'rb');
    data=fread(handle);
    fclose(handle);
    tam=size(data,1);
    salida=zeros(255*tam,1);

    k=1;
    for iter=0:254
        for i=0:tam-1
            if (data(i+1)>iter)
                salida(k)=i;
            else
                salida(k)=-1;
            end;
            k=k+1;
        end;
     end;

     save(destino,'salida');
```

11.3.2 Simulador Mapper Probabilistico

```
function mapea_eventos_prob(origen, mapptable,destino,step)
    load (origen);
    load (mapptable);
    tam=size(salida,1);
    dest=zeros(tam*step,1);
    j=1;
    for i=1:tam
        evento_in=salida(i);
        if(evento_in>=0)
            k=1;
            ultimo=0;
            while(ultimo==0)
                dato=mapper(evento_in*step+k);
                evento=bitand(dato, 65535);
                ultimo=bitand(bitshift(dato,-16), 1);
                repets=bitand(bitshift(dato,-17), 15);
                prob= bitand(bitshift(dato, -24), 255);
                lfsr=fix(rand*255);
                if(prob>=lfsr)
                    for r=1:repets
                        dest(j)=evento;
                        j=j+1;
                    end;
                end;
                k=k+1;
            end;
        end;
    end;


    salida=dest(1:j);
    save(destino,'salida');
```

11.3.3 Simulador Integrador Frames

```
function integra_frame(origen, side,zoom)
    load (origen);

    tam=size(salida);
    %side=sqrt(max);
    despla_neg=side*side;
    imagen=ones(2*side*side,1);
    for i=1:tam
        dato=salida(i)+1;
        signo=bitand(bitshift(dato,-15), 1);
        index=bitand(dato,32767);
        if (index>0)
            if (signo==1 & imagen(despla_neg+index)<255)
                imagen(despla_neg+index)=imagen(despla_neg+index)+1;
            elseif(imagen(index)<255)
                imagen(index)=imagen(index)+1;
            end;
        end;
    end;

    salida=reshape(imagen,side,2*side)';
    salida=[salida(1:side,:) salida(side+1:2*side,:)];
    micolormap=[0:1/255:1; 0:1/255:1; 0:1/255:1]';
    colormap(micolormap);
    image(salida);    %% debe ser image


    salida2=ones(side*zoom,2*side*zoom);
    for i=1:side
        for j=1:side*2

            salida2(i*zoom-zoom+1:i*zoom,j*zoom-zoom+1:j*zoom) =
             fix(1+(salida(i,j)-1))*ones(zoom,zoom);

        end;
    end;

    imwrite(salida2,micolormap,'rafainterm.jpg','jpg');

    'pulse para continuar'

    pause;
    r1=salida(:,1:side);
    r2=salida(:,side+1:2*side);

    image(abs(r1-r2));
    %imagesc(r1-r2);
    imagen=abs(r1-r2);

    imagen2=ones(side*zoom,side*zoom);
    for i=1:side
        for j=1:side
                        imagen2(i*zoom-zoom+1:i*zoom,j*zoom-zoom+1:j*zoom)  =  fix(1+
(imagen(i,j)))*ones(zoom,zoom);
        end;
    end;

    imwrite(imagen2,micolormap,[origen '.jpg'],'jpg');
```

11.3.4 Simplificador eventos

```
function simplifica_eventos(origen,destino,side,cuenta)
    load (origen);
    tam=size(salida,1);
    dest=zeros(tam,1);
    matriz=zeros(side*side,1);

    j=1;
    for i=1:tam
        evento_in=salida(i);
        signo=bitand(bitshift(evento_in,-15), 1);
        evento=bitand(evento_in,32767);
        if (signo==0)
            valor=matriz(evento+1);
            if (valor==cuenta)
                dest(j)=evento_in;
                j=j+1;
            else
                matriz(evento+1)=valor+1;
            end;
        else
            valor=matriz(evento+1);
            if (valor==0)
                dest(j)=evento_in;
                j=j+1;
            else
                matriz(evento+1)=valor-1;
            end;
        end;
    end;

    salida=dest(1:j-1);
    save(destino,'salida');
```

11.3.5 Rutinas para generar tablas del mapper probabilístico

11.3.5.1 Desplazamiento Arbitrario

```
function mapper=genera_desplaxy_prob(destino,dx,dy,side,step)
    maxx=side;
    maxy=side;
    mapper=zeros(maxx*maxy*step,1);

    i=1;
    for oldy=0:maxy-1
        for oldx=0:maxx-1

            x=mod(oldx+dx,maxx);
            y=mod(oldy+dy,maxy);

            if (x<maxx) && (x>=0)&& (y<maxy) && (y>=0)
                x1=fix(x);
                y1=fix(y);
                px=x-x1;
                py=y-y1;
                                mapper(i)=y1*maxx+x1+hex2dec('00020000')+fix(255*(1-px)*(1-
py))*hex2dec('01000000');
                r=1;
                if (y1<maxy-1)
                                 mapper(i+r)=(y1+1)*maxx+x1+hex2dec('00020000')+fix(255*(1-
px)*(py))*hex2dec('01000000');
                    r=r+1;
                end;
                if (x1<maxx-1)
                                mapper(i+r)=y1*maxx+x1+1+hex2dec('00020000')+fix(255*(px)*(1-
py))*hex2dec('01000000');
                    r=r+1;
                end;
                if((y1<maxy-1)&&(x1<maxx-1))
                                            mapper(i+r)=(y1+1)*maxx+x1+1+hex2dec('00030000')
+fix(255*px*py)*hex2dec('01000000');
                end;
            else
                mapper(i)=hex2dec('00010000');
            end;

            i=i+step;
        end;
    end;
```

11.3.5.2 Rotación Arbitrario

11.3.5.3 Ampliación o zoom in

```
function genera_scale_prob2(destino,escala,side,step)
    maxx=side;
    maxy=side;

    offset=[(maxx-1)/2 ; (maxy-1)/2];
    mapper=zeros(maxx*maxy*step,1);

    nivel=255;

    i=1;
    for oldy=0:maxy-1
        for oldx=0:maxx-1
            punto=[oldx; oldy];
            punto=punto-offset;
            res=punto'*escala;
            res=res'+offset;
            x=res(1);
            y=res(2);
            r=0;
             widey=escala;
            fixx=floor(x);
            fixy=floor(y);
             py=1-(y-fixy);
            iy=0;
            while(widey>0)
                ix=0;
                px=1-(x-fixx);
                widex=escala;

                while(widex>0)
                    if (fixx+ix<maxx) && (fixx+ix>=0)&& (fixy+iy<maxy) && (fixy+iy>=0)
                              mapper(i+r)=((fixy+iy)*maxx+fixx+ix)+hex2dec('00020000')+
                          fix(nivel*px*py)*hex2dec('01000000');
                        r=r+1;
                    end;
                    widex=widex-px;
                    ix=ix+1;
                    if(widex>1)
                        px=1;
                    else
                        px=widex;
                    end;
                end;
                iy=iy+1;
                widey=widey-py;
                if(widey>1)
                    py=1;
                else
                    py=widey;
                end;
            end;
            if (r>0)
                mapper(i+r-1)=mapper(i+r-1)+hex2dec('00010000');
            end;
             i=i+step;
        end;
    end;

    save (destino,'mapper');
```

11.3.5.4 Convoluciones

```
function mapper=genera_convolucion(destino,side,mconv)
    %mconv=[1 0 0; 0 0 0; 0 0 0];
    maxx=side;
    maxy=side;
    step=size(mconv(:),1);

    mapper=zeros(maxx*maxy*step,1);
    i=0;

    for y=0:maxy-1
        for x=0:maxx-1
            r=1;
            for px=1:size(mconv,2)
                for py=1:size(mconv,1)


                    ctmp=mconv(py,px);
                    csigno=ctmp<0;
                    cvalor=abs(ctmp);
                    salx=x+px-1;
                    saly=y+py-1;

                        if (salx<side & saly<side & cvalor >0)
                            repe=ceil(cvalor);
                            resto=cvalor/repe;
                            csalida=saly*maxx+salx+repe*hex2dec('00020000')+
                          fix(255*resto)*hex2dec('01000000')
+csigno*hex2dec('00008000');
                            if (px==size(mconv,2) & py == size(mconv,1))
                                csalida=csalida + hex2dec('00010000');
                            end;
                            mapper(i*step+r)=csalida;
                            r=r+1;
                        elseif (px==size(mconv,2) & py == size(mconv,1))
                            mapper(i*step+r)=hex2dec('00010000');
                        end;
                end;
            end;
            i=i+1;
        end;
    end;

    save (destino,'mapper');
```

11.3.5.5 Simulación olvido

```
function [medias]=simula(datos)
TFrame=1;
frames=6;
colores='rgbRGB';
medias=zeros(1,size(datos,2));
for i=[1:size(datos,2)]
    [x,y]=olvido1(TFrame,datos(i),frames);
    fmed=size(y,1);
    imed=fix(fmed*(frames-1)/frames);
    medias(i)=mean(y(imed:fmed));
    plot(x,y,colores(i));
    hold on;
end
hold off;


function [x,y]=olvido1(TFrame,Pixel,frames)

TSim=frames*TFrame;
Tslice=TFrame/255;

Tpixel=TFrame/Pixel;

slots=TSim/Tslice+1;

x=[0:Tslice:TSim]';
y=zeros(slots,1);

frame=0;
evento=0;
valor=0;

ganancia=10;
folvido=ganancia/255;

for t=[1:slots]
    if(x(t)>evento*Tpixel)
        evento=evento+1;
        if(valor<255)
            valor=valor+ganancia;
        end
    end
    y(t)=valor;


    %Olvido:


    if(valor >0)
        valor=(1-folvido)*valor;
    end

    %%%% frames
    if(x(t)>frame*TFrame)
        %valor=0;                % Olvido por frames
        frame=frame+1;
    end
end
```

11.3.5.6 Simulación AER tradicional con olvido

```
function [x,y]=olvido3(MAXPIXEL,Pixel)
%MAXPIXEL=255;

slots=size(Pixel,1)*MAXPIXEL;
TFrame=MAXPIXEL;

Tpixel=TFrame/Pixel;

x=[0:0.5/MAXPIXEL:slots/MAXPIXEL]';

eventos=zeros(2*slots+1,1);
y=zeros(2*slots+1,1);
z=zeros(2*slots+1,1);
e=zeros(2*slots+1,1);

frame=1;
evento=0;
valor=0;

folvido=0.1;
%folvido=0.0;
ferror=0.77;
valor_local=0;
valor_remoto=0;
t=0;

%ganancia=1;
ganancia=1.4*(folvido)*MAXPIXEL;
    svc=0;
for f=[1:size(Pixel,1)]
    valor_pixel=Pixel(f);

    sv=MAXPIXEL/(valor_pixel+0.01);


    for s=[1:MAXPIXEL]
        t=t+1;

        e(2*t)=valor_pixel;
        e(2*t+1)=valor_pixel;

        evento=0;
        if(valor_local ~= 0)
            valor_local=(1-folvido)*valor_local;
        end
        svc=svc+1;
        if(svc>sv)
            evento=1;
            svc=0;
        end


        eventos(2*t)=evento;
        y(2*t)=valor_local;
        y(2*t+1)=valor_local;

        if(evento==1 && valor_local<MAXPIXEL)
            valor_local=valor_local+ganancia;
        elseif (evento==-1 && valor_local>0)
            valor_local=valor_local-ganancia;
        end


        if(valor_remoto ~= 0)
            valor_remoto=(1-folvido)*valor_remoto;
```

```
        end

        if(evento==1 && valor_remoto<MAXPIXEL&&rand<=ferror)
            valor_remoto=valor_remoto+ganancia;
        elseif (evento==-1 && valor_remoto>0&&rand<=ferror)
            valor_remoto=valor_remoto-ganancia;
        end

        z(2*t)=valor_remoto;
        z(2*t+1)=valor_remoto;


    end
    plot(x,y,'g');
    hold on;
    plot(x,e,'r');
    plot(x,eventos);
    hold off;
end
```

11.3.5.7 Simulación AER diferencial

```
function [x,y]=olvido2(MAXPIXEL,Pixel)
%MAXPIXEL=255;

slots=size(Pixel,1)*MAXPIXEL;
TFrame=MAXPIXEL;

Tpixel=TFrame/Pixel;

x=[0:0.5/MAXPIXEL:slots/MAXPIXEL]';

eventos=zeros(2*slots+1,1);
y=zeros(2*slots+1,1);
z=zeros(2*slots+1,1);

frame=1;
evento=0;
valor=0;

ganancia=1;
folvido=0.0;
ferror=0.8;
valor_local=0;
valor_remoto=0;
t=0;
for f=[1:size(Pixel,1)]
    valor_pixel=Pixel(f);
    for s=[1:MAXPIXEL]
        t=t+1;
        evento=0;

        if(valor_pixel>valor_local) %emite
            evento=1;
        elseif(valor_pixel<valor_local)
            evento=-1;
        end
        eventos(2*t)=evento;
        y(2*t)=valor_local;
        y(2*t+1)=valor_local;

        if(evento==1 && valor_local<MAXPIXEL)
            valor_local=valor_local+ganancia;
        elseif (evento==-1 && valor_local>0)
            valor_local=valor_local-ganancia;
        end
        if(evento==1 && valor_remoto<MAXPIXEL&&rand<=ferror)
            valor_remoto=valor_remoto+ganancia;
        elseif (evento==-1 && valor_remoto>0&&rand<=ferror)
            valor_remoto=valor_remoto-ganancia;
        end

        z(2*t)=valor_remoto;
        z(2*t+1)=valor_remoto;


    end
    plot(x,y,'g');
    hold on;
    plot(x,z,'r');
    plot(x,eventos);
    hold off;
end
```

11.3.5.8 Simulación AER diferencial con olvido

```
function [x,y]=olvido2(MAXPIXEL,Pixel)
%MAXPIXEL=255;

slots=size(Pixel,1)*MAXPIXEL;
TFrame=MAXPIXEL;
 Tpixel=TFrame/Pixel;
 x=[0:0.5/MAXPIXEL:slots/MAXPIXEL]';
 eventos=zeros(2*slots+1,1);
y=zeros(2*slots+1,1);
z=zeros(2*slots+1,1);
 frame=1;
evento=0;
valor=0;

ganancia=1;
folvido=0.05;
ferror=0.8;
valor_local=0;
valor_remoto=0;
t=0;
for f=[1:size(Pixel,1)]
    valor_pixel=Pixel(f);
    for s=[1:MAXPIXEL]
        t=t+1;
         evento=0;
        if(valor_local ~= 0)
            valor_local=(1-folvido)*valor_local;
        end
        if(valor_pixel>valor_local) %emite
            evento=1;
        elseif(valor_pixel<valor_local-ganancia)
            evento=-1;
        end
        eventos(2*t)=evento;
        y(2*t)=valor_local;
        y(2*t+1)=valor_local;
        if(evento==1 && valor_local<MAXPIXEL)
            valor_local=valor_local+ganancia;
        elseif (evento==-1 && valor_local>0)
            valor_local=valor_local-ganancia;
        end
        if(valor_remoto ~= 0)
            valor_remoto=(1-folvido)*valor_remoto;
        end

        if(evento==1 && valor_remoto<MAXPIXEL&&rand<=ferror)
            valor_remoto=valor_remoto+ganancia;
        elseif (evento==-1 && valor_remoto>0&&rand<=ferror)
            valor_remoto=valor_remoto-ganancia;
        end

        z(2*t)=valor_remoto;
        z(2*t+1)=valor_remoto;


    end
    plot(x,y,'g');
    hold on;
    plot(x,z,'r');
    plot(x,eventos);
    hold off;
end
```

11.4 Código VHDL

11.4.1 Generador exhaustivo bit-wise

mapper_genera_exaustivo.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity mapper_genera_exaustivo is
       Port (
              clk50 : in std_logic;
              rst_l : in std_logic;
              led : out std_logic_vector( 2 downto 0 );

              -- AER input
              aer_in_data : in std_logic_vector( 15 downto 0 );
              aer_in_req_l : in std_logic;
              aer_in_ack_l : out std_logic;

              -- AER output
              aer_out_req_l : out std_logic;
              aer_out_ack_l : in std_logic;
              aer_out_data : out std_logic_vector( 15 downto 0 );

              -- SRAM interface
              sram_oe_l : out std_logic;
              sram_we_l : out std_logic_vector(3 downto 0);
              address : out std_logic_vector( 18 downto 0 );
              sram_data : inout std_logic_vector( 31 downto 0 );

              -- Micro interface
              micro_vdata : in std_logic;
              micro_prog : in std_logic;
              micro_control: in std_logic;     -- Aux0 en sch
              micro_rw: in std_logic;
              micro_busy:out std_logic;
              micro_data:  inout std_logic_vector( 7 downto 0 );

              -- Other control lines
              enable_out_buffers_l : out std_logic;
              enable_in_buffers_l: out std_logic
       );
end mapper_genera_exaustivo;

architecture Behavioral of mapper_genera_exaustivo is

--- duplicar reloj ---
component IBUFG
port (
   O : out std_logic;
   I : in std_logic
 );
end component;
```

```
component CLKDLL
 port (
   CLK0 : out std_logic;
   CLK90 : out std_logic;
   CLK180 : out std_logic;
   CLK270 : out std_logic;
   CLK2X : out std_logic;
   CLKDV : out std_logic;
   LOCKED : out std_logic;
   CLKIN : in std_logic;
   CLKFB : in std_logic;
   RST : in std_logic
 );
end component;

component BUFG
 port (
   O : out std_logic;
   I : in std_logic
 );
end component;
signal CLKIN_w, RESET_w, CLK2X_dll, CLK2X_g,CLK100 : std_logic;
signal LOCKED2X, LOCKED2X_delay: std_logic;

---- fin duplicar reloj ----


       COMPONENT metodo_exaustivo1
       generic(RAM_Size: in integer; RAM_Addr_Size: in integer;   RAM_Data_Size:     in
integer);
       PORT(
              address : IN std_logic_vector(RAM_Addr_Size -1 downto 0);
              data : IN std_logic_vector(RAM_Data_Size-1 downto 0);
              cfgmode : IN std_logic;
              vdata: in std_logic;
              enable : IN std_logic;
              CLK : IN std_logic;
              emite : OUT std_logic;
              evento : OUT std_logic_vector(RAM_Addr_Size-1 downto 0);
              rst_n: in std_logic;
              debug : OUT std_logic_vector(8 downto 0)
              );
       END COMPONENT;

       COMPONENT aer_out
       PORT(
              REQ_N : OUT std_logic;
              ACK_N : IN std_logic;
              Data_AER : OUT std_logic_vector(15 downto 0);
              Data_Fifo : IN std_logic_vector(15 downto 0);
              Enable_Fifo : IN std_logic;
              RD_Fifo : OUT std_logic;
              CLK : IN std_logic;
              RST_N : IN std_logic
              );
       END COMPONENT;


       COMPONENT ramfifo
       generic (TAM: in integer; IL: in integer; WL: in integer);
       PORT(
              clk : IN std_logic;
              wr : IN std_logic;
              rd : IN std_logic;
              rst_n : IN std_logic;
              empty : OUT std_logic;
```

```
              full : OUT std_logic;
              data_in : IN std_logic_vector(WL-1 downto 0);
              data_out : OUT std_logic_vector(WL-1 downto 0);
              mem_used: out std_logic_vector(IL-1 downto 0)

              );
       END COMPONENT;

signal imicro_prog, smicro_prog: std_logic;
signal imicro_vdata, smicro_vdata: std_logic;
signal imicro_control, smicro_control: std_logic;
signal imicro_rw, smicro_rw: std_logic;
signal iaer_in_req_l,  saer_in_req_l: std_logic;
signal iaer_out_ack_l, saer_out_ack_l: std_logic;

signal counter: std_logic_vector(23 downto 0);

SIGNAL maddress :  std_logic_vector(11 downto 0);
SIGNAL cfgmode :  std_logic;
SIGNAL vdata :  std_logic;
SIGNAL enable :  std_logic;
SIGNAL emite :  std_logic;
SIGNAL evento :  std_logic_vector(11 downto 0);
SIGNAL debug :  std_logic_vector(8 downto 0);

signal fifo_wr, fifo_rd, fifo_rd_2, fifo_empty, fifo_full: std_logic;
signal fifo_data_in, fifo_data_out: std_logic_vector(15 downto 0);
signal latched_smicro_vdata: std_logic;

signal not_fifo_empty: std_logic;

signal mem_used: std_logic_vector(8 downto 0);


signal rst_n: std_logic;
signal rst: std_logic;
signal clk: std_logic;


begin

---- duplicar reloj-----
rst <= not rst_l;
clkpad : IBUFG  port map (I=>CLK50, O=>CLKIN_w);

dll2x  : CLKDLL port map (CLKIN=>CLKIN_w,   CLKFB=>CLK2X_g, RST=>RST,
                          CLK0=>open,   CLK90=>open, CLK180=>open, CLK270=>open,
                           CLK2X=>CLK2X_dll, CLKDV=>open, LOCKED=>LOCKED2X);

clk2xg : BUFG   port map (I=>CLK2X_dll,   O=>CLK2X_g);
clk50g : BUFG   port map (I=>CLKIN_w,   O=>open); --CLK
--CLK <= CLK2X_g;


CLK100 <= CLK2X_g;
CLK <= CLKIN_w;


       exaustivo: metodo_exaustivo1
       GENERIC MAP(4096,12,8)
       PORT MAP(
              address => maddress,
              data => micro_data,
              cfgmode => cfgmode,
              vdata => vdata,
              enable => enable,
              emite => emite,
              evento => evento,
```

```
              debug => debug,
              rst_n => rst_n,
              CLK => clk
       );

       aerout: aer_out PORT MAP(
              REQ_N => aer_out_req_l,
              ACK_N => saer_out_ack_l,
              Data_AER => aer_out_data,
              Data_Fifo => fifo_data_out,
              Enable_Fifo => not_fifo_empty,
              RD_Fifo => Fifo_rd,
              CLK => clk,
              RST_N => rst_n
       );

       fifo: ramfifo
       ---GENERIC MAP(1024,10,16)
   GENERIC MAP(512,9,16)

       PORT MAP(
              clk => clk,
              wr => fifo_wr,
              rd => Fifo_rd,
              rst_n => RST_N,
              empty => Fifo_empty,
              full => Fifo_full,
              data_in => fifo_Data_In,
              data_out => fifo_Data_Out,
              mem_used => mem_used
       );


syncronizer: process(RST_N, CLK100)
begin
if (RST_N = '0') then
       imicro_vdata <= '0';
       smicro_vdata <= '0';
       imicro_rw <= '0';
       smicro_rw <= '0';
       imicro_prog <= '0';
       smicro_prog <= '0';
       imicro_control <= '0';
       smicro_control <= '0';
       iaer_out_ack_l <= '0';
       saer_out_ack_l      <= '0';
       counter <= (others =>'0');
elsif(CLK100'event and CLK100 = '1') then
       imicro_vdata <= micro_vdata;
       smicro_vdata <= imicro_vdata;
       imicro_rw <= micro_rw;
       smicro_rw <= imicro_rw;
       imicro_control <= micro_control;
       smicro_control <= imicro_control;
       imicro_prog <= micro_prog;
       smicro_prog <= imicro_prog;
       iaer_out_ack_l <= aer_out_ack_l;
       saer_out_ack_l      <= iaer_out_ack_l;   -- Solo 1 biestable!
end if;
end process;



divisor: process(RST_N, clk)
begin
if (RST_N ='0') then
       counter <= (others =>'0');
elsif(clk'event and clk='1') then
```

```
      counter <= counter +1;
end if;
end process;

interfaz_cygnal: process(RST_N, clk,smicro_prog)
begin
if (RST_N = '0' or smicro_prog ='0') then
      maddress <= (others => '0');
      latched_smicro_vdata <= '0';
elsif (clk'event and clk = '1') then
      latched_smicro_vdata      <= smicro_vdata;
      if (latched_smicro_vdata = '1' and smicro_vdata = '0' and cfgmode = '1') then
            maddress <= maddress + 1;
      end if;
end if;
end process;

cfgmode <= '1' when (smicro_prog='1' and smicro_control='0' and smicro_rw = '0') else
'0';
fifo_wr <= emite and not fifo_full;
enable <= not fifo_full;
vdata <= smicro_vdata;
not_fifo_empty <= not fifo_empty;

led(2) <= counter(22);
led(1)<= aer_out_ack_l;
led(0) <= '1' when (aer_in_data = x"0000") else '0';



fifo_data_in(5 downto 0) <= evento(5 downto 0);
fifo_data_in(7 downto 6) <= (others => '0');
fifo_data_in(13 downto 8) <= evento(11 downto 6);
fifo_data_in(15 downto 14) <= (others => '0');
micro_busy <= '0';
--enable <= '1' when ((cfgmode='0') or (cfgmode='1' and micro_vdata = '1')) else '0';
micro_data <= (others =>'Z');
aer_in_ack_l <= aer_in_req_l;

-- Deshabilitamos las RAMS
sram_oe_l <='1';
sram_we_l <= "1111";
address <= (others =>'0');
sram_data <= (others =>'0');
enable_out_buffers_l <= '0';
enable_in_buffers_l <= '0';
rst_n <= rst_l;
end Behavioral;
```

metodo_exaustivo1.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED."+";
use IEEE.STD_LOGIC_UNSIGNED."-";

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;


entity metodo_exaustivo1 is
generic(RAM_Size: in integer := 4096; RAM_Addr_Size: in integer := 12;  RAM_Data_Size:
```

```
in integer := 8);
--generic(RAM_Size: in integer := 64; RAM_Addr_Size: in integer := 6;   RAM_Data_Size:
in integer := 8);
    Port ( address : in std_logic_vector(RAM_Addr_Size -1 downto 0);
           data : in std_logic_vector(RAM_Data_Size-1 downto 0);
                     cfgmode: in std_logic;
                     vdata: in std_logic;
                     enable: in std_logic;
           emite : out std_logic;
                     evento: out std_logic_vector(RAM_Addr_Size-1 downto 0);
                     debug: out std_logic_vector(8 downto 0);
                     RST_N: in std_logic;
                     CLK: in std_logic
                     );
end metodo_exaustivo1;

architecture Behavioral of metodo_exaustivo1 is

      COMPONENT dualram
      generic (TAM: in integer:= RAM_Size; IL: in integer:=RAM_Addr_Size; WL: in
integer:=RAM_Data_Size);
      PORT(
            clk : IN std_logic;
            wr   : in std_logic;
            index_i : in std_logic_vector(IL-1 downto 0);
            index_o : in std_logic_vector(IL-1 downto 0);
            word_i   : in std_logic_vector(WL-1 downto 0);
            word_o  : out std_logic_vector(WL-1 downto 0)
            );
      END COMPONENT;

signal image_out: std_logic_vector(RAM_Data_Size-1 downto 0);

signal maddress: std_logic_vector(RAM_Addr_Size-1 downto 0);


signal iaddress: std_logic_vector(RAM_Addr_Size-1 downto 0);
signal numitera: std_logic_vector(RAM_Data_Size-1 downto 0);

signal levento: std_logic_vector(RAM_Addr_Size-1 downto 0);

signal contador: std_logic_vector(RAM_Data_Size+RAM_Addr_Size -1 downto 0);
signal dcontador: std_logic_vector(RAM_Data_Size+RAM_Addr_Size -1 downto 0);
signal LFSR: std_logic_vector(15 downto 0);

begin

imagen: dualram PORT MAP(
            clk => clk,
            wr => vdata,
            index_i => address,
            index_o => maddress,
            word_i => data,
            word_o => image_out
            );

--lfsr_counter: process(cfgmode, CLK, LFSR)
--begin
--if(cfgmode = '1') then
--     LFSR <= (others =>'1');          --  Iniciación,  siempre  distinto  de  todos  a
cero.
--elsif (CLK'event and CLK='1') then
--     for i in 15 downto 1 loop
--           LFSR(i) <= LFSR(i-1);
--     end loop;
--     LFSR(0) <= LFSR(15) xor LFSR(14) xor LFSR(12) xor LFSR(3); --    El    polinomio
empieza en 0, 1 menos!
--end if;
```

```
--end process;

invertir: process(dcontador)
begin
for i in 0 to RAM_Data_Size-1 loop
       numitera(RAM_Data_Size-1-i) <= dcontador (RAM_Addr_Size+i);
end loop;
end process;



icounter: process(rst_n, cfgmode, CLK, contador, enable, image_out, numitera, levento)
begin
if (rst_n = '0') then
       contador <= (others=>'0');
       dcontador <= (others =>'1');
elsif(CLK'event and CLK='1') then
       if (enable ='1') then
              contador <= contador      + 1;
              dcontador <= contador;
       end if;
end if;
evento <= dcontador(RAM_Addr_Size -1 downto 0);
iaddress <= contador (RAM_Addr_Size -1 downto 0);
end process;

exaustivo: process(rst_n, data, cfgmode, address, iaddress, enable, vdata)
begin
       if(enable = '1') then
              maddress <= contador (RAM_Addr_Size -1 downto 0);
              if (image_out>numitera) then
                     emite <='1';
              else
                     emite <= '0';
              end if;
       else
              maddress <= dcontador(RAM_Addr_Size -1 downto 0);
              emite <= '0';
       end if;
end process;


end Behavioral;
```

aer_out.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity aer_out is
    Port (REQ_N : out std_logic;
          ACK_N : in std_logic;
          Data_AER : out std_logic_vector(15 downto 0);
          Data_Fifo : in std_logic_vector(15 downto 0);
          Enable_Fifo : in std_logic;
                    RD_Fifo: out std_logic;
                    CLK : in std_logic;
          RST_N : in std_logic);
end aer_out;


architecture Behavioral of aer_out is

type STATE_TYPE is (IDLE, SendDATA);
signal CS, NS: STATE_TYPE;
```

```
begin

       SYNC_PROC: process (CLK, RST_N)
       begin
        if (RST_N='0') then
               CS <= IDLE;
        elsif (CLK'event and CLK = '1') then
             CS <= NS;
        end if;
    end process;

    COMB_PROC: process (CS, Enable_Fifo, ack_n)
    begin
    case CS is

             when IDLE =>
                   REQ_N <= '1';
             if (Enable_Fifo ='1' and ACK_N='1') then
                         NS <= SendDATA;
                   else
                   NS <= IDLE;
             end if;
                   RD_Fifo <= '0';

             when SendDATA =>
                   REQ_N <='0';
                   if (ACK_N='0') then
                         NS <= IDLE;
                         RD_Fifo <= '1';
                   else
                         NS <= SendDATA;
                         RD_Fifo <= '0';
                   end if;
    end case;
      end process;

Data_AER <= Data_Fifo;

end Behavioral;
```

ramfifo.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ramfifo is
generic (TAM: in integer:= 64; IL: in integer:=6; WL: in integer:=16);
 port (clk  : in std_logic;
       wr   : in std_logic;
       rd    : in std_logic;
       rst_n: in std_logic;
       empty: out std_logic;
       full: out std_logic;
       data_in   : in std_logic_vector(WL-1 downto 0);
       data_out  : out std_logic_vector(WL-1 downto 0);
       mem_used : out std_logic_vector(IL-1 downto 0));
end ramfifo;



architecture syn of ramfifo is

       COMPONENT dualram
       generic (TAM: in integer:= TAM; IL: in integer:=IL; WL: in integer:=WL);
       PORT(
```

```
              clk : IN std_logic;
              wr   : in std_logic;
              index_i : in std_logic_vector(IL-1 downto 0);
              index_o : in std_logic_vector(IL-1 downto 0);
              word_i   : in std_logic_vector(WL-1 downto 0);
              word_o  : out std_logic_vector(WL-1 downto 0)
              );
       END COMPONENT;

       SIGNAL index_i :  std_logic_vector(IL-1 downto 0);
       SIGNAL index_o :  std_logic_vector(IL-1 downto 0);
       SIGNAL iempty: std_logic;
       SIGNAL ifull: std_logic;
       SIGNAL ramwr: std_logic;
       SIGNAL memused: std_logic_vector(IL downto 0);
       SIGNAL dout: std_logic_vector(WL-1 downto 0);


BEGIN

       uut: dualram generic map (TAM,IL,WL)
       PORT MAP(
              clk => clk,
              wr => ramwr,
              index_i => index_i,
              index_o => index_o,
              word_i => data_in,
              word_o => dout
              );

process (clk, rst_n)
begin
       if (rst_n ='0') then
              index_i <= (others=>'0');
              index_o <= (others=>'0');
              memused <= (others=>'0');
       elsif (clk'event and clk = '1') then
       if (wr = '1' and rd = '1') then
                     index_i <= index_i + 1;
                     index_o <= index_o + 1 ;
                     --memused <= memused; -- Al ser proceso sincrono y no cambiar
memused no es necesaria esta linea.
              elsif (wr = '1' and ifull = '0') then
                     index_i <= index_i + 1;
                     memused <= memused + 1;
              elsif (rd = '1' and iempty = '0') then
                     index_o <= index_o +1;
                     memused <= memused - 1;
              end if;
       end if;
 end process;

 --ramindex <= index_o when (rd ='1') else index_o;

 iempty <= '1' when memused = 0 else '0';
 ifull <= '1' when  memused = TAM-1  else '0';
 empty <= iempty;
 full <= ifull;
 ramwr <= '1' when (wr = '1' and (ifull='0' or rd='1')) else '0';
  data_out <= dout;-- when (iempty='0') else (others => 'Z'); -- Puede eliminarse y
puentearse la salida de dualram con la salida de la fifo.
 mem_used <= memused(IL-1 downto 0);
 end syn;
```

dual_port_ram.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity dualram is
generic (TAM: in integer:= 64; IL: in integer:=6; WL: in integer:=16);
 port (clk  : in std_logic;
       wr   : in std_logic;
       index_i : in std_logic_vector(IL-1 downto 0);
       index_o : in std_logic_vector(IL-1 downto 0);
       word_i   : in std_logic_vector(WL-1 downto 0);
       word_o  : out std_logic_vector(WL-1 downto 0));
end dualram;


-- Only XST supports RAM inference
-- Infers Dual Port Distributed Ram

 architecture syn of dualram is
 type ram_type is array (TAM-1 downto 0) of std_logic_vector (WL-1 downto 0);
 signal RAM : ram_type;
 signal indice_lectura: std_logic_vector(IL-1 downto 0);

 begin
 process (clk)
 begin
       if (clk'event and clk = '1') then
             if (wr = '1') then
                   RAM(conv_integer(index_i)) <= word_i;
             end if;
             indice_lectura <= index_o;
       end if;
 end process;
                            --indice_lectura <= index_o;
 word_o <= RAM(conv_integer(indice_lectura));


 end syn;
```

11.4.2 Retina sintética básica

mapper_genera_exaustivo.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity mapper_genera_exaustivo is
      Port (
            clk50 : in std_logic;
            rst_l : in std_logic;
            led : out std_logic_vector( 2 downto 0 );

            -- AER input
            aer_in_data : in std_logic_vector( 15 downto 0 );
            aer_in_req_l : in std_logic;
            aer_in_ack_l : out std_logic;

            -- AER output
            aer_out_req_l : out std_logic;
            aer_out_ack_l : in std_logic;
            aer_out_data : out std_logic_vector( 15 downto 0 );

            -- SRAM interface
            sram_oe_l : out std_logic;
            sram_we_l : out std_logic_vector(3 downto 0);
            address : out std_logic_vector( 18 downto 0 );
            sram_data : inout std_logic_vector( 31 downto 0 );

            -- Micro interface
            micro_vdata : in std_logic;
            micro_prog : in std_logic;
            micro_control: in std_logic;     -- Aux0 en sch
            micro_rw: in std_logic;
            micro_busy:out std_logic;
            micro_data:  inout std_logic_vector( 7 downto 0 );

            -- Other control lines
            enable_out_buffers_l : out std_logic;
            enable_in_buffers_l: out std_logic
      );
end mapper_genera_exaustivo;

architecture Behavioral of mapper_genera_exaustivo is

--- duplicar reloj ---
component IBUFG
port (
   O : out std_logic;
   I : in std_logic
 );
end component;


component CLKDLL
 port (
   CLK0 : out std_logic;
   CLK90 : out std_logic;
   CLK180 : out std_logic;
```

```
   CLK270 : out std_logic;
   CLK2X : out std_logic;
   CLKDV : out std_logic;
   LOCKED : out std_logic;
   CLKIN : in std_logic;
   CLKFB : in std_logic;
   RST : in std_logic
 );
end component;

component BUFG
 port (
   O : out std_logic;
   I : in std_logic
 );
end component;
signal CLKIN_w, RESET_w, CLK2X_dll, CLK2X_g,CLK100 : std_logic;
signal LOCKED2X, LOCKED2X_delay: std_logic;
---- fin duplicar reloj ----

      COMPONENT metodo_exaustivo1
      generic(RAM_Size: in integer; RAM_Addr_Size: in integer;   RAM_Data_Size:     in
integer);
      PORT(
            address : IN std_logic_vector(RAM_Addr_Size -1 downto 0);
            data : IN std_logic_vector(RAM_Data_Size-1 downto 0);
            vdata: in std_logic;
            enable : IN std_logic;
            CLK : IN std_logic;
            emite : OUT std_logic;
            evento : OUT std_logic_vector(RAM_Addr_Size-1 downto 0);
            rst_n: in std_logic;
            debug : OUT std_logic_vector(8 downto 0)
            );
      END COMPONENT;

      COMPONENT aer_out
      PORT(
            REQ_N : OUT std_logic;
            ACK_N : IN std_logic;
            Data_AER : OUT std_logic_vector(15 downto 0);
            Data_Fifo : IN std_logic_vector(15 downto 0);
            Enable_Fifo : IN std_logic;
            RD_Fifo : OUT std_logic;
            CLK : IN std_logic;
            RST_N : IN std_logic
            );
      END COMPONENT;

      COMPONENT ramfifo
      generic (TAM: in integer; IL: in integer; WL: in integer);
      PORT(
            clk : IN std_logic;
            wr : IN std_logic;
            rd : IN std_logic;
            rst_n : IN std_logic;
            empty : OUT std_logic;
            full : OUT std_logic;
            data_in : IN std_logic_vector(WL-1 downto 0);
            data_out : OUT std_logic_vector(WL-1 downto 0);
            mem_used: out std_logic_vector(IL-1 downto 0)

            );
      END COMPONENT;

signal imicro_prog, smicro_prog: std_logic;
signal imicro_vdata, smicro_vdata: std_logic;
signal imicro_control, smicro_control: std_logic;
```

```
signal imicro_rw, smicro_rw: std_logic;
signal iaer_in_req_l,  saer_in_req_l: std_logic;
signal iaer_out_ack_l, saer_out_ack_l: std_logic;

signal counter: std_logic_vector(23 downto 0);

signal maddress :  std_logic_vector(11 downto 0);
signal cfgmode :  std_logic;
signal vdata :  std_logic;
signal enable :  std_logic;
signal emite :  std_logic;
signal evento :  std_logic_vector(11 downto 0);
signal debug :  std_logic_vector(8 downto 0);

signal fifo_wr, fifo_rd, fifo_rd_2, fifo_empty, fifo_full: std_logic;
signal fifo_data_in, fifo_data_out: std_logic_vector(15 downto 0);
signal latched_smicro_vdata: std_logic;
signal not_fifo_empty: std_logic;
signal mem_used: std_logic_vector(15 downto 0);

signal rst_n: std_logic;
signal rst: std_logic;
signal clk: std_logic;
----------------------------------------
type STATE_TYPE is (IDLE, empiezaSYNC, SYNC, SCAN);
signal CS, NS: STATE_TYPE;
signal vidcap_counter: std_logic_vector(15 downto 0);
signal vidcap_counter_reset: std_logic;
signal vidcap_vert: std_logic_vector(10 downto 0);
signal vidcap_horz: std_logic_vector(10 downto 0);
signal vidcap_x: std_logic_vector(7 downto 0);
signal vidcap_y: std_logic_vector(7 downto 0);
signal vidcap_data: std_logic_vector (7 downto 0);
----------------------------------------
begin
---- duplicar reloj-----
rst <= not rst_l;
clkpad : IBUFG  port map (I=>CLK50, O=>CLKIN_w);

dll2x  : CLKDLL port map (CLKIN=>CLKIN_w,   CLKFB=>CLK2X_g, RST=>RST,
                          CLK0=>open,   CLK90=>open, CLK180=>open, CLK270=>open,
                           CLK2X=>CLK2X_dll, CLKDV=>open, LOCKED=>LOCKED2X);
clk2xg : BUFG   port map (I=>CLK2X_dll,   O=>CLK2X_g);
clk50g : BUFG   port map (I=>CLKIN_w,   O=>CLK);
CLK100 <= CLK2X_g;

      exaustivo: metodo_exaustivo1
      GENERIC MAP(4096,12,8)
      PORT MAP(
            address => maddress,
            data => micro_data,
            vdata => vdata,
            enable => enable,
            emite => emite,
            evento => evento,
            debug => debug,
            rst_n => rst_n,
            CLK => clk
      );

      aerout: aer_out PORT MAP(
            REQ_N => aer_out_req_l,
            ACK_N => saer_out_ack_l,
            Data_AER => aer_out_data,
            Data_Fifo => fifo_data_out,
            Enable_Fifo => not_fifo_empty,
            RD_Fifo => Fifo_rd,
            CLK => clk,
```

```
              RST_N => rst_n
       );

       fifo: ramfifo
--     GENERIC MAP(1024,10,16)
       GENERIC MAP(512,9,16)

       PORT MAP(
              clk => clk,
              wr => fifo_wr,
              rd => Fifo_rd,
              rst_n => RST_N,
              empty => Fifo_empty,
              full => Fifo_full,
              data_in => fifo_Data_In,
              data_out => fifo_Data_Out,
              mem_used => mem_used
       );

syncronizer: process(RST_N, CLK100)
begin
if (RST_N = '0') then
   counter <= (others =>'0');
       imicro_vdata <= '0';
       smicro_vdata <= '0';
       imicro_rw <= '0';
       smicro_rw <= '0';
       imicro_prog <= '0';
       smicro_prog <= '0';
       imicro_control <= '0';
       smicro_control <= '0';
       iaer_out_ack_l <= '0';
       saer_out_ack_l      <= '0';
       counter <= (others =>'0');
elsif(CLK100'event and CLK100 = '1') then
   counter <= counter + 1;
       imicro_vdata <= micro_vdata;
       smicro_vdata <= imicro_vdata;
       imicro_rw <= micro_rw;
       smicro_rw <= imicro_rw;
       imicro_control <= micro_control;
       smicro_control <= imicro_control;
       imicro_prog <= micro_prog;
       smicro_prog <= imicro_prog;
       iaer_out_ack_l <= aer_out_ack_l;
       saer_out_ack_l      <= iaer_out_ack_l;   -- Solo 1 biestable!
end if;
end process;
----------------------------------------------------------------------

vidcap_data <= aer_in_data(7 downto 0);
aer_in_ack_l <= counter(1);
vidcap_s: process (RST_N, CLK)
begin
if (RST_N = '0') then
       CS <= IDLE;
       vidcap_counter <= (others =>'0');
       vidcap_horz <= (others =>'0');
       vidcap_vert <= (others =>'0');
       vidcap_x <= (others =>'0');
       vidcap_y <= (others =>'0');
       maddress <= (others =>'0');
elsif (CLK'event and CLK='1') then
       maddress <= vidcap_y(5 downto 0) & vidcap_x (5 downto 0);
       micro_data <= vidcap_data;
       vdata <= '0';
       case CS is
              when IDLE =>
```

```
                        if (vidcap_data <5) then
                              CS <= empiezaSYNC;
                        else
                              CS <= IDLE;
                        end if;

                  when empiezaSYNC =>
                        vidcap_counter <=(others =>'0');
                        CS <= SYNC;

                  when SYNC =>
                        if (vidcap_data <5) then
                              CS <= SYNC;
                              vidcap_counter <= vidcap_counter + 1;
                        elsif (vidcap_counter <180) then
                              CS <= IDLE;
                        elsif(vidcap_counter <800) then  -- Sincronismo horizontal
                              CS <= SCAN;
                              vidcap_vert <= vidcap_vert + 1;
                              if (vidcap_vert > 20 and vidcap_vert(1 downto 0) ="00") then
                                    vidcap_y <= vidcap_y + 1;
                              end if;
                              vidcap_horz <= (others =>'0');
                              vidcap_x <= (others => '0');
                              vidcap_counter <= (others =>'0');
                        else                  -- Sincronismo vertical
                              CS <= IDLE;
                              vidcap_vert <= (others =>'0');
                              vidcap_horz <= (others =>'0');
                              vidcap_x <= (others =>'0');
                              vidcap_y <= (others =>'0');
                              vidcap_counter <= (others =>'0');
                        end if;

                  when SCAN =>
                        vidcap_counter <= vidcap_counter + 1;
                        if (vidcap_counter >500 and vidcap_counter(4 downto 0) = "00000"
                              and vidcap_vert > 20 and vidcap_vert(1 downto 0) ="00")then
                              vidcap_x <= vidcap_x + 1;
                              if(vidcap_x < 64 and vidcap_y < 64) then
                                    vdata <= '1';
                              end if;

                        end if;
                        if (vidcap_data <   5) then
                              CS <= empiezaSYNC;
                        end if;
      end case;
end if;
end process;

vidcap_c: process(CS, aer_in_data,vidcap_counter)
begin

end process;
-------------------------------------------------------------------------
fifo_wr <= emite and not fifo_full;
enable <= not fifo_full;
not_fifo_empty <= not fifo_empty;
led(2) <= counter(22);
led(1)<= aer_out_ack_l xor aer_in_req_l;
led(0) <= '1' when (aer_in_data = x"0000") else '0';
fifo_data_in(5 downto 0) <= evento(5 downto 0);
fifo_data_in(7 downto 6) <= (others => '0');
fifo_data_in(13 downto 8) <= evento(11 downto 6);
fifo_data_in(15 downto 14) <= (others => '0');
micro_busy <= '0';
--enable <= '1' when ((cfgmode='0') or (cfgmode='1' and micro_vdata = '1')) else '0';
```

```
micro_data <= (others =>'Z');

-- Deshabilitamos las RAMS
sram_oe_l <='1';
sram_we_l <= "1111";
address <= (others =>'0');
sram_data <= (others =>'0');
enable_out_buffers_l <= '0';
enable_in_buffers_l <= '0';
rst_n <= rst_l;
end Behavioral;
```

metodo_exaustivo1.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED."+";
use IEEE.STD_LOGIC_UNSIGNED."-";

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;


entity metodo_exaustivo1 is
generic(RAM_Size: in integer := 4096; RAM_Addr_Size: in integer := 12;  RAM_Data_Size:
in integer := 8);
--generic(RAM_Size: in integer := 64; RAM_Addr_Size: in integer := 6;   RAM_Data_Size:
in integer := 8);
    Port ( address : in std_logic_vector(RAM_Addr_Size -1 downto 0);
           data : in std_logic_vector(RAM_Data_Size-1 downto 0);
                     vdata: in std_logic;
                     enable: in std_logic;
                   emite : out std_logic;
                     evento: out std_logic_vector(RAM_Addr_Size-1 downto 0);
                     debug: out std_logic_vector(8 downto 0);
                     RST_N: in std_logic;
                     CLK: in std_logic
                     );
end metodo_exaustivo1;
architecture Behavioral of metodo_exaustivo1 is

      COMPONENT dualram
      generic (TAM: in integer:= RAM_Size; IL: in integer:=RAM_Addr_Size; WL: in
integer:=RAM_Data_Size);
      PORT(
            clk : IN std_logic;
            wr   : in std_logic;
            index_i : in std_logic_vector(IL-1 downto 0);
            index_o : in std_logic_vector(IL-1 downto 0);
            word_i   : in std_logic_vector(WL-1 downto 0);
            word_o  : out std_logic_vector(WL-1 downto 0)
            );
      END COMPONENT;
signal image_out: std_logic_vector(RAM_Data_Size-1 downto 0);
signal maddress: std_logic_vector(RAM_Addr_Size-1 downto 0);

signal iaddress: std_logic_vector(RAM_Addr_Size-1 downto 0);
signal numitera: std_logic_vector(RAM_Data_Size-1 downto 0);
signal levento: std_logic_vector(RAM_Addr_Size-1 downto 0);
signal contador: std_logic_vector(RAM_Data_Size+RAM_Addr_Size -1 downto 0);
signal dcontador: std_logic_vector(RAM_Data_Size+RAM_Addr_Size -1 downto 0);
signal LFSR: std_logic_vector(15 downto 0);

begin
```

```
imagen: dualram PORT MAP(
             clk => clk,
             wr => vdata,
             index_i => address,
             index_o => maddress,
             word_i => data,
             word_o => image_out
             );

invertir: process(dcontador)
begin
for i in 0 to RAM_Data_Size-1 loop
       numitera(RAM_Data_Size-1-i) <= dcontador (RAM_Addr_Size+i);
end loop;
end process;

icounter: process(rst_n, clk)
begin
if (rst_n = '0') then
       contador <= (others=>'0');
       dcontador <= (others =>'1');
elsif(CLK'event and CLK='1') then
       if (enable ='1') then
              contador <= contador      + 1;
              dcontador <= contador;
       end if;
end if;
end process;
evento <= dcontador(RAM_Addr_Size -1 downto 0);
iaddress <= contador (RAM_Addr_Size -1 downto 0);
exaustivo: process(enable, contador, image_out, numitera,dcontador)
begin
       if(enable = '1') then
              maddress <= contador (RAM_Addr_Size -1 downto 0);
              if (image_out>numitera) then
                     emite <='1';
              else
                     emite <= '0';
              end if;
       else
              maddress <= dcontador(RAM_Addr_Size -1 downto 0);
              emite <= '0';
       end if;
end process;
end Behavioral;
```

aer_out.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity aer_out is
    Port (REQ_N : out std_logic;
          ACK_N : in std_logic;
          Data_AER : out std_logic_vector(15 downto 0);
          Data_Fifo : in std_logic_vector(15 downto 0);
          Enable_Fifo : in std_logic;
          RD_Fifo: out std_logic;
         CLK : in std_logic;
          RST_N : in std_logic);
end aer_out;
architecture Behavioral of aer_out is
type STATE_TYPE is (IDLE, SendDATA);
signal CS, NS: STATE_TYPE;
begin
    SYNC_PROC: process (CLK, RST_N)
       begin
       if (RST_N='0') then
```

```
            CS <= IDLE;
      elsif (CLK'event and CLK = '1') then
            CS <= NS;
      end if;
    end process;
    COMB_PROC: process (CS, Enable_Fifo, ack_n)
    begin
    case CS is
            when IDLE =>
                  REQ_N <= '1';
                        if (Enable_Fifo ='1' and ACK_N='1') then
                        NS <= SendDATA;
                  else
                        NS <= IDLE;
                  end if;
                  RD_Fifo <= '0';
            when SendDATA =>
                  REQ_N <='0';
                  if (ACK_N='0') then
                        NS <= IDLE;
                        RD_Fifo <= '1';
                  else
                        NS <= SendDATA;
                        RD_Fifo <= '0';
                  end if;
    end case;
      end process;
Data_AER <= Data_Fifo;
end Behavioral;
```

ramfifo.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ramfifo is
generic (TAM: in integer:= 64; IL: in integer:=6; WL: in integer:=16);
 port (clk  : in std_logic;
      wr   : in std_logic;
      rd    : in std_logic;
      rst_n: in std_logic;
      empty: out std_logic;
      full: out std_logic;
      data_in   : in std_logic_vector(WL-1 downto 0);
      data_out  : out std_logic_vector(WL-1 downto 0);
      mem_used : out std_logic_vector(IL-1 downto 0));
end ramfifo;

architecture syn of ramfifo is

      COMPONENT dualram
      generic (TAM: in integer:= TAM; IL: in integer:=IL; WL: in integer:=WL);
      PORT(
            clk : IN std_logic;
            wr   : in std_logic;
            index_i : in std_logic_vector(IL-1 downto 0);
            index_o : in std_logic_vector(IL-1 downto 0);
            word_i   : in std_logic_vector(WL-1 downto 0);
            word_o  : out std_logic_vector(WL-1 downto 0)
            );
      END COMPONENT;

      SIGNAL index_i :  std_logic_vector(IL-1 downto 0);
      SIGNAL index_o :  std_logic_vector(IL-1 downto 0);
      SIGNAL iempty: std_logic;
      SIGNAL ifull: std_logic;
```

```
	SIGNAL ramwr: std_logic;
	SIGNAL memused: std_logic_vector(IL downto 0);
	SIGNAL dout: std_logic_vector(WL-1 downto 0);

BEGIN

	uut: dualram generic map (TAM,IL,WL)
	PORT MAP(
		clk => clk,
		wr => ramwr,
		index_i => index_i,
		index_o => index_o,
		word_i => data_in,
		word_o => dout
		);

process (clk, rst_n)
begin
	if (rst_n ='0') then
		index_i <= (others=>'0');
		index_o <= (others=>'0');
		memused <= (others=>'0');
	elsif (clk'event and clk = '1') then
	if (wr = '1' and rd = '1') then
			index_i <= index_i + 1;
			index_o <= index_o + 1 ;
			--memused <= memused; -- Al ser proceso sincrono y no cambiar
memused no es necesaria esta linea.
		elsif (wr = '1' and ifull = '0') then
			index_i <= index_i + 1;
			memused <= memused + 1;
		elsif (rd = '1' and iempty = '0') then
			index_o <= index_o +1;
			memused <= memused - 1;
		end if;
	end if;
 end process;

 --ramindex <= index_o when (rd ='1') else index_o;

 iempty <= '1' when memused = 0 else '0';
 ifull <= '1' when  memused = TAM-1  else '0';
 empty <= iempty;
 full <= ifull;
 ramwr <= '1' when (wr = '1' and (ifull='0' or rd='1')) else '0';
  data_out <= dout;-- when (iempty='0') else (others => 'Z'); -- Puede eliminarse y
puentearse la salida de dualram con la salida de la fifo.
 mem_used <= memused(IL-1 downto 0);
 end syn;
```

dualram.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity dualram is
generic (TAM: in integer:= 64; IL: in integer:=6; WL: in integer:=16);
 port (clk  : in std_logic;
	wr   : in std_logic;
	index_i : in std_logic_vector(IL-1 downto 0);
	index_o : in std_logic_vector(IL-1 downto 0);
	word_i   : in std_logic_vector(WL-1 downto 0);
	word_o  : out std_logic_vector(WL-1 downto 0));
end dualram;
```

```
-- Only XST supports RAM inference
-- Infers Dual Port Distributed Ram

 architecture syn of dualram is
 type ram_type is array (TAM-1 downto 0) of std_logic_vector (WL-1 downto 0);
 signal RAM : ram_type;
 signal indice_lectura: std_logic_vector(IL-1 downto 0);

 begin
 process (clk)
 begin
      if (clk'event and clk = '1') then
            if (wr = '1') then
                  RAM(conv_integer(index_i)) <= word_i;
            end if;
            indice_lectura <= index_o;
      end if;
 end process;
                           --indice_lectura <= index_o;
 word_o <= RAM(conv_integer(indice_lectura));


 end syn;
```

11.4.3 Retina sintética derivativa

mapper_genera_exaustivo.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity mapper_genera_exaustivo is
      Port (
            clk50 : in std_logic;
            rst_l : in std_logic;
            led : out std_logic_vector( 2 downto 0 );

            -- AER input
            aer_in_data : in std_logic_vector( 15 downto 0 );
            aer_in_req_l : in std_logic;
            aer_in_ack_l : out std_logic;

            -- AER output
            aer_out_req_l : out std_logic;
            aer_out_ack_l : in std_logic;
            aer_out_data : out std_logic_vector( 15 downto 0 );

            -- Micro interface
            micro_vdata : in std_logic;
            micro_prog : in std_logic;
            micro_control: in std_logic;     -- Aux0 en sch
            micro_rw: in std_logic;
            micro_busy:out std_logic;
            micro_data:  inout std_logic_vector( 7 downto 0 );

            -- SRAM interface
            sram_oe_l : out std_logic;
            sram_we_l : out std_logic_vector(3 downto 0);
            sram_address : out std_logic_vector( 18 downto 0 );
            sram_data : inout std_logic_vector( 31 downto 0 );


            -- Other control lines
            enable_out_buffers_l : out std_logic;
            enable_in_buffers_l: out std_logic
      );
end mapper_genera_exaustivo;

architecture Behavioral of mapper_genera_exaustivo is

--- duplicar reloj ---
component IBUFG
port (
   O : out std_logic;
   I : in std_logic
 );
end component;


component CLKDLL
 port (
   CLK0 : out std_logic;
   CLK90 : out std_logic;
```

```
   CLK180 : out std_logic;
   CLK270 : out std_logic;
   CLK2X : out std_logic;
   CLKDV : out std_logic;
   LOCKED : out std_logic;
   CLKIN : in std_logic;
   CLKFB : in std_logic;
   RST : in std_logic
 );
end component;

component BUFG
 port (
   O : out std_logic;
   I : in std_logic
 );
end component;
signal CLKIN_w, RESET_w, CLK2X_dll, CLK2X_g,CLK100 : std_logic;
signal LOCKED2X, LOCKED2X_delay: std_logic;

---- fin duplicar reloj ----


      COMPONENT metodo_exaustivo1
      generic(RAM_Size: in integer; RAM_Addr_Size: in integer;   RAM_Data_Size:     in
integer);
      PORT(
            address : IN std_logic_vector(RAM_Addr_Size -1 downto 0);
            data : IN std_logic_vector(RAM_Data_Size-1 downto 0);
            cfgmode : IN std_logic;
            enable : IN std_logic;
            CLK : IN std_logic;
            emite : OUT std_logic;
            evento : OUT std_logic_vector(RAM_Addr_Size-1 downto 0);
            signo: out std_logic;
            rst_n: in std_logic;
            derivativa: in std_logic;
            RAM_ADDRESS : out std_logic_vector(18 downto 0);
      RAM_DATA : inout std_logic_vector(31 downto 0);
      RAM_OE : out std_logic;
      RAM_WE : out std_logic_vector(3 downto 0);
            debug : OUT std_logic_vector(8 downto 0)
            );
      END COMPONENT;

      COMPONENT aer_out
      PORT(
            REQ_N : OUT std_logic;
            ACK_N : IN std_logic;
            Data_AER : OUT std_logic_vector(15 downto 0);
            Data_Fifo : IN std_logic_vector(15 downto 0);
            Enable_Fifo : IN std_logic;
            RD_Fifo : OUT std_logic;
            CLK : IN std_logic;
            RST_N : IN std_logic
            );
      END COMPONENT;


      COMPONENT ramfifo
      generic (TAM: in integer; IL: in integer; WL: in integer);
      PORT(
            clk : IN std_logic;
            wr : IN std_logic;
            rd : IN std_logic;
            rst_n : IN std_logic;
            empty : OUT std_logic;
            full : OUT std_logic;
```

```
            data_in : IN std_logic_vector(WL-1 downto 0);
            data_out : OUT std_logic_vector(WL-1 downto 0);
            mem_used: out std_logic_vector(IL-1 downto 0)

            );
      END COMPONENT;

signal imicro_prog, smicro_prog: std_logic;
signal imicro_vdata, smicro_vdata: std_logic;
signal imicro_control, smicro_control: std_logic;
signal imicro_rw, smicro_rw: std_logic;
signal iaer_in_req_l,  saer_in_req_l: std_logic;
signal iaer_out_ack_l, saer_out_ack_l: std_logic;

signal counter: std_logic_vector(23 downto 0);

SIGNAL vidcap_address :  std_logic_vector(13 downto 0);
SIGNAL cfgmode :  std_logic;
SIGNAL vdata :  std_logic;
SIGNAL enable :  std_logic;
SIGNAL emite :  std_logic;
SIGNAL evento :  std_logic_vector(13 downto 0);
SIGNAL signo: std_logic;
SIGNAL debug :  std_logic_vector(8 downto 0);

signal fifo_wr, fifo_rd, fifo_rd_2, fifo_empty, fifo_full: std_logic;
signal fifo_data_in, fifo_data_out: std_logic_vector(15 downto 0);
signal latched_smicro_vdata: std_logic;

signal not_fifo_empty: std_logic;

signal mem_used: std_logic_vector(8 downto 0);


signal rst_n: std_logic;
signal rst: std_logic;
signal clk: std_logic;
signal derivativa: std_logic;

----------------------------------------
--
type STATE_TYPE is (IDLE, empiezaSYNC, SYNC, SCAN);
signal CS, NS: STATE_TYPE;
signal vidcap_counter: std_logic_vector(15 downto 0);
signal vidcap_counter_reset: std_logic;
signal vidcap_vert: std_logic_vector(10 downto 0);
signal vidcap_horz: std_logic_vector(10 downto 0);
signal vidcap_x: std_logic_vector(7 downto 0);
signal vidcap_y: std_logic_vector(7 downto 0);
signal vidcap_data: std_logic_vector (7 downto 0);

--
----------------------------------------

begin

---- duplicar reloj-----
rst <= not rst_n;
clkpad : IBUFG  port map (I=>CLK50, O=>CLKIN_w);

dll2x  : CLKDLL port map (CLKIN=>CLKIN_w,   CLKFB=>CLK2X_g, RST=>RST,
                          CLK0=>open,   CLK90=>open, CLK180=>open, CLK270=>open,
                           CLK2X=>CLK2X_dll, CLKDV=>open, LOCKED=>LOCKED2X);

clk2xg : BUFG   port map (I=>CLK2X_dll,   O=>CLK2X_g);
clk50g : BUFG   port map (I=>CLKIN_w,   O=>open); --CLK
--CLK <= CLK2X_g;
```

```
CLK100 <= CLK2X_g;
CLK <= CLKIN_w;
rst_n <= not micro_prog;

	exaustivo: metodo_exaustivo1
	GENERIC MAP(16384,14,8)
	PORT MAP(
		address => vidcap_address,
		data => vidcap_data,
		cfgmode => cfgmode,
		enable => enable,
		emite => emite,
		evento => evento,
		signo => signo,
		debug => debug,
		derivativa => derivativa,
		rst_n => rst_n,
		ram_address => sram_address,
		ram_data => sram_data,
		ram_oe => sram_oe_l,
		ram_we => sram_we_l,
		CLK => clk
	);

	aerout: aer_out PORT MAP(
		REQ_N => aer_out_req_l,
		ACK_N => saer_out_ack_l,
		Data_AER => aer_out_data,
		Data_Fifo => fifo_data_out,
		Enable_Fifo => not_fifo_empty,
		RD_Fifo => Fifo_rd,
		CLK => clk,
		RST_N => rst_n
	);

	fifo: ramfifo
	---GENERIC MAP(1024,10,16)
   GENERIC MAP(512,9,16)

	PORT MAP(
		clk => clk,
		wr => fifo_wr,
		rd => Fifo_rd,
		rst_n => RST_N,
		empty => Fifo_empty,
		full => Fifo_full,
		data_in => fifo_Data_In,
		data_out => fifo_Data_Out,
		mem_used => mem_used
	);

syncronizer: process(RST_N, CLK100)
begin
if (RST_N = '0') then
	iaer_out_ack_l <= '0';
	saer_out_ack_l	<= '0';
elsif(CLK100'event and CLK100 = '1') then
	iaer_out_ack_l <= aer_out_ack_l;
	saer_out_ack_l	<= iaer_out_ack_l;   -- Solo 1 biestable!
end if;
end process;
-----------------------------------------------------------------------
-----------------------------------------------------------------------


vidcap_data <= aer_in_data(7 downto 0);
aer_in_ack_l <= counter(1);                    -- CLK para el ADC
```

```
vidcap_s: process (RST_N, CLK)
begin
if (RST_N = '0') then
	CS <= IDLE;
	counter <= (others =>'0');
	vidcap_counter <= (others =>'0');
	vidcap_horz <= (others =>'0');
	vidcap_vert <= (others =>'0');
	vidcap_x <= (others =>'0');
	vidcap_y <= (others =>'0');
	vidcap_address <= (others =>'0');
	cfgmode <= '0';
elsif (CLK'event and CLK='1') then
	counter <= counter + 1;
	vidcap_address <= vidcap_y(6 downto 0) & vidcap_x (6 downto 0);
	cfgmode <= '0';
	case CS is
		when IDLE =>
			if (vidcap_data <10) then
				CS <= empiezaSYNC;
			else
				CS <= IDLE;
			end if;

		when empiezaSYNC =>
			vidcap_counter <=(others =>'0');
			CS <= SYNC;
			when SYNC =>
			if (vidcap_data <10) then	-- Contamos el tiempo sync activo
				CS <= SYNC;
				vidcap_counter <= vidcap_counter + 1;
			elsif (vidcap_counter <180) then -- Falso sincronismo??
				CS <= IDLE;
			elsif(vidcap_counter <800) then
				CS <= SCAN;
				vidcap_vert <= vidcap_vert + 1;
				if (vidcap_vert > 20 and vidcap_vert(0) ='0') then
					vidcap_y <= vidcap_y + 1; --	vidcap_y	linea	en
destino
				end if;
				vidcap_horz <= (others =>'0');
				vidcap_x <= (others => '0');
				vidcap_counter <= (others =>'0');
			else			-- Sincronismo vertical
				CS <= IDLE;
				vidcap_vert <= (others =>'0');
				vidcap_horz <= (others =>'0');
				vidcap_x <= (others =>'0');
				vidcap_y <= (others =>'0');
				vidcap_counter <= (others =>'0');
			end if;

		when SCAN =>
			vidcap_counter <= vidcap_counter + 1;
			if (vidcap_counter >500 and vidcap_counter(3 downto 0) = "0000"
					and vidcap_vert > 20 and vidcap_vert(0) ='0')then
				vidcap_x <= vidcap_x + 1;
				if(vidcap_x < 128 and vidcap_y < 128) then
					cfgmode <= '1';	--- '1'
				end if;
			else
				cfgmode <= '0';
			end if;
			if (vidcap_data <  10) then
				CS <= IDLE;
			end if;
	end case;
```

```
end if;
end process;



fifo_wr <= emite and not fifo_full;
enable <= not fifo_full;

--vdata <= smicro_vdata;
not_fifo_empty <= not fifo_empty;

led(2) <= counter(22);
led(1)<= aer_out_ack_l;
led(0) <= '1' when (aer_in_data = x"0000") else '0';

fifo_data_in(7 downto 1) <= evento(6 downto 0);
fifo_data_in(0) <= signo;
fifo_data_in(14 downto 8) <= evento(13 downto 7);
fifo_data_in(15) <= '0';

--enable <= '1' when ((cfgmode='0') or (cfgmode='1' and micro_vdata = '1')) else '0';

micro_busy <= '0';
micro_data <= (others =>'Z');

enable_out_buffers_l <= '0';
enable_in_buffers_l <= '0';
derivativa <= rst_l;

end Behavioral;
```

metodo_exaustivo1.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED."+";
use IEEE.STD_LOGIC_UNSIGNED."-";

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;


entity metodo_exaustivo1 is
generic(RAM_Size: in integer := 16384; RAM_Addr_Size: in integer := 14; RAM_Data_Size:
in integer := 8);
--generic(RAM_Size: in integer := 64; RAM_Addr_Size: in integer := 6;   RAM_Data_Size:
in integer := 8);
    Port ( address : in std_logic_vector(RAM_Addr_Size -1 downto 0);
           data : in std_logic_vector(RAM_Data_Size-1 downto 0);
                     cfgmode: in std_logic;
                     enable: in std_logic;
           emite : out std_logic;
                     evento: out std_logic_vector(RAM_Addr_Size-1 downto 0);
                     signo: out std_logic;
                     derivativa: in std_logic;
                     debug: out std_logic_vector(8 downto 0);
                     RAM_ADDRESS : out std_logic_vector(18 downto 0);
           RAM_DATA : inout std_logic_vector(31 downto 0);
           RAM_OE : out std_logic;
           RAM_WE : out std_logic_vector(3 downto 0);
                     RST_N: in std_logic;
                     CLK: in std_logic
                     );
end metodo_exaustivo1;

architecture Behavioral of metodo_exaustivo1 is

signal image_we: std_logic;
signal image_in:  std_logic_vector(RAM_Data_Size-1 downto 0);
signal image_out: std_logic_vector(RAM_Data_Size-1 downto 0);

signal maddress: std_logic_vector(RAM_Addr_Size-1 downto 0);


signal iaddress: std_logic_vector(RAM_Addr_Size-1 downto 0);
signal numitera: std_logic_vector(RAM_Data_Size-1 downto 0);

signal levento: std_logic_vector(RAM_Addr_Size-1 downto 0);

signal contador: std_logic_vector(RAM_Data_Size+RAM_Addr_Size -1 downto 0);
signal dcontador: std_logic_vector(RAM_Data_Size+RAM_Addr_Size -1 downto 0);
signal LFSR: std_logic_vector(15 downto 0);

signal miRAM_Address: std_logic_vector(RAM_Addr_Size -1 downto 0);
signal miRAM_Data: std_logic_vector(RAM_Data_Size-1 downto 0);
signal miCFG_Mode: std_logic;
signal miWrite, miWrite2: std_logic;
signal rafa: std_logic_vector(7 downto 0);
signal rafa_signo: std_logic;
begin

latcher: process(rst_n, clk)
begin
if(rst_n = '0') then
      miRAM_Address <= (others =>'0');
```

```
	miRAM_Data <= (others =>'0');
	miCFG_Mode <= '0';
	miWrite <= '0';
	miWrite2 <= '0';
elsif (clk'event and clk='1') then
	if(cfgmode='1') then
		miRAM_Address <= address;
		miRAM_Data <= data;
		miCFG_Mode <= '1';
		miWrite <= '0';
		miWrite2 <= '0';
	elsif (miCFG_Mode = '1') then
		miWrite <= '1';
		miCFG_Mode <= '0';
		miWrite2 <= '0';
	elsif (miWrite = '1') then
		miWrite <='0';
		miWrite2 <= '1';
	else
		miWrite2 <= '0';
	end if;
end if;
end process;


--lfsr_counter: process(cfgmode, CLK, LFSR)
--begin
--if(cfgmode = '1') then
--	LFSR <= (others =>'1');		-- Iniciación, siempre distinto de todos a
cero.
--elsif (CLK'event and CLK='1') then
--	for i in 15 downto 1 loop
--		LFSR(i) <= LFSR(i-1);
--	end loop;
--	LFSR(0) <= LFSR(15) xor LFSR(14) xor LFSR(12) xor LFSR(3); --	El	polinomio
empieza en 0, 1 menos!
--end if;
--end process;

invertir: process(contador)
begin
for i in 0 to RAM_Data_Size-1 loop
	numitera(RAM_Data_Size-1-i) <= dcontador (RAM_Addr_Size+i);
end loop;
end process;



icounter: process(rst_n, cfgmode, CLK, contador, enable, image_out, numitera, levento)
begin
if (rst_n = '0') then
	contador <= (others=>'0');
	dcontador <= (others =>'1');
	levento <= (others =>'0');
--	emite <= '0';
elsif(CLK'event and CLK='1') then
	if (enable ='1') then
		contador <= contador	+ 1;
		dcontador <= contador;
	end if;
end if;
evento <= dcontador(RAM_Addr_Size -1 downto 0);
iaddress <= dcontador (RAM_Addr_Size -1 downto 0); -- contador
end process;



rafa <= miRAM_data-RAM_DATA(15 downto 8);
```

```
exaustivo: process(rst_n, data, cfgmode, address, iaddress, enable)
begin

if (rst_n = '0') then
       RAM_DATA <= (others =>'Z');
       RAM_WE <= "1111";
       emite <= '0';
       maddress <= (others =>'0');
elsif  (miWrite = '1') then
       RAM_DATA(31 downto 8) <= (others =>'Z');
       if(rafa(7)='0') then
              RAM_DATA(7 downto 0) <= rafa;
              RAM_DATA(16) <= '0';
       else
              RAM_DATA(7 downto 0) <= x"ff"-rafa;
              RAM_DATA(16) <= '1';
       end if;
       RAM_WE <= "1010";
       emite <= '0';
       maddress <= miRAM_address;
elsif(miWrite2 = '1') then
       RAM_DATA(7 downto 0) <= (others =>'Z');
       RAM_DATA(15 downto 8) <= miRAM_data;
       RAM_DATA(31 downto 16) <= (others =>'Z');
       RAM_WE <= "1101";
       emite <= '0';
       maddress <= miRAM_address;
else
       RAM_DATA <= (others => 'Z');
       RAM_WE <= "1111";
       if(enable = '1') then
              maddress <= dcontador (RAM_Addr_Size -1 downto 0); -- contador
              if (image_out>numitera(7 downto 0)) then
                     emite <='1';
              else
                     emite <= '0';
              end if;
       else
              maddress <= dcontador(RAM_Addr_Size -1 downto 0);
              emite <= '0';
       end if;
end if;
end process;

RAM_ADDRESS <= maddress;

--process (derivativa, RAM_DATA)
--begin
--     if (derivativa='1') then
--            image_out <= RAM_DATA(7 downto 0);
--     else
--            image_out <= RAM_DATA(15 downto 8);
--     end if;
--end process;

image_out <= RAM_DATA(7 downto 0) when derivativa ='1' else  RAM_DATA(15 downto 8);
signo <= RAM_DATA(16);

--image_out <= RAM_DATA(7 downto 0);    -- DERIVATIVA
--image_out <= RAM_DATA(15 downto 8);   -- NORMAL


end Behavioral;
```

11.4.4 Mapeador básico

mapper_function.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Mapper_function is
    Port ( CLK : in std_logic;
           RST_N : in std_logic;
           ENABLE_N : in std_logic;
           AER_IN_DATA : in std_logic_vector(15 downto 0);
           AER_IN_REQ_L : in std_logic;
           AER_IN_ACK_L : out std_logic;
           AER_OUT_DATA : out std_logic_vector(15 downto 0);
           AER_OUT_REQ_L : out std_logic;
           AER_OUT_ACK_L : in std_logic;
           RAM_ADDRESS : out std_logic_vector(18 downto 0);
           RAM_DATA : inout std_logic_vector(31 downto 0);
           RAM_OE : out std_logic;
           RAM_WE : out std_logic_vector(3 downto 0);
           LED : out std_logic_vector(2 downto 0)
                 );
end Mapper_function;

architecture Behavioral of Mapper_function is
type states is (IDLE, WAIT_REQ_L, WAIT_REQ_H, READ_RAM, SEND_EVENT, SUBE_ACK );
signal CS,NS: states;
signal latched_input: std_logic_vector(15 downto 0);


begin

SYNC: process(RST_N, enable_n, CLK)
begin
      if (RST_N = '0' or ENABLE_N = '1') then
            CS <= IDLE;
      elsif(CLK'event and CLK ='1') then
            CS <= NS;
      end if;
end process;


COMB: process(CS, AER_IN_REQ_L, AER_OUT_ACK_L, RAM_DATA, latched_input)
begin
case CS is
      when IDLE =>
                                        NS <= WAIT_REQ_L;
                                        AER_IN_ACK_L <= '1';
                                        AER_OUT_REQ_L <= '1';
                                        AER_OUT_DATA <= (others =>'Z');
                                        RAM_ADDRESS <= (others =>'Z');
                                        RAM_OE <= 'Z';
                                        RAM_WE <= "ZZZZ";
                                        led <= "000";

      when WAIT_REQ_L =>
```

```
                        if (AER_IN_REQ_L = '0') then
                              NS <= WAIT_REQ_H;
                        else
                              NS <= WAIT_REQ_L;
                        end if;
                        AER_IN_ACK_L <= '1';
                        AER_OUT_REQ_L <= '1';
                        AER_OUT_DATA <= RAM_DATA(15 downto 0);
                        RAM_ADDRESS <= "000" & latched_input;
                        RAM_OE <= '0';
                        RAM_WE <= "1111";
                        led <= "001";

      when WAIT_REQ_H =>
                        if     (AER_IN_REQ_L = '0') then
                              NS <= WAIT_REQ_H;
                        else
                              NS <= READ_RAM;
                        end if;
                        AER_IN_ACK_L <= '0';
                        AER_OUT_REQ_L <= '1';
                        AER_OUT_DATA <= RAM_DATA(15 downto 0);
                        RAM_ADDRESS <= "000" & latched_input;
                        RAM_OE <= '0';
                        RAM_WE <= "1111";
                        led <= "011";

      when READ_RAM =>
                        if (AER_OUT_ACK_L = '0') then
                              NS <= READ_RAM;
                        else
                              NS <= SEND_EVENT;
                        end if;
                        AER_IN_ACK_L <= '0';
                        AER_OUT_REQ_L <= '1';
                        AER_OUT_DATA <= RAM_DATA(15 downto 0);
                        RAM_ADDRESS <= "000" & latched_input;
                        RAM_OE <= '0';
                        RAM_WE <= "1111";
                        led <= "100";

      when SEND_EVENT =>
                        if (AER_OUT_ACK_L = '1') then
                              NS <= SEND_EVENT;
                        else
                              NS <= SUBE_ACK;
                        end if;
                        AER_IN_ACK_L <= '0';
                        AER_OUT_REQ_L <= '0';
                        AER_OUT_DATA <= RAM_DATA(15 downto 0);
                        RAM_ADDRESS <= "000" & latched_input;
                        RAM_OE <= '0';
                        RAM_WE <= "1111";
                        led <= "101";

      when SUBE_ACK =>                                  --    Tambien    se    podria
comprobar ACK_OUT aqui
                        NS <= WAIT_REQ_L;
                        AER_IN_ACK_L <= '1';
                        AER_OUT_REQ_L <= '1';
                        AER_OUT_DATA <= RAM_DATA(15 downto 0);
                        RAM_ADDRESS <= "000" & latched_input;
                        RAM_OE <= '0';
                        RAM_WE <= "1111";
                        led <= "111";

      end case;
end process;
```

```
SYCN: process( RST_N, CLK, CS, NS, AER_IN_DATA)
begin
if (RST_N = '0')  then
       latched_input <= (others =>'0');
elsif(CLK'event and CLK='1') then
       if (CS = WAIT_REQ_L and NS = WAIT_REQ_H)  then
              latched_input <= AER_IN_DATA;
       end if;
end if;
end process;

RAM_DATA <= (others =>'Z');


end Behavioral;
```

usb_aer.vhdl

```
--Test Mapper
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity usb_aer is
       Port (
              clk : in std_logic;
              rst_l : in std_logic;
              led : out std_logic_vector( 2 downto 0 );

              -- AER input
              aer_in_data : in std_logic_vector( 15 downto 0 );
              aer_in_req_l : in std_logic;
              aer_in_ack_l : out std_logic;

              -- AER output
              aer_out_req_l : out std_logic;
              aer_out_ack_l : in std_logic;
              aer_out_data : out std_logic_vector( 15 downto 0 );

              -- SRAM interface
              sram_oe_l : out std_logic;
              sram_we_l : out std_logic_vector(3 downto 0);
              address : out std_logic_vector( 18 downto 0 );
              sram_data : inout std_logic_vector( 31 downto 0 );

              -- Micro interface
              micro_vdata : in std_logic;
              micro_prog : in std_logic;
              micro_control: in std_logic;     -- Aux0 en sch
              micro_rw: in std_logic;
              micro_busy:out std_logic;
              micro_data:  inout std_logic_vector( 7 downto 0 );

              -- Other control lines
              enable_out_buffers_l : out std_logic;
              enable_in_buffers_l: out std_logic
       );
end usb_aer;

architecture Behavioral of usb_aer is

COMPONENT program_ram
    PORT ( RST_N : in std_logic;
```

```
            CLK : in std_logic;
            ENABLE_N : in std_logic;
            DATA_VALID : in std_logic;
            DATA : inout std_logic_vector(7 downto 0);
            BUSY : out std_logic;
                       CONTROL: in std_logic;
                       READWRITE: in std_logic;
            RAM_ADDRESS : out std_logic_vector(18 downto 0);
            RAM_DATA : inout std_logic_vector(31 downto 0);
            RAM_OE : out std_logic;
            RAM_WE : out std_logic_vector(3 downto 0);
            LED : out std_logic_vector(2 downto 0));
end component;

COMPONENT Mapper_function
    PORT ( CLK : in std_logic;
            RST_N : in std_logic;
            ENABLE_N : in std_logic;
            AER_IN_DATA : in std_logic_vector(15 downto 0);
            AER_IN_REQ_L : in std_logic;
            AER_IN_ACK_L : out std_logic;
            AER_OUT_DATA : out std_logic_vector(15 downto 0);
            AER_OUT_REQ_L : out std_logic;
            AER_OUT_ACK_L : in std_logic;
            RAM_ADDRESS : out std_logic_vector(18 downto 0);
            RAM_DATA : inout std_logic_vector(31 downto 0);
            RAM_OE : out std_logic;
            RAM_WE : out std_logic_vector(3 downto 0);
            LED : out std_logic_vector(2 downto 0)
                  );
end component;


signal mled: std_logic_vector(2 downto 0);
signal pled: std_logic_vector(2 downto 0);
signal not_micro_prog: std_logic;
signal direccion: std_logic_vector(18 downto 0);

signal imicro_prog, smicro_prog: std_logic;
signal imicro_vdata, smicro_vdata: std_logic;
signal imicro_control, smicro_control: std_logic;
signal imicro_rw, smicro_rw: std_logic;
signal iaer_in_req_l,  saer_in_req_l: std_logic;
signal iaer_out_ack_l, saer_out_ack_l: std_logic;

signal miCLK: std_logic;
signal counter: std_logic_vector(23 downto 0);




begin

ControlRAM: program_ram PORT MAP (
                        RST_N  => RST_L,
                  CLK          => miCLK,
                        ENABLE_N => not_micro_prog,
                        DATA_VALID => smicro_vdata,
                        DATA         => micro_data,
                        BUSY         => micro_busy,
                        CONTROL  => smicro_control,
                        READWRITE =>smicro_rw,
                        RAM_ADDRESS => address,
                        RAM_DATA => sram_data,
                        RAM_OE       => sram_oe_l,
                        RAM_WE       => sram_we_l,
                        LED          => pled
                        );
```

```
Mapper: mapper_function PORT MAP(
                           CLK          => miCLK,
                           RST_N  => RST_L,
                           ENABLE_N => smicro_prog,
                           AER_IN_DATA => AER_IN_DATA,
                           AER_IN_REQ_L => sAER_IN_REQ_L,
                           AER_IN_ACK_L => aer_in_ack_l,
                           AER_OUT_DATA => aer_out_data,
                           AER_OUT_REQ_L => aer_out_req_l,
                           AER_OUT_ACK_L => sAER_OUT_ACK_L,
                           RAM_ADDRESS => address,
                           RAM_DATA => sram_data,
                           RAM_OE       => sram_oe_l,
                           RAM_WE => sram_we_l,
                           LED => mled
                           );

syncronizer: process(RST_L, CLK)
begin
if (RST_L = '0') then
       imicro_vdata <= '0';
       smicro_vdata <= '0';
       imicro_rw <= '0';
       smicro_rw <= '0';
       imicro_prog <= '0';
       smicro_prog <= '0';
       imicro_control <= '0';
       smicro_control <= '0';
       iaer_in_req_l <= '1';
       saer_in_req_l <= '1';
       iaer_out_ack_l <= '1';
       saer_out_ack_l      <= '1';
       counter <= (others =>'0');
elsif(CLK'event and CLK = '1') then
       imicro_vdata <= micro_vdata;
       smicro_vdata <= imicro_vdata;
       imicro_rw <= micro_rw;
       smicro_rw <= imicro_rw;
       imicro_control <= micro_control;
       smicro_control <= imicro_control;
       imicro_prog <= micro_prog;
       smicro_prog <= imicro_prog;
       iaer_in_req_l <= aer_in_req_l;
       saer_in_req_l <= iaer_in_req_l;
       iaer_out_ack_l <= aer_out_ack_l;
       saer_out_ack_l      <= iaer_out_ack_l;
       counter <= counter + 1;
end if;
end process;

divisor: process(RST_L, CLK)
begin
if (RST_L ='0') then
       counter <= (others =>'0');
elsif(CLK'event and CLK='1') then
       counter <= counter +1;
end if;
end process;

miCLK <= CLK;

led(2) <= pled(2) when smicro_prog='1' else counter(22);
led(1 downto 0) <= pled(1 downto 0) when smicro_prog ='1' else mled(1 downto 0);
enable_out_buffers_l <= '0';
enable_in_buffers_l <= '0';
not_micro_prog <= not smicro_prog;
```

```
end Behavioral;
```

program_ram.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity program_ram is
    Port ( RST_N : in std_logic;
           CLK : in std_logic;
           ENABLE_N : in std_logic;
           DATA_VALID : in std_logic;
           DATA : inout std_logic_vector(7 downto 0);
           BUSY : out std_logic;
                     CONTROL: in std_logic;
                     READWRITE: in std_logic;
           RAM_ADDRESS : out std_logic_vector(18 downto 0);
           RAM_DATA : inout std_logic_vector(31 downto 0);
           RAM_OE : out std_logic;
           RAM_WE : out std_logic_vector(3 downto 0);
           LED : out std_logic_vector(2 downto 0));
end program_ram;

architecture Behavioral of program_ram is
type  states  is  (IDLE,WAIT_VALID,  WRITE_ADDR,  WRITE,  CHECK,  READ,  ESTABILIZA,
ESTABILIZA2 );
signal CS,NS: states;
signal ADDR: std_logic_vector(18 downto 0);
signal LDATA: std_logic_vector(7 downto 0);
signal nibble: integer range 0 to 3;
signal incrementa: std_logic;

begin
SYNC: process(RST_N, enable_n, CLK)
      begin
      if (RST_N = '0' or ENABLE_N = '1') then
            CS <= IDLE;
            nibble<=0;
            ADDR <= (others =>'0');
            LDATA <= (others =>'0');
      elsif(CLK'event and CLK ='1') then
            CS <= NS;
            if (CS = WRITE_ADDR) then
                  case nibble is
                        when 0 => ADDR(7 downto 0) <= DATA;
                        when 1 => ADDR(15 downto 8) <= DATA;
                        when 2 => ADDR(18 downto 16) <= DATA (2 downto 0);
                        when 3 =>
                  end case;
            end if;
            if (CS = WAIT_VALID and DATA_VALID ='1') then
                  LDATA <= DATA;
            end if;
            if (incrementa='1') then
                  if (nibble = 3) then
                        if (CS /= WRITE_ADDR) then
                              nibble <= 0;
                              ADDR <= ADDR +1;
                        end if;
                  else
```

```
                        nibble <= nibble + 1;
                  end if;
            end if;
      end if;
end process;

COMB: process(cs,enable_n, DATA_VALID, ADDR, nibble, RAM_DATA, CONTROL, READWRITE)
variable rdata: std_logic_vector(7 downto 0);
begin
      case nibble is
            when 0 => rdata := ram_data(7    downto 0);
            when 1 => rdata := ram_data(15 downto 8);
            when 2 => rdata := ram_data(23 downto 16);
            when 3 => rdata := ram_data(31 downto 24);
      end case;
      case CS is
            when IDLE =>
                              NS <= WAIT_VALID;
                              BUSY <= '1';
                              RAM_DATA <= (others =>'Z');
                              RAM_OE <= 'Z';
                              RAM_WE <= "ZZZZ";
                              DATA <= (others =>'Z');
                              incrementa <= '0';
                              LED(1 downto 0) <= "01";

            when WAIT_VALID =>
                              if (DATA_VALID='0') then
                                    NS <= WAIT_VALID;
                              elsif (CONTROL = '1') then
                                    NS <= WRITE_ADDR;
                              elsif (READWRITE = '1') then
                                    NS <= READ;
                              else
                                    NS <= WRITE;
                              end if;
                              BUSY <= '0';
                              RAM_DATA <= (others =>'Z');
                              RAM_OE <= '1';
                              RAM_WE <= "1111";
                              DATA <= (others =>'Z');
                              incrementa <= '0';
                              LED(1 downto 0) <= "00";
            when WRITE =>
                              if (DATA_VALID = '1') then
                                    NS <= WRITE;
                              else
                                    NS <= ESTABILIZA;
                              end if;
                              BUSY <= '1';
                              RAM_DATA <= DATA & DATA & DATA & DATA;
                              case nibble is
                                    when 0=> RAM_WE <= "1110";
                                    when 1=> RAM_WE <= "1101";
                                    when 2=> RAM_WE <= "1011";
                                    when 3=> RAM_WE <= "0111";
                              end case;
                              RAM_OE <= '1';
                              DATA <= (others =>'Z');
                              incrementa <= '0';
                              LED(1 downto 0) <= "11";
            when ESTABILIZA =>
                              NS <= ESTABILIZA2;
                              incrementa <= '0';
                              BUSY <= '1';
                              RAM_DATA <= DATA & DATA & DATA & DATA;
                              RAM_OE <= '1';
                              RAM_WE <= "1111";
```

```
                        DATA <= (others =>'Z');
                        LED(1 downto 0) <= "11";
            when ESTABILIZA2 =>
                        NS <= CHECK;
                        incrementa <= '0';
                        BUSY <= '1';
                        RAM_DATA <= (others =>'Z');
                        RAM_OE <= '0';
                        RAM_WE <= "1111";
                        DATA <= (others =>'Z');
                        LED(1 downto 0) <= "11";
            when CHECK =>
                        if (rdata = DATA) then
                              NS <= WAIT_VALID;
                              incrementa <= '1';
                        else
                              NS <= WRITE;
                              incrementa <= '0';
                        end if;
                        BUSY <= '1';
                        RAM_DATA <= (others =>'Z');
                        RAM_OE <= '0';
                        RAM_WE <= "1111";
                        DATA <= (others =>'Z');
                        LED(1 downto 0) <= "11";
            when WRITE_ADDR =>
                        if (DATA_VALID='0') then
                              NS <= WAIT_VALID;
                              incrementa <= '1';
                        else
                              NS <= WRITE_ADDR;
                              incrementa <= '0';
                        end if;
                        BUSY <= '0';
                        RAM_DATA <= (others =>'Z');
                        RAM_OE <= '1';
                        RAM_WE <= "1111";
                        DATA <= (others =>'Z');
                        LED(1 downto 0) <= "01";
            when READ =>
                        if (DATA_VALID='0') then
                              NS <= WAIT_VALID;
                              incrementa <= '1';
                        else
                              NS <= READ;
                              incrementa <= '0';
                        end if;
                        BUSY <= '0';
                        RAM_DATA <= (others =>'Z');
                        RAM_OE <= '0';
                        RAM_WE <= "1111";
                        DATA <= rdata;
                        LED(1 downto 0) <= "10";
      end case;
end process;

RAM_ADDRESS <= ADDR when CS /= IDLE else (others =>'Z');
LED(2) <= DATA_VALID;
end Behavioral;
```

11.4.5 Mapeador probabilistico multievento

mapper_function.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

      entity Mapper_function is
    Port ( CLK : in std_logic;
           RST_N : in std_logic;
           ENABLE_N : in std_logic;
                     BUSY : out std_logic;
           AER_IN_DATA : in std_logic_vector(15 downto 0);
           AER_IN_REQ_L : in std_logic;
           AER_IN_ACK_L : out std_logic;
           AER_OUT_DATA : out std_logic_vector(15 downto 0);
           AER_OUT_REQ_L : out std_logic;
           AER_OUT_ACK_L : in std_logic;
           RAM_ADDRESS : out std_logic_vector(18 downto 0);
           RAM_DATA : inout std_logic_vector(31 downto 0);
           RAM_OE : out std_logic;
           RAM_WE : out std_logic_vector(3 downto 0);
           LED : out std_logic_vector(2 downto 0)
                   );
end Mapper_function;

architecture Behavioral of Mapper_function is
type  states  is  (IDLE,  WAIT_REQ_L,  WAIT_REQ_H,  READ_RAM,  READ_RAM2,  SEND_EVENT,
SEND_EVENTrepe,SUBE_ACK );
signal CS,NS: states;
signal latched_input: std_logic_vector(15 downto 0);
signal last_event, no_event: std_logic;
signal event_counter: std_logic_vector(2 downto 0);
signal repeticiones: std_logic_vector (3 downto 0);
signal last_rep: std_logic;
signal lfsr: std_logic_vector(31 downto 0);


begin
-- generador de probabilidades, basado en lfsr

prob: process (clk,RST_N,lfsr)
variable i: natural;
begin
      if RST_N = '0' then
            lfsr <= x"80000000";
      elsif CLK'event and CLK='1' then
            for i in 31 downto 1 loop
       lfsr(i) <= lfsr(i-1);
      end loop;
            lfsr(0)<= lfsr(31) xor lfsr(21) xor lfsr(1) xor lfsr (0);
      end if;
end process;
```

```
SYNC: process(RST_N, enable_n, CLK)
begin
	if RST_N = '0' or ENABLE_N ='1' then
		CS <= IDLE;
	elsif(CLK'event and CLK ='1') then
		CS <= NS;

	end if;
end process;


COMB:		process(CS,		AER_IN_REQ_L,		AER_OUT_ACK_L,		RAM_DATA,
latched_input,ENABLE_N,last_rep)
begin
case CS is
	when IDLE =>
					if ENABLE_N = '1' then
						NS <= IDLE;
					else
						NS <= WAIT_REQ_L;
					end if;

					--NS <= WAIT_REQ_L;
					AER_IN_ACK_L <= '1';
					AER_OUT_REQ_L <= '1';
					AER_OUT_DATA <= (others =>'Z');
					RAM_ADDRESS <= (others =>'Z');
					RAM_OE <= 'Z';
					RAM_WE <= "ZZZZ";
					led <= "000";
					busy <='0';

	when WAIT_REQ_L =>

					if ENABLE_N = '1' then
						NS <= IDLE;
					elsif (AER_IN_REQ_L = '0') then
						NS <= WAIT_REQ_H;
					else
						NS <= WAIT_REQ_L;
					end if;
					AER_IN_ACK_L <= '1';
					AER_OUT_REQ_L <= '1';
					AER_OUT_DATA <= RAM_DATA(15 downto 0);
					RAM_ADDRESS <=  latched_input & event_counter;
					RAM_OE <= '0';
					RAM_WE <= "1111";
					led <= "001";
					busy <='1';

	when WAIT_REQ_H =>
					if	(AER_IN_REQ_L = '0') then
						NS <= WAIT_REQ_H;
					else
						NS <= READ_RAM2;
					end if;
					AER_IN_ACK_L <= '0';
					AER_OUT_REQ_L <= '1';
					AER_OUT_DATA <= RAM_DATA(15 downto 0);
					RAM_ADDRESS <= latched_input & event_counter;
					RAM_OE <= '0';
					RAM_WE <= "1111";
					led <= "011";
					busy <='1';


	when READ_RAM =>
```

```
                                    NS <= READ_RAM2;
                                    AER_IN_ACK_L <= '0';
                                    AER_OUT_REQ_L <= '1';
                                    AER_OUT_DATA <= RAM_DATA(15 downto 0);
                                    RAM_ADDRESS <= latched_input & event_counter;
                                    RAM_OE <= '0';
                                    RAM_WE <= "1111";
                                    led <= "100";
                                    busy <='1';

      when READ_RAM2 =>
                                    if (AER_OUT_ACK_L = '0') then
                                          NS <= READ_RAM2;
                                    elsif RAM_DATA (20 downto 17)="0000" then -- se
come el evento, no debe poner el req a la salida
                                          NS <= WAIT_REQ_L;
                                    elsif RAM_DATA(31 downto 24) < lfsr(7 downto 0)
then -- comprueba la probabilidad
                                          if last_event ='1' then
                                                NS <= SUBE_ACK;
                                          else
                                                NS <= READ_RAM;
                                          end if;
                                    else
                                          NS <= SEND_EVENT;
                                    end if;
                                    AER_IN_ACK_L <= '0';
                                    AER_OUT_REQ_L <= '1';
                                    AER_OUT_DATA <= RAM_DATA(15 downto 0);
                                    RAM_ADDRESS <= latched_input & event_counter;
                                    RAM_OE <= '0';
                                    RAM_WE <= "1111";
                                    led <= "100";
                                    busy <='1';

      when SEND_EVENT =>
                                    if (AER_OUT_ACK_L = '1') then
                                          NS <= SEND_EVENT;
                                    elsif last_rep= '0' then
                                          NS <= SEND_EVENTrepe;
                                    elsif (last_event = '0') then
                                          NS <= READ_RAM2;
                                    else
                                          --NS <= READ_RAM2;
                                          NS <= SUBE_ACK;
                                    end if;
                                    AER_IN_ACK_L <= '0';
                                    AER_OUT_REQ_L <= '0';
                                    AER_OUT_DATA <= RAM_DATA(15 downto 0);
                                    RAM_ADDRESS <= latched_input & event_counter;
                                    RAM_OE <= '0';
                                    RAM_WE <= "1111";
                                    led <= "101";
                                    busy <='1';
when SEND_EVENTrepe =>

                                    if (AER_OUT_ACK_L = '0') then
                                          NS <= SEND_EVENTrepe;
                                    else
                                          NS <= SEND_EVENT;
                                    end if;
                                    AER_IN_ACK_L <= '0';
                                    AER_OUT_REQ_L <= '1';
                                    AER_OUT_DATA <= RAM_DATA(15 downto 0);
                                    RAM_ADDRESS <= latched_input & event_counter;
                                    RAM_OE <= '0';
                                    RAM_WE <= "1111";
                                    led <= "101";
```

```
                                    busy <='1';

      when SUBE_ACK =>                                    --    Tambien    se    podria
comprobar ACK_OUT aqui
                                    NS <= WAIT_REQ_L;
                                    AER_IN_ACK_L <= '1';
                                    AER_OUT_REQ_L <= '1';
                                    AER_OUT_DATA <= RAM_DATA(15 downto 0);
                                    RAM_ADDRESS <= latched_input & event_counter;
                                    RAM_OE <= '0';
                                    RAM_WE <= "1111";
                                    led <= "111";
                                    busy <='1';

      end case;
end process;

SYCN: process( RST_N, CLK, CS, NS, AER_IN_DATA)
begin
if (RST_N = '0')  then
      event_counter <= (others =>'0');
            latched_input <= (others =>'0');
--          last_event <= '0';
--          no_event <= '0';

elsif(CLK'event and CLK='1') then
--    last_event <= RAM_DATA(16);
--    no_event <= RAM_DATA(17);

      if (CS = WAIT_REQ_L and NS = WAIT_REQ_H)  then
                  --latched_input   <=  AER_IN_DATA(15   downto   8)&"0"&AER_IN_DATA(7
downto 1);
                  latched_input <= AER_IN_DATA;
                  event_counter <= (others =>'0');
      elsif (CS = SEND_EVENT and NS = READ_RAM2) or (CS = READ_RAM2 and NS = READ_RAM)
then

                  event_counter <= event_counter + 1;

      end if;
end if;
end process;

repe: process (RST_N,clk, repeticiones, cs, ns)
begin
      if rst_n='0' then
            repeticiones<=(others =>'0');
      elsif clk='1' and clk'event then
            if cs = READ_RAM2 then
                  repeticiones <= RAM_DATA (20 downto 17);
            elsif cs= SEND_EVENT and NS=SEND_EVENTrepe    then
                  repeticiones <= repeticiones -1;
            else
                  repeticiones <= repeticiones;
            end if;
      end if;
end process;


last_rep <= '1' when repeticiones = "0001" else '0';

last_event <= RAM_DATA(16) ;
--no_event <= RAM_DATA(16);
RAM_DATA <= (others =>'Z');

end Behavioral;
```

11.4.6 Framegrabber interno

aer_framegrabber.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity aer_framegrabber is
    Port ( aer_in_data : in std_logic_vector(15 downto 0);
           aer_in_req_l : in std_logic;
           aer_in_ack_l : out std_logic;
           rst_l : in std_logic;
           clk : in std_logic;
           micro_data : out std_logic_vector(7 downto 0);
           micro_vdata : in std_logic;
           micro_prog : in std_logic;
           micro_control : in std_logic;
                     micro_rw: in std_logic;
           micro_busy : out std_logic;
                     enable_in_buffers_l: out std_logic;
                     LED: out std_logic_vector (2 downto 0)
                     );
end aer_framegrabber;

architecture Behavioral of aer_framegrabber is

 component spblockram
 generic (TAM: in integer:= 4096; IL: in integer:=12; WL: in integer:=8);
 port (clk : in std_logic;
       we  : in std_logic;
       a   : in std_logic_vector(11 downto 0);
       di  : in std_logic_vector(7 downto 0);
       do  : out std_logic_vector(7 downto 0));
 end component;

 signal counter_address: std_logic_vector(11 downto 0);
 signal counter_data_in: std_logic_vector(7 downto 0);
 signal counter_data_out: std_logic_vector(7 downto 0);
 signal counter_we: std_logic;

  type estados is (IDLE, WAIT_ACKH, READ_RAM, WRITE_RAM, WAIT_VALID, WAIT_NOVALID,
ERASE);
 signal CS, NS: estados;

 signal LATCHED_EVENT: std_logic_vector(11 downto 0);
 signal contador: std_logic_vector(11 downto 0);

 signal imicro_vdata,smicro_vdata: std_logic;
 signal imicro_rw, smicro_rw: std_logic;
 signal imicro_prog, smicro_prog: std_logic;
 signal imicro_control, smicro_control: std_logic;
 signal iaer_in_req_l, saer_in_req_l: std_logic;

begin

       contadores: spblockram generic map(4096,12,8)
       PORT MAP (
```

```
		clk => clk,
		we => counter_we,
		a => counter_address,
		di => counter_data_in,
		do => counter_data_out
		);

SYNC_AER: process(RST_L, CLK)
begin
if(RST_L = '0') then
	CS <= IDLE;
	LATCHED_EVENT <=(others =>'0');
	contador <= (others =>'0');
elsif(CLK'event and CLK ='1') then
	CS <= NS;
	if (CS = IDLE and saer_in_req_l = '0') then
		LATCHED_EVENT <= aer_in_data(13 downto 8) & aer_in_data(5 downto 0);
	end if;
	if (CS = IDLE) then
		contador <= (others =>'0');
	elsif(CS = ERASE) then
		contador <= contador + 1;
	end if;
end if;
end process;

COMB_AER:  process(counter_data_out,  aer_in_data,  saer_in_req_l,  CS,  smicro_prog,
smicro_vdata, smicro_control, LATCHED_EVENT, contador)
begin
case CS is

	when IDLE =>
			if (smicro_prog = '1') then
				NS <= WAIT_VALID;
			elsif (saer_in_req_l = '0') then
				NS <= WAIT_ACKH;
			else
				NS <= IDLE;
			end if;
			aer_in_ack_l <= '1';
			counter_address <= LATCHED_EVENT;
			counter_data_in <= (others =>'0');
			counter_we <= '0';

	when WAIT_ACKH =>
			if (saer_in_req_l = '1') then
				NS <= READ_RAM;
			else
				NS <= WAIT_ACKH;
			end if;
			aer_in_ack_l <= '0';
			counter_address <= LATCHED_EVENT;
			counter_data_in <= (others =>'0');
			counter_we <= '0';

	when READ_RAM =>
			NS <= WRITE_RAM;
			aer_in_ack_l <= '1';
			counter_address <= LATCHED_EVENT;
			counter_data_in <= (others =>'0');
			counter_we <= '0';

	when WRITE_RAM =>
			NS <= IDLE;
			aer_in_ack_l <= '1';
			counter_address <= LATCHED_EVENT;
			if (counter_data_out /= x"ff") then
				counter_data_in <= counter_data_out + 1;
```

```
                        counter_we <= '1';
                else
                        counter_data_in <= counter_data_out;
                        counter_we <= '0';
                end if;

        when WAIT_VALID =>
                if (smicro_prog = '0') then
                        NS <= IDLE;
                elsif (smicro_vdata = '0' or smicro_control = '1') then
                        NS <= WAIT_VALID;
                else
                        NS <= WAIT_NOVALID;
                end if;
                aer_in_ack_l <= saer_in_req_l;
                counter_address <= contador;
                counter_data_in <= (others =>'0');
                counter_we <= '0';

        when WAIT_NOVALID =>
                if (smicro_vdata = '1') then
                        NS <= WAIT_NOVALID;
                else
                        NS <= ERASE;
                end if;
                aer_in_ack_l <= saer_in_req_l;
                counter_address <= contador;
                counter_data_in <= (others =>'0');
                counter_we <= '0';

        when ERASE =>
                NS <= WAIT_VALID;
                aer_in_ack_l <= saer_in_req_l;
                counter_address <= contador;
                counter_data_in <= (others =>'0');
                counter_we <= '1';
        end case;
end process;


micro_busy <= '0';
micro_data <= counter_data_out when (smicro_prog = '1' and smicro_control = '0' and
smicro_rw = '1') else (others =>'Z');


LED (2 downto 0) <= "101";
enable_in_buffers_l <= '0';

syncronizer: process(RST_L, CLK)
begin
if (RST_L = '0') then
        imicro_vdata <= '0';
        smicro_vdata <= '0';
        imicro_rw <= '0';
        smicro_rw <= '0';
        imicro_prog <= '0';
        smicro_prog <= '0';
        imicro_control <= '0';
        smicro_control <= '0';
        iaer_in_req_l <= '0';
        saer_in_req_l <= '0';
elsif(CLK'event and CLK = '1') then
        imicro_vdata <= micro_vdata;
        smicro_vdata <= imicro_vdata;
        imicro_rw <= micro_rw;
        smicro_rw <= imicro_rw;
        imicro_control <= micro_control;
        smicro_control <= imicro_control;
```

```
      imicro_prog <= micro_prog;
      smicro_prog <= imicro_prog;
      iaer_in_req_l <= aer_in_req_l;
      saer_in_req_l <= iaer_in_req_l;
end if;
end process;
end Behavioral;
```

single_ram.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;


-- Only XST supports RAM inference
-- Infers Single Port Block Ram

 entity spblockram is
 generic (TAM: in integer:= 2048; IL: in integer:=11; WL: in integer:=8);
 port (clk : in std_logic;
      we  : in std_logic;
      a   : in std_logic_vector(IL-1 downto 0);
      di  : in std_logic_vector(WL-1 downto 0);
      do  : out std_logic_vector(WL-1 downto 0));
 end spblockram;

 architecture syn of spblockram is

 type ram_type is array (TAM-1 downto 0) of std_logic_vector (WL-1 downto 0);
 signal RAM : ram_type;
 signal read_a : std_logic_vector(IL-1 downto 0);

 begin
 process (clk)
 begin
      if (clk'event and clk = '1') then
            if (we = '1') then
                  RAM(conv_integer(a)) <= di;
            end if;
            read_a <= a;
      end if;
 end process;

   --do   <=   (others   =>'0')   when   (RAM(conv_integer(read_a))="UUUUUUUU")   else
RAM(conv_integer(read_a)) ;
 do <= RAM(conv_integer(read_a));
 end syn;
```

11.4.7 Framegrabber con signo

aer_framegrabber.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity aer_framegrabber is
    Port ( aer_in_data : in std_logic_vector(15 downto 0);
           aer_in_req_l : in std_logic;
           aer_in_ack_l : out std_logic;
           rst_l : in std_logic;
           clk : in std_logic;
           micro_data : out std_logic_vector(7 downto 0);
           micro_vdata : in std_logic;
           micro_prog : in std_logic;
           micro_control : in std_logic;
                     micro_rw: in std_logic;
           micro_busy : out std_logic;
                     enable_in_buffers_l: out std_logic;
                     LED: out std_logic_vector (2 downto 0)
                     );
end aer_framegrabber;

architecture Behavioral of aer_framegrabber is

 component spblockram
 generic (TAM: in integer:= 4096; IL: in integer:=12; WL: in integer:=8);
 port (clk : in std_logic;
       we  : in std_logic;
       a   : in std_logic_vector(11 downto 0);
       di  : in std_logic_vector(7 downto 0);
       do  : out std_logic_vector(7 downto 0));
 end component;

 signal counter_address: std_logic_vector(11 downto 0);
 signal counter_data_in: std_logic_vector(7 downto 0);
 signal counter_data_out: std_logic_vector(7 downto 0);
 signal counter_we: std_logic;

  type estados is (IDLE, WAIT_ACKH, READ_RAM, WRITE_RAM, WAIT_VALID, WAIT_NOVALID,
ERASE);
 signal CS, NS: estados;

 signal LATCHED_EVENT: std_logic_vector(11 downto 0);
 signal LATCHED_SIGNO: std_logic;

 signal contador: std_logic_vector(11 downto 0);

 signal imicro_vdata,smicro_vdata: std_logic;
 signal imicro_rw, smicro_rw: std_logic;
 signal imicro_prog, smicro_prog: std_logic;
 signal imicro_control, smicro_control: std_logic;
 signal iaer_in_req_l, saer_in_req_l: std_logic;

begin

       contadores: spblockram generic map(4096,12,8)
       PORT MAP (
```

```
              clk => clk,
              we => counter_we,
              a => counter_address,
              di => counter_data_in,
              do => counter_data_out
              );


SYNC_AER: process(RST_L, CLK)
begin
if(RST_L = '0') then
       CS <= IDLE;
       LATCHED_EVENT <=(others =>'0');
       LATCHED_SIGNO <= '0';
       contador <= (others =>'0');
elsif(CLK'event and CLK ='1') then
       CS <= NS;
       if (CS = IDLE and saer_in_req_l = '0') then
              LATCHED_EVENT <= aer_in_data(13 downto 8) & aer_in_data(5 downto 0);
              LATCHED_SIGNO <= aer_in_data(15);             -- Bit de signo
       end if;
       if (CS = IDLE) then
              contador <= (others =>'0');
       elsif(CS = ERASE) then
              contador <= contador + 1;
       end if;
end if;
end process;

COMB_AER:  process(counter_data_out,  aer_in_data,  saer_in_req_l,  CS,  smicro_prog,
smicro_vdata, smicro_control, LATCHED_EVENT, LATCHED_SIGNO, contador)
begin
case CS is

       when IDLE =>
                     if (smicro_prog = '1') then
                            NS <= WAIT_VALID;
                     elsif (saer_in_req_l = '0') then
                            NS <= WAIT_ACKH;
                     else
                            NS <= IDLE;
                     end if;
                     aer_in_ack_l <= '1';
                     counter_address <= LATCHED_EVENT;
                     counter_data_in <= (others =>'0');
                     counter_we <= '0';

       when WAIT_ACKH =>
                     if (saer_in_req_l = '1') then
                            NS <= READ_RAM;
                     else
                            NS <= WAIT_ACKH;
                     end if;
                     aer_in_ack_l <= '0';
                     counter_address <= LATCHED_EVENT;
                     counter_data_in <= (others =>'0');
                     counter_we <= '0';

       when READ_RAM =>
                     NS <= WRITE_RAM;
                     aer_in_ack_l <= '1';
                     counter_address <= LATCHED_EVENT;
                     counter_data_in <= (others =>'0');
                     counter_we <= '0';

       when WRITE_RAM =>
                     NS <= IDLE;
                     aer_in_ack_l <= '1';
```

```
                        counter_address <= LATCHED_EVENT;
                        if (LATCHED_SIGNO = '0')  then -- si es positivo
                              if (counter_data_out /= x"7f") then
                                    counter_data_in <= counter_data_out + 1;
                                    counter_we <= '1';
                              else
                                    counter_data_in <= counter_data_out;
                                    counter_we <= '0';
                              end if;
                        else
                              if(counter_data_out /= x"80") then
                                    counter_data_in <= counter_data_out - 1;
                                    counter_we <= '1';
                              else
                                    counter_data_in <= counter_data_out;
                                    counter_we <= '0';
                              end if;
                        end if;



      when WAIT_VALID =>
                        if (smicro_prog = '0') then
                              NS <= IDLE;
                        elsif (smicro_vdata = '0' or smicro_control = '1') then
                              NS <= WAIT_VALID;
                        else
                              NS <= WAIT_NOVALID;
                        end if;
                        aer_in_ack_l <= saer_in_req_l;
                        counter_address <= contador;
                        counter_data_in <= (others =>'0');
                        counter_we <= '0';

      when WAIT_NOVALID =>
                        if (smicro_vdata = '1') then
                              NS <= WAIT_NOVALID;
                        else
                              NS <= ERASE;
                        end if;
                        aer_in_ack_l <= saer_in_req_l;
                        counter_address <= contador;
                        counter_data_in <= (others =>'0');
                        counter_we <= '0';

      when ERASE =>
                        NS <= WAIT_VALID;
                        aer_in_ack_l <= saer_in_req_l;
                        counter_address <= contador;
                        counter_data_in <= (others =>'0');
                        counter_we <= '1';
      end case;
end process;


micro_busy <= '0';
micro_data <= counter_data_out when (smicro_prog = '1' and smicro_control = '0' and
smicro_rw = '1') else (others =>'Z');


LED (2 downto 0) <= "101";
enable_in_buffers_l <= '0';

syncronizer: process(RST_L, CLK)
begin
if (RST_L = '0') then
      imicro_vdata <= '0';
      smicro_vdata <= '0';
```

```
        imicro_rw <= '0';
        smicro_rw <= '0';
        imicro_prog <= '0';
        smicro_prog <= '0';
        imicro_control <= '0';
        smicro_control <= '0';
        iaer_in_req_l <= '0';
        saer_in_req_l <= '0';
elsif(CLK'event and CLK = '1') then
        imicro_vdata <= micro_vdata;
        smicro_vdata <= imicro_vdata;
        imicro_rw <= micro_rw;
        smicro_rw <= imicro_rw;
        imicro_control <= micro_control;
        smicro_control <= imicro_control;
        imicro_prog <= micro_prog;
        smicro_prog <= imicro_prog;
        iaer_in_req_l <= aer_in_req_l;
        saer_in_req_l <= iaer_in_req_l;
end if;
end process;



end Behavioral;
```

11.4.8 Framegrabber externo

aer_framegrabber.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity aer_framegrabber is
    Port ( aer_in_data : in std_logic_vector(15 downto 0);
           aer_in_req_l : in std_logic;
           aer_in_ack_l : out std_logic;
           rst_l : in std_logic;
           clk50 : in std_logic;
                     RAM_ADDRESS : out std_logic_vector(18 downto 0);
           RAM_DATA : inout std_logic_vector(31 downto 0);
           RAM_OE : out std_logic;
           RAM_WE : out std_logic_vector(3 downto 0);
           micro_data : out std_logic_vector(7 downto 0);
           micro_vdata : in std_logic;
           micro_prog : in std_logic;
           micro_control : in std_logic;
                     micro_rw: in std_logic;
           micro_busy : out std_logic;
                     enable_in_buffers_l: out std_logic;
                     LED: out std_logic_vector (2 downto 0)
                     );
end aer_framegrabber;

architecture Behavioral of aer_framegrabber is


--- duplicar reloj ---
component IBUFG
port (
   O : out std_logic;
   I : in std_logic
 );
end component;


component CLKDLL
 port (
   CLK0 : out std_logic;
   CLK90 : out std_logic;
   CLK180 : out std_logic;
   CLK270 : out std_logic;
   CLK2X : out std_logic;
   CLKDV : out std_logic;
   LOCKED : out std_logic;
   CLKIN : in std_logic;
   CLKFB : in std_logic;
   RST : in std_logic
 );
end component;

component BUFG
 port (
   O : out std_logic;
```

```
   I : in std_logic
 );
end component;
signal CLKIN_w, RESET_w, CLK2X_dll, CLK2X_g,CLK100 : std_logic;
signal LOCKED2X, LOCKED2X_delay, rst,CLK: std_logic;

---- fin duplicar reloj ----

--
  type estados is (IDLE, WAIT_ACKH, READ_RAM, WRITE_RAM, WAIT_VALID, WAIT_NOVALID,
ERASE);
 signal CS, NS: estados;

 signal LATCHED_EVENT: std_logic_vector(13 downto 0);
 signal contador: std_logic_vector(13 downto 0);

 signal imicro_vdata,smicro_vdata: std_logic;
 signal imicro_rw, smicro_rw: std_logic;
 signal imicro_prog, smicro_prog: std_logic;
 signal imicro_control, smicro_control: std_logic;
 signal iaer_in_req_l, saer_in_req_l: std_logic;
 signal latched_ram_data:std_logic_vector(31 downto 0);

begin
---- duplicar reloj-----
rst <= not rst_l;
clkpad : IBUFG  port map (I=>CLK50, O=>CLKIN_w);

dll2x  : CLKDLL port map (CLKIN=>CLKIN_w,   CLKFB=>CLK2X_g, RST=>RST,
                          CLK0=>open,   CLK90=>open, CLK180=>open, CLK270=>open,
                           CLK2X=>CLK2X_dll, CLKDV=>open, LOCKED=>LOCKED2X);

clk2xg : BUFG   port map (I=>CLK2X_dll,   O=>CLK2X_g);
clk50g : BUFG   port map (I=>CLKIN_w,   O=>open); --CLK
--CLK <= CLK2X_g;

CLK100 <= CLK2X_g;
CLK <= CLKIN_w;
------- fin duplicar reloj --------------

SYNC_AER: process(RST_L, CLK)
begin
if(RST_L = '0') then
       CS <= IDLE;
       LATCHED_EVENT <=(others =>'0');
       contador <= (others =>'0');
elsif(CLK'event and CLK ='1') then
       CS <= NS;
       if (CS = IDLE and saer_in_req_l = '0') then
              LATCHED_EVENT <= aer_in_data(14 downto 8) & aer_in_data(6 downto 0);
       end if;
       if (CS = IDLE) then
              contador <= (others =>'0');
       elsif(CS = ERASE) then
              contador <= contador + 1;
       end if;
       if (NS=READ_RAM) then
              latched_ram_data <= ram_data;
       end if;
end if;
end process;

COMB_AER:  process(latched_ram_data,  aer_in_data,  saer_in_req_l,  CS,  smicro_prog,
smicro_vdata, smicro_control, LATCHED_EVENT, contador)
begin
case CS is

       when IDLE =>
```

```
            if (smicro_prog = '1') then
                  NS <= WAIT_VALID;
            elsif (saer_in_req_l = '0') then
                  NS <= WAIT_ACKH;
            else
                  NS <= IDLE;
            end if;
            aer_in_ack_l <= '1';
            ram_address <= "00000" & LATCHED_EVENT;
            ram_data <= (others =>'0');
            ram_we <="1111";
            ram_oe <='1';

when WAIT_ACKH =>
            if (saer_in_req_l = '1') then
                  NS <= READ_RAM;
            else
                  NS <= WAIT_ACKH;
            end if;
            aer_in_ack_l <= '0';
            ram_address <= "00000" & LATCHED_EVENT;
            ram_data <= (others =>'Z');
            ram_we <="1111";
            ram_oe <='0';

when READ_RAM =>
            NS <= WRITE_RAM;
            aer_in_ack_l <= '1';
            ram_address <= "00000" & LATCHED_EVENT;
            ram_data <= (others =>'Z');
            ram_we <="1111";
            ram_oe <='0';

when WRITE_RAM =>
            NS <= IDLE;
            aer_in_ack_l <= '1';
            ram_address <= "00000" & LATCHED_EVENT;
            ram_oe <= '1';
            if (latched_ram_data(7 downto 0) /= x"ff") then
                  ram_data <= latched_ram_data + 1;
                  ram_we <= "0000";
            else
                  ram_data <= latched_ram_data;
                  ram_we <= "1111";
            end if;

when WAIT_VALID =>
            if (smicro_prog = '0') then
                  NS <= IDLE;
            elsif (smicro_vdata = '0' or smicro_control = '1') then
                  NS <= WAIT_VALID;
            else
                  NS <= WAIT_NOVALID;
            end if;
            aer_in_ack_l <= saer_in_req_l;
            ram_address <= "00000" & contador;
            ram_data <= (others =>'Z');
            ram_we <="1111";
            ram_oe <='0';

when WAIT_NOVALID =>
            if (smicro_vdata = '1') then
                  NS <= WAIT_NOVALID;
            else
                  NS <= ERASE;
            end if;
            aer_in_ack_l <= saer_in_req_l;
            ram_address <= "00000" & contador;
```

```
                    ram_data <= (others =>'Z');
                    ram_we <="1111";
                    ram_oe <='0';


      when ERASE =>
                    NS <= WAIT_VALID;
                    aer_in_ack_l <= saer_in_req_l;
                    ram_address <= "00000" & contador;
                    ram_data <= (others =>'0');
                    ram_we <="0000";
                    ram_oe <='1';

      end case;
end process;

micro_busy <= '0';
micro_data <= ram_data(7 downto 0) when (smicro_prog = '1' and smicro_control = '0'
and smicro_rw = '1') else (others =>'Z');
LED (2 downto 0) <= "101";
enable_in_buffers_l <= '0';

syncronizer: process(RST_L, CLK100)
begin
if (RST_L = '0') then
      imicro_vdata <= '0';
      smicro_vdata <= '0';
      imicro_rw <= '0';
      smicro_rw <= '0';
      imicro_prog <= '0';
      smicro_prog <= '0';
      imicro_control <= '0';
      smicro_control <= '0';
      iaer_in_req_l <= '0';
      saer_in_req_l <= '0';
elsif(CLK100'event and CLK100 = '1') then
      imicro_vdata <= micro_vdata;
      smicro_vdata <= imicro_vdata;
      imicro_rw <= micro_rw;
      smicro_rw <= imicro_rw;
      imicro_control <= micro_control;
      smicro_control <= imicro_control;
      imicro_prog <= micro_prog;
      smicro_prog <= imicro_prog;
      iaer_in_req_l <= aer_in_req_l;
      saer_in_req_l <= iaer_in_req_l;
end if;
end process;



end Behavioral;
```

11.4.9 Framegrabber pseudoinstantáneo (externo)

aer_framegrabber.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity aer_framegrabber is
    Port ( aer_in_data : in std_logic_vector(15 downto 0);
           aer_in_req_1 : in std_logic;
           aer_in_ack_1 : out std_logic;
           rst_1 : in std_logic;
           clk : in std_logic;
                     RAM_ADDRESS : out std_logic_vector(18 downto 0);
           RAM_DATA : inout std_logic_vector(31 downto 0);
           RAM_OE : out std_logic;
           RAM_WE : out std_logic_vector(3 downto 0);
           micro_data : out std_logic_vector(7 downto 0);
           micro_vdata : in std_logic;
           micro_prog : in std_logic;
           micro_control : in std_logic;
                     micro_rw: in std_logic;
           micro_busy : out std_logic;
                     enable_in_buffers_1: out std_logic;
                     LED: out std_logic_vector (2 downto 0)
                     );
end aer_framegrabber;

architecture Behavioral of aer_framegrabber is

 component spblockram
 generic (TAM: in integer:= 4096; IL: in integer:=12; WL: in integer:=7);
 port (clk : in std_logic;
       we  : in std_logic;
       a   : in std_logic_vector(11 downto 0);
       di  : in std_logic_vector(6 downto 0);
       do  : out std_logic_vector(6 downto 0));
 end component;

 signal counter_address: std_logic_vector(11 downto 0);
 signal counter_data_in: std_logic_vector(6 downto 0);
 signal counter_data_out: std_logic_vector(6 downto 0);
 signal counter_we: std_logic;

 signal counter_address2: std_logic_vector(11 downto 0);
 signal counter_data_in2: std_logic_vector(6 downto 0);
 signal counter_data_out2: std_logic_vector(6 downto 0);
 signal counter_we2: std_logic;


  type estados is (IDLE, WAIT_ACKH, READ_RAM, WRITE_RAM, WAIT_VALID, WAIT_NOVALID,
ERASE);
 signal CS, NS: estados;

 signal LATCHED_EVENT: std_logic_vector(11 downto 0);
```

```
 signal contador: std_logic_vector(11 downto 0);

 signal imicro_vdata,smicro_vdata: std_logic;
 signal imicro_rw, smicro_rw: std_logic;
 signal imicro_prog, smicro_prog: std_logic;
 signal imicro_control, smicro_control: std_logic;
 signal iaer_in_req_l, saer_in_req_l: std_logic;

 signal timer: std_logic_vector(31 downto 0);
 signal dato_salida: std_logic_vector(7 downto 0);

begin

       contadores: spblockram generic map(4096,12,7)
       PORT MAP (
              clk => clk,
              we => counter_we,
              a => counter_address,
              di => counter_data_in,
              do => counter_data_out
              );

       contadores2: spblockram generic map(4096,12,7)
       PORT MAP (
              clk => clk,
              we => counter_we2,
              a => counter_address2,
              di => counter_data_in2,
              do => counter_data_out2
              );

SYNC_AER: process(RST_L, CLK)
begin
if(RST_L = '0') then
       CS <= IDLE;
       LATCHED_EVENT <=(others =>'0');
       contador <= (others =>'0');
       timer <= (others =>'0');
elsif(CLK'event and CLK ='1') then
       CS <= NS;
       timer <= timer + 1;
       if (CS = IDLE and saer_in_req_l = '0') then
              LATCHED_EVENT <= aer_in_data(13 downto 8) & aer_in_data(5 downto 0);
       end if;
       if (CS = IDLE) then
              contador <= (others =>'0');
       elsif(CS = ERASE) then
              contador <= contador + 1;
       end if;
end if;
end process;

COMB_AER:  process(counter_data_out,  aer_in_data,  saer_in_req_l,  CS,  smicro_prog,
smicro_vdata, smicro_control, LATCHED_EVENT, contador)
begin
case CS is

       when IDLE =>
                     if (smicro_prog = '1') then
                            NS <= WAIT_VALID;
                     elsif (saer_in_req_l = '0') then
                            NS <= WAIT_ACKH;
                     else
                            NS <= IDLE;
                     end if;
                     aer_in_ack_l <= '1';
                     counter_address <= LATCHED_EVENT;
                     counter_data_in <= (others =>'0');
```

```
                counter_we <= '0';
                counter_address2 <= LATCHED_EVENT;
                counter_data_in2 <= (others =>'0');
                counter_we2 <= '0';

        when WAIT_ACKH =>
                if (saer_in_req_l = '1') then
                        NS <= READ_RAM;
                else
                        NS <= WAIT_ACKH;
                end if;
                aer_in_ack_l <= '0';
                counter_address <= LATCHED_EVENT;
                counter_data_in <= (others =>'0');
                counter_we <= '0';
                counter_address2 <= LATCHED_EVENT;
                counter_data_in2 <= (others =>'0');
                counter_we2 <= '0';

        when READ_RAM =>
                NS <= WRITE_RAM;
                aer_in_ack_l <= '1';
                counter_address <= LATCHED_EVENT;
                counter_data_in <= (others =>'0');
                counter_we <= '0';
                counter_address2 <= LATCHED_EVENT;
                counter_data_in2 <= (others =>'0');
                counter_we2 <= '0';

        when WRITE_RAM =>
                NS <= IDLE;
                aer_in_ack_l <= '1';
                counter_address <= LATCHED_EVENT;
                counter_address2 <= LATCHED_EVENT;

                counter_data_in <= timer(14+6 downto 14);
                counter_data_in2 <= counter_data_out;
                counter_we <= '1';
                counter_we2 <= '1';

--              if (counter_data_out /= x"7f") then
--                      counter_data_in <= counter_data_out2 + 1;
--                      counter_data_in2 <= counter_data_out + 1;
--                      counter_we <= '1';
--                      counter_we2 <= '1';
--              else
--                      counter_data_in <= counter_data_out;
--                      counter_data_in2 <= counter_data_out2;
--                      counter_we <= '0';
--                      counter_we2 <= '0';
--              end if;

        when WAIT_VALID =>
                if (smicro_prog = '0') then
                        NS <= IDLE;
                elsif (smicro_vdata = '0' or smicro_control = '1') then
                        NS <= WAIT_VALID;
                else
                        NS <= WAIT_NOVALID;
                end if;
                aer_in_ack_l <= saer_in_req_l;
                counter_address <= contador;
                counter_data_in <= (others =>'0');
                counter_we <= '0';
                counter_address2 <= contador;
                counter_data_in2 <= (others =>'0');
                counter_we2 <= '0';
```

```
	when WAIT_NOVALID =>
			if (smicro_vdata = '1') then
				NS <= WAIT_NOVALID;
			else
				NS <= ERASE;
			end if;
			aer_in_ack_l <= saer_in_req_l;
			counter_address <= contador;
			counter_data_in <= (others =>'0');
			counter_we <= '0';
			counter_address2 <= contador;
			counter_data_in2 <= (others =>'0');
			counter_we2 <= '0';

	when ERASE =>
			NS <= WAIT_VALID;
			aer_in_ack_l <= saer_in_req_l;
			counter_address <= contador;
			counter_data_in <= (others =>'0');
			counter_we <= '1';
			counter_address2 <= contador;
			counter_data_in2 <= (others =>'0');
			counter_we2 <= '1';
	end case;
end process;

deltatime: process( counter_data_out, counter_data_out2)
begin
if (counter_data_out2 > counter_data_out) then
	dato_salida <= ("1" & counter_data_out) - ("0" & counter_data_out2);
else
	dato_salida <= ("0" & counter_data_out) - ("0" & counter_data_out2);
end if;
end process;

micro_busy <= '0';
micro_data <= dato_salida when (smicro_prog = '1' and smicro_control = '0' and
smicro_rw = '1') else (others =>'Z');


LED (2 downto 0) <= "101";
enable_in_buffers_l <= '0';

syncronizer: process(RST_L, CLK)
begin
--if (RST_L = '0') then
--	imicro_vdata <= '0';
--	smicro_vdata <= '0';
--	imicro_rw <= '0';
--	smicro_rw <= '0';
--	imicro_prog <= '0';
--	smicro_prog <= '0';
--	imicro_control <= '0';
--	smicro_control <= '0';
--	iaer_in_req_l <= '0';
--	saer_in_req_l <= '0';
if(CLK'event and CLK = '1') then
	imicro_vdata <= micro_vdata;
	smicro_vdata <= imicro_vdata;
	imicro_rw <= micro_rw;
	smicro_rw <= imicro_rw;
	imicro_control <= micro_control;
	smicro_control <= imicro_control;
	imicro_prog <= micro_prog;
	smicro_prog <= imicro_prog;
	iaer_in_req_l <= aer_in_req_l;
	saer_in_req_l <= iaer_in_req_l;
end if;
```

```
end process;
end Behavioral;
```

single_ram.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;


-- Only XST supports RAM inference
-- Infers Single Port Block Ram

 entity spblockram is
 generic (TAM: in integer:= 2048; IL: in integer:=11; WL: in integer:=8);
 port (clk : in std_logic;
       we  : in std_logic;
       a   : in std_logic_vector(IL-1 downto 0);
       di  : in std_logic_vector(WL-1 downto 0);
       do  : out std_logic_vector(WL-1 downto 0));
 end spblockram;

 architecture syn of spblockram is

 type ram_type is array (TAM-1 downto 0) of std_logic_vector (WL-1 downto 0);
 signal RAM : ram_type;
 signal read_a : std_logic_vector(IL-1 downto 0);

 begin
 process (clk)
 begin
       if (clk'event and clk = '1') then
             if (we = '1') then
                   RAM(conv_integer(a)) <= di;
             end if;
             read_a <= a;
       end if;
 end process;

   --do   <=   (others   =>'0')   when   (RAM(conv_integer(read_a))="UUUUUUUU")   else
RAM(conv_integer(read_a)) ;
 do <= RAM(conv_integer(read_a));
 end syn;
```