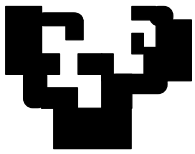


UNIVERSIDAD DEL PAIS VASCO
EUSKAL HERRIKO UNIBERTSITATEA

ESCUELA SUPERIOR DE INGENIEROS DE BILBAO
BILBOKO INGENIARIEN GOI ESKOLA

DEPARTAMENTO DE ELECTRONICA Y TELECOMUNICACIONES
ELEKTRONIKA ETA TELEKOMUNIKAZIOAK SAILA

TESIS DOCTORAL

REALIZACION DE MODELOS DE PROCESAMIENTO MEDIANTE VHDL PARA LA DETERMINACION DEL FLUJO OPTICO A PARTIR DE SECUENCIAS DE IMAGENES

Autor: Aitzol Zuloaga Izaguirre
Director: José Luis Martín González

Bilbao, Octubre de 2001

Biotz biotzez,

nere amari eta nere anaiei.

De corazón,

a mi madre y mis hermanos

Indice

Capítulo 1. Introducción 1

1.1. Marco de realización y justificación de la Tesis 2

1.2. Objetivos de la Tesis 3

1.3. Estructura de la Tesis 4

Capítulo 2. Determinación del flujo óptico 6

2.1. Introducción al tema de la Tesis 7

2.2. Historia del procesamiento de imágenes 9

2.2.1. La búsqueda de un sensor de imágenes: la cámara de vídeo. 9

2.2.2. La Transmisión de imágenes 13

2.2.3. La era digital y el procesamiento de imágenes 17

2.3. Procesamiento de imágenes 18

2.3.1. Secuencias de imágenes 21

2.3.2. Arquitecturas para el procesamiento de imágenes 22

2.4. El flujo óptico 26

2.5. Aplicaciones de la determinación del flujo óptico 28

2.5.1. Recuperación de la estructura tridimensional 29

2.5.2. Compresión de imágenes MPEG 30

2.5.3. Reconstrucción de imágenes 30

2.5.4. Determinación de dinámica de fluidos 31

2.6. Métodos para la determinación del flujo óptico 33

2.6.1. Método de la segmentación 34

2.6.2. Método de los gradientes 35

2.7. Determinación del flujo óptico por gradientes 36

2.7.1. Desarrollo teórico. 36

2.7.2. El problema de apertura 37

2.7.3. Restricciones de suavidad en el flujo óptico. 38

2.7.4. Estimando el Laplaciano 40

2.7.5. Ecuaciones iterativas de flujo óptico 41

Capítulo 3. Materiales y métodos 43

3.1. Señalización entre módulos segmentados 44

3.2. Herramientas diseñadas para esta Tesis 46

3.3. Banco de prueba en VHDL 46

3.3.1. El formato *hex* 48

3.3.2. Convirtiendo y visualizando ficheros de imagen *hex* 50

3.3.3. Módulos VHDL del banco de pruebas 51

3.4. Analizador de algoritmos 51

Capítulo 4. Arquitectura para la determinación del flujo óptico 55

4.1. Realización en bloques de las ecuaciones de flujo óptico 56

4.1.1. Definición de la arquitectura 56

4.1.2. Entradas y salidas del sistema 59

4.1.3. Descomposición en partes de las ecuaciones de flujo óptico 62

4.2. Diagrama a bloques general 68

4.2.1. Bloques de retardadores para el cálculo de gradientes 70

4.2.2. Bloque de cálculo de gradiente 72

4.2.3. Bloque de cálculo de término D 74

4.2.4. Bloque de cálculo de términos P 76

4.2.5. Bloques de cálculo de términos Rx y Ry 77

4.2.6. Bloques estimadores de velocidad 79

4.2.7. Bloques conversores 80

4.2.8. Retardadores para los coeficientes de promediado 80

4.2.9. Bloque de cálculo de los coeficientes de promediado 82

4.2.10. Bloque de control de FIFOs 84

4.2.11. Bloque retardador para compensación 85

4.2.12. Bloque de sincronización 86

4.2.13. FPGA para estimación del flujo óptico 88

4.2.14. Sistema completo y banco de pruebas 89

Capítulo 5. Simulaciones y resultados 91

5.1. Organización de las pruebas del sistema 92

5.2. Pruebas del algoritmo 92

5.3. Pruebas de los bloques 93

5.3.1. Prueba de bloque de cálculo de gradiente 94

5.3.2. Prueba de bloque de cálculo de término D 95

5.3.3. Prueba de bloque de cálculo de términos P 97

5.3.4. Prueba de bloque de cálculo de términos R 98

5.3.5. Prueba del bloque estimador de velocidad 99

5.3.6. Prueba del bloque conversor 100

5.3.7. Prueba del bloque de cálculo de los coeficientes de promediado 101

5.3.8. Prueba del bloque de control de FIFOs 102

5.3.9. Prueba del bloque de sincronización 103

5.4. Pruebas generales del sistema 105

5.4.1. Prueba con secuencia 1 107

5.4.2. Prueba con secuencia 2 109

5.4.3. Prueba con secuencia 3 111

5.4.4. Prueba con secuencia 4 113

5.4.5. Prueba con secuencia 5 115

5.5. Síntesis de la FPGA 117

5.6. Análisis de resultados 118

Capítulo 6. Conclusiones y trabajo futuro 121

6.1. Conclusiones 122

6.2. Resumen de las aportaciones originales 122

6.3. Trabajo futuro 123

6.4. Publicaciones generadas a partir de esta Tesis 124

Bibliografía 127

Bibliografía 128

Apéndices 135

Apéndice A. Especificaciones técnicas de memorias FIFO 136

Apéndice B. Descripciones VHDL de los módulos 140

B1. CDTERM: Calculador de término D 140

B2. CONVER: Conversor 141

B3. CPROMED: Calculador de coeficientes de promediado 141

B4. CPTERM: Calculador de términos P 142

B5. CTRFIFO: Control de FIFOs 143

B6. DELAYCP1: Retardador para coeficientes de promediado 144

B7. DELAYCP2: Retardador para coeficientes de promediado (Interno) 144

B8. DELAYCP3: Retardador para coeficientes de promediado (Externo) 145

B9. DELAYGR1: Retardador primario para cálculo de gradientes 146

B10. DELAYGR2: Retardador secundario para cálculo de gradientes 147

B11. DELAYSI: Retardador para sincronismo 148

B12. DELAYVP: Retardador para compensación 149

B13. DIVIP03: Divisor por tres para coeficientes de promediado 149

B14. DIVISOR: Divisor para el cálculo de términos R 152

B15. FPGABLK: Bloque principal de la FPGA 158

B16. GRADEXT: Extractor de gradientes 162

B17. MEMFIFO: Descripción VHDL comportamental de memorias FIFO 163

B18. MULTSGN1: Multiplicador con signo 164

B19. MULTSGN2: Multiplicador con signo 165

B20. MULTSGN3: Multiplicador con signo 165

B21. SINCRONI: Generacion de señales de reloj y sincronismo 166

B22. SYSTEM: Sistema completo y banco de pruebas 167

B23. VESTIM: Estimador de velocidad 170

Apéndice C. Descripciones VHDL del banco de pruebas 171

C1. CAMERA: Cámara digital 171

C2. MONITOR: Monitor digital 173

Resumen

En esta Tesis se propone una arquitectura para la determinación del flujo óptico a partir de secuencias de imágenes en tiempo real. Se pretende llevar a cabo esta arquitectura utilizando básicamente dispositivos lógicos programables del tipo FPGA y por ello la arquitectura será desarrollada utilizando un lenguaje de descripción de *hardware*, concretamente el VHDL.

El flujo óptico es un componente importante en el desarrollo de sistemas de visión que tienen que ver con el análisis del movimiento. Sus aplicaciones son muy variadas y van desde la caracterización dinámica de objetos presentes en las imágenes hasta la determinación de las características tridimensionales del entorno, pasando por la reconstrucción y mejora de imágenes.

Entre las características más importantes que se han dispuesto para el sistema están las de la búsqueda del procesamiento en tiempo real y a una resolución adecuada, similares a las que se obtienen en una cámara de televisión estándar.

Para el desarrollo de esta Tesis fue necesario crear una serie de herramientas que permiten trabajar con sistemas para el procesamiento de imágenes desarrollados con VHDL, además de plantear una metodología para desarrollar un sistema *hardware* a partir de un modelo matemático como el utilizado para la obtención del flujo óptico.

Abstract

In this Thesis an architecture to determine the optical flow from image sequences in real-time has been proposed. This architecture will be developed using basically programmable logic devices, specifically FPGAs. By that reason the architecture will be developed using a hardware description language, concretely the VHDL.

The optical flow is an important component in the development of vision systems related with the motion analysis. The applications of the optical flow are diverse, going from the dynamic characterization of objects in images to the determination of the three-dimensional characteristics of environment, passing through the reconstruction and improvement of images.

Most important constrains imposed to the system was the real time processing and a suitable image resolution, similar to the obtained from an standard television camera.

For the development of this Thesis it was necessary to create a set of tools that allow to work with image processing systems developed with VHDL. In addition a methodology to develop a system hardware from a mathematical model has been presented, like the used to obtain the optical flow.

Laburpena

Denbora errealean irudi-sekuentzitatik fluxu optikoa ateratzeko arkitektura bat proposatzen da Tesi honetan. Arkitektura hau FPGA motako gailu logiko programableak erabilita egin nahi da, horregatik *hardwarea* deskribatzeko lengoaia bat erabiliko da arkitektura hau garatzeko, VHDL hain zuzen.

Fluxu optikoa mugimendu analisiarekin zerikusia duten ikus sistemak garatzeko osagai berezi bat da. Era askotakoak dira bere aplikazioak, irudietan agertzen diren gauzen karakterizazio dinamikotik hasita eta ingurune tridimentsionalen ezagutza lortzeraino, irudien hobekuntza eta berrerakutzatik pasatuta.

Sistemari inposatutako ezaugarrien artean, denbora errealen eta zehaztasun egokiaz lan egitea azpimarra dezakegu. Telebista kamera estandar batek dituen antzeko ezaugarriak bilatu dira

Tesia hau aurrera eramateko ezinbestekoa izan da VHDL lenguaiaz deskribatutako irudien prozesamendurako sistemekin lan egitea aholbidetzen duten zenbait tresna sortzea. Gainera eredu matematiko batetik, *hardware* sistema bat garatzeko metodologia proposatu da, fluxu optiko ateratzeko egin dan bezala.

Agradecimientos

Debo agradecer a todos los profesores que durante estos años se esforzaron en enseñarme parte de lo que sabían:

Gerardo Aranguren A.
Esteban Azaceta F.
Xabier Basogain O.
José Miguel de Diego R.
Iñaki Goiricelaia O.
Luis Antonio L. Nozal
José Luis Martín G.
Juan Luis Ordiales B.
Jesús Mª Ruiz O.
Eduardo Zabala L.

Pero entre ellos debo agradecer especialmente a José Luis Martín, mi director de Tesis, por su apoyo en las distintas fases de la realización de la Tesis. A Gerardo Aranguren y Luis L. Nozal por su gran apoyo, confianza y estímulo a lo largo de estos años.

A mi familia más cercana, mi madre y mis hermanos Xabier e Iñaki, que a pesar de la distancia física que nos ha separado han estado a mi lado brindándome todo el apoyo posible gracias a ese gran aliado tecnológico: Internet.

Al resto de mi familia, mis tíos y primos, que han estado a mi lado ayudándome en los momentos más difíciles de mi adaptación al nuevo entorno físico, en especial a toda la familia Ubillos Izaguirre.

También debo agradecer a todos esos amigos que me han ayudado, animado y apoyado en múltiples aspectos durante la realización de esta Tesis Doctoral, y en especial a Alberto Chavarría, Juan Carlos Sanz y Angel Uría.

A todo el grupo de profesores y alumnos del Grupo de Diseño Electrónico GDE, tanto a los que se encuentran ahora en él como a los que pasaron por él en estos últimos años, algunos de los cuales han ayudado directamente en esta Tesis. Entre ellos debo reconocer el apoyo que desinteresadamente me han brindado Unai Bidarte y Jaime Jimenez.

También he de agradecer a las instituciones por la ayuda económica que han prestado para la realización de este doctorado, al Ministerio de Educación y Ciencia de España por su beca "Reina Sofía", a la Universidad del País Vasco / Euskal Herriko Unibertsitatea por su beca de Investigación Predoctoral y al Departamento de Educación, Universidades e Investigación del Gobierno Vasco por su Ayuda a Proyectos de Investigación Básica y Aplicada (PI-1998-41).

Bilbao, 04 de Octubre de 2001

1

Capítulo 1

Introducción

Se ha dicho que la ciencia deshumanizará a la gente
y la convertirá en números.
Esto es falso, trágicamente falso.
[En el campo de concentración y crematorio de Auschwitz fue]
donde la gente se convirtió en números

Jacob Bronowski
El ascenso del hombre, 1973

1.1. Marco de realización y justificación de la Tesis

Esta Tesis se ha desarrollado en el Grupo de Diseño Electrónico del Departamento de Electrónica y Telecomunicaciones de la Universidad del País Vasco. Fundamentalmente, las líneas de investigación de este grupo han estado orientadas al desarrollo de arquitecturas y sistemas de procesamiento electrónicos a alta velocidad.

Aunque hoy día la mayoría de los sistemas electrónicos avanzados se basan en el uso de procesadores, y aunque en la actualidad estos alcanzan un nivel muy avanzado de prestaciones y velocidad, el procesamiento de señales a alta velocidad impone muchas veces restricciones muy difíciles de cumplir con el uso exclusivo de procesadores. Por ello se hace necesario el uso de otras tecnologías como son los dispositivos lógicos programables.

En el Grupo de Diseño Electrónico, el desarrollo de sistemas de procesamiento alta velocidad nos ha llevado al uso extensivo tanto de microprocesadores como dispositivos lógicos programables. Más específicamente, las investigaciones se realizan en los siguientes campos de aplicación:

- Visión artificial
- Comunicaciones digitales
- Sistemas de adquisición de datos y control

Específicamente, en el campo de la visión artificial se han realizado varios proyectos, de los cuales se pueden mencionar los siguientes:

- Tarjeta pre-procesadora para visión artificial con conexión al puerto EPP.
- Circuito integrado convertidor de RGB en HSI para aplicaciones de visión artificial en color real.
- Procesado automático e inspección de ensamblaje por visión 3D (*Automated process and assembly inspection by 3D vision*). (ESPRIT 2017 - TRIOS).
- Convolucionador paralelo para aplicaciones de visión artificial.
- Proyecto de investigación sobre movimiento captado en una secuencia de imágenes: Visión Artificial Dinámica (VARDIN - UPV 147.345-TA029/95).
- Reconocimiento de caracteres para aplicaciones de voto electrónico.

La mayoría de estos proyectos trataron de visión artificial estática y sólo en los últimos años el Grupo de Diseño Electrónico viene dando sus primeros pasos en visión artificial dinámica, la cual representa mayores retos al igual que un mayor campo de aplicaciones.

Fruto de ese interés en la visión artificial dinámica surge esta Tesis. Se han hecho algunos estudios referidos a las posibilidades de la determinación del movimiento a partir de secuencias de imágenes y de la información que puede ser extraída del mismo. En particular se busca, en concordancia con el área general de investigaciones del grupo, el procesamiento en tiempo real. Entre estas investigaciones surge como pieza de importancia la determinación del flujo óptico y con ello el tema básico de esta Tesis, la cual pasó a formar parte de un proyecto de investigación auspiciado por el Gobierno Vasco (PI-1994-41).

Dadas las exigencias del procesamiento de imágenes en tiempo real, se hace necesario el uso de dispositivos lógicos programables con los cuales es posible alcanzar grandes velocidades de procesamiento.

A través de los años, la manera de trabajar con los dispositivos lógicos programables ha ido cambiando continuamente. Las tendencias más recientes van encaminadas al uso del lenguaje de descripción de *hardware* VHDL, y varios de los esfuerzos realizados en el grupo han estado dirigidos a aprender y desarrollar los conocimientos relacionados con este lenguaje.

Esta Tesis nace de la fusión de esas dos inquietudes presentes: la adquisición de experiencia en la metodología de uso de los lenguajes de descripción de *hardware* y el estudio de métodos para el procesamiento de secuencias de imágenes.

1.2. Objetivos de la Tesis

El objetivo fundamental de esta Tesis Doctoral es el *modelado por medio de un lenguaje de descripción de hardware de un sistema para la determinación del flujo óptico a partir de secuencias de imágenes en tiempo real*.

Para llevar a cabo el objetivo propuesto es preciso cumplir con los siguientes objetivos parciales:

- Estudio de las posibles arquitecturas electrónicas para la extracción de información relativa al movimiento a partir de secuencias de imágenes, y en especial aquellas encaminadas a la extracción del flujo óptico.

- Diseño de una arquitectura específica para la determinación del flujo óptico.

- Modelado en lenguaje de descripción de *hardware* VHDL de las diferentes partes que componen el sistema para la determinación del flujo óptico.

- Prueba del sistema con secuencias de imágenes y análisis de los resultados con vistas a una posible síntesis del modelo en circuitos integrados programables.

La Tesis aquí propuesta dejará como resultados concretos:

- Un estudio de las posibles vías para la determinación del movimiento que puede servir de base para otros trabajos que se realicen con objetivos similares.

- Modelos de diversas piezas de uso común en el campo de la visión artificial

- Un modelo sintetizable del sistema que permitirá la creación de un sistema para la determinación del flujo óptico utilizando dispositivos lógicos programables.

Para llevar a cabo los objetivos definidos para la Tesis aquí propuesta seguirá la metodología que se propone a continuación:

En primer lugar, es preciso hacer un estudio de las diversas opciones habitualmente aplicadas en la determinación del movimiento a partir de secuencias de imágenes, entre las cuales se seleccionará una de las opciones en vistas a su adaptabilidad a las tecnologías disponibles en la actualidad para el procesamiento en tiempo real.

En segundo lugar se establecerá en concreto un algoritmo para la determinación del movimiento a partir de secuencias de imágenes y se elaborará el diagrama a bloques de un sistema electrónico que pueda llevar a cabo tal algoritmo.

En tercer lugar, antes de proceder al modelado del sistema por medio de un lenguaje de descripción de *hardware*, será preciso estudiar las posibilidades de los diferentes sintetizadores de lenguaje VHDL disponibles, y adaptarlos para el trabajo que se pretende realizar. Hay que remarcar que las herramientas para trabajar en lenguaje VHDL son nuevas y presentan muchas diferencias con respecto a las operaciones que soportan. Además, es necesario preparar un sistema de simulación y estudio acorde con las características de los sistemas de visión que se vienen desarrollando.

En último lugar se modelarán y probarán las diferentes partes que formarían el sistema para la determinación del flujo óptico a partir de secuencias de imágenes.

1.3. Estructura de la Tesis

Esta memoria de Tesis Doctoral está organizada siguiendo la clásica estructura establecida en el método científico, es decir, identificar el problema, recopilar información, sugerir una solución al problema, experimentar, analizar los resultados y establecer unas conclusiones.

En este capítulo 1, se hace una introducción a los temas tratados en la Tesis, se establece el problema y a partir de ello se establecen los objetivos de la Tesis.

La información referente al problema se recopila y se analiza en el capítulo 2. Como el lector podrá concluir de la lectura de ese capítulo, existen tantas aplicaciones como métodos en el tema de la determinación del movimiento a partir de secuencias de imágenes. En estos métodos, algunos buscan la extracción del flujo óptico, el cual se requiere para algunas de las aplicaciones.

El capítulo 3 está dedicado a la recopilación de la información que no está directamente relacionada con el problema, pero que se requiere para el desarrollo de los experimentos y las tecnologías que son aplicables a la resolución del problema.

La solución al problema se propone en el capítulo 4, y en él se desarrollan todas las partes que van a permitir crear el sistema electrónico. En este capítulo se establecen también las

diferentes aproximaciones y ajustes que es necesario establecer para poder implementar los algoritmos matemáticos en los sistemas electrónicos físicos disponibles en la actualidad.

El capítulo 5 contiene los diferentes experimentos realizados y un análisis de los resultados de los mismos.

Por último en el capítulo 6 se presentan las conclusiones de la Tesis y se trazan las vías para nuevos desarrollos futuros o mejoras que es posible realizar sobre el sistema aquí desarrollado. En este capítulo se anexan además los diferentes trabajos publicados en revistas y congresos como resultado de la Tesis.

2

Capítulo 2

Determinación del flujo óptico

Los circuitos de los computadores transmiten la información a la velocidad de la luz. En el hombre, la transmisión neuronal es un millón de veces más lenta. El hecho de que en las operaciones no aritméticas el cerebro humano, más pequeño y menos rápido, opere con mayor eficacia que los grandes y rapidísimos computadores electrónicos constituye un formidable tributo a la sutileza con que ha sido circuitado y programado nuestro cerebro.

Carl Sagan.
Los dragones del Edén, 1977

2.1. Introducción al tema de la Tesis

La invención de la cámara de vídeo electrónica en 1923 abrió las puertas de una cantidad infinita de aplicaciones, comenzando por la simple transmisión de televisión y finalizando en la visión artificial de las máquinas. Hoy en día puede decirse sin lugar a dudas que la industria relacionada con la manipulación de imágenes animadas es una de las áreas de negocios más importantes del mundo. Como un dato simple puede reseñarse que en 1983 la *International Telecommunication Union* determinó que en el mundo existían 565 millones de teléfonos y 600 millones de televisores [PETE98].

Aunque es la televisión la primera y más importante aplicación de las tecnologías del procesamiento de imágenes animadas, pueden considerarse tres grandes áreas de aplicación: la mejora de imágenes para la interpretación humana, la compresión de imágenes para su transmisión o almacenamiento y la interpretación de imágenes por máquinas autónomas [GONZ77] [JAIN97].

Históricamente, el desarrollo de las tres áreas de aplicación ha sido en el mismo orden en que se han señalado arriba. Inicialmente los esfuerzos se centraron en mejorar las transmisiones de televisión introduciendo mejoras en la resolución y el color, luego se pasó a las aplicaciones científicas donde se buscaba una mejora de imágenes procedentes de satélites, equipos médicos, etc. Este esfuerzo de mejora de imágenes aún continúa con la televisión de alta definición y la digitalización del cine, entre otros.

La compresión de imágenes nace a raíz del gran espacio de almacenamiento y de ocupación de canales de transmisión que requieren las imágenes animadas. Hace algunos años, la demanda de imágenes y de calidad era mucho menor, pero el aumento de las mismas requiere nuevas técnicas para aprovechar el limitado recurso de los canales de transmisión y de los medios de almacenamiento.

Gran parte de la mejora de imágenes está dirigida a ayudar a la interpretación de las mismas por seres humanos, pero dado que el número de imágenes a interpretar aumenta cada día, la habilidad de análisis humano de una manera consistente, fiable y rápida se deteriora.

Más reciente es el área de la interpretación de imágenes por máquinas autónomas o visión artificial. Si bien se hicieron varios intentos años atrás, se puede decir que hasta hace poco la tecnología no estaba lo suficientemente desarrollada como para permitir aplicaciones verdaderamente prácticas. El desarrollo de dispositivos integrados de alta velocidad y procesadores de instrucciones avanzadas y alta velocidad, ha abierto las puertas a la interpretación de imágenes por máquinas autónomas, aunque aún queda mucho camino por recorrer.

Tal y como se ha mencionado, una imagen tiene un gran contenido de información por sí misma, pero su interpretación agrega otra dimensión a la cantidad de información. De esta manera una simple foto tomada por un satélite no sólo proporciona una vista de la superficie terrestre sino que también, si es analizada con detenimiento, puede proporcionar información sobre tipos de vegetación, construcciones humanas, etc.

De la misma manera hay que considerar que una imagen animada, que no es otra cosa que una serie de imágenes en secuencia, proporciona una información muy superior y su interpretación también dará lugar a una mayor cantidad de información.

Desafortunadamente, una mayor cantidad de información a procesar trae consigo una serie de complicaciones para el procesamiento, y en mayor proporción si el flujo de información es constante como en el caso de las imágenes animadas.

En el campo de la interpretación de imágenes por máquinas autónomas, aún los pasos que se han dado han sido relativamente incipientes. Existen grandes complicaciones. Lo que para un ser vivo parece ser elemental, con máquinas es muy difícil llevarlo a cabo. En los últimos veinte años se han logrado varios avances, pero aún hay que considerar que queda mucho trabajo por hacer. Todavía está lejos el día en que un robot pueda pasearse por cualquier lugar y descifrar lo que le rodea. Sin embargo, es necesario ir estudiando en ello para lograr algún día acercarse al objetivo final.

Gran parte del trabajo de investigación que se realiza en el campo de la visión artificial se refiere a la creación de nuevos algoritmos y su realización por medio de avanzados procesadores. Sin embargo, las altas velocidades involucradas y el volumen de datos manejados obligan al uso de novedosas arquitecturas de procesamiento paralelo.

Generalmente, cuando se habla de procesamiento paralelo, se asocia con múltiples microprocesadores funcionando en paralelo, pero el término debe ser considerado en sentido más amplio. Un procesador puede ser un bloque de circuitos que realicen una operación básica, una suma, una división, etc. Estos "bloques procesadores" suelen ser más rápidos que los procesadores de propósito general como son los microprocesadores y requieren menos recursos pudiendo integrarse en dispositivos electrónicos programables del tipo FPGA (*Field Programmable Gate Array*) o CPLD (*Complex Programmable Gate Array*).

Ultimamente, los avances en la integración de dispositivos programables han llegado a niveles prácticamente impensables hace algunos años, por lo que la posibilidad de construir sistemas en un circuito integrado ("*system on a chip"*) es hoy día una realidad. El problema se presenta en los métodos para diseñar, transferir el diseño al circuito integrado y realizar las pruebas al diseño. Para ello los fabricantes han venido proporcionando una serie de herramientas, pero hasta hace relativamente poco no se disponía de una única herramienta que permitiera ser utilizada independientemente del dispositivo. Hoy en día los diferentes fabricantes han venido adoptando el lenguaje de descripción de *hardware* VHDL (*VHSIC Hardware Description Language*).

La utilización de un lenguaje estándar como el VHDL no sólo permite una independencia en la selección de dispositivos programables de diferentes fabricantes, sino que también permite la utilización de dispositivos de diferentes tecnologías y la utilización de nuevas herramientas de simulación y síntesis desarrolladas por distintos fabricantes de *software*.

En el campo de la visión artificial, la determinación del movimiento de objetos en secuencias de imágenes es un área donde se desarrollan muchos trabajos que tienen objetivos y métodos distintos. Uno de esos métodos es a través de la obtención del "flujo óptico denso", el cual busca la estimación del movimiento de cada uno de los puntos o *pixels* que componen las imágenes. Tal y como se verá más adelante en el desarrollo de esta memoria, las aplicaciones finales son variadas pero van dirigidas principalmente a la determinación de la estructura tridimensional del entorno.

2.2. Historia del procesamiento de imágenes

Las primeras aplicaciones del procesamiento de imágenes estuvieron encaminadas a la mejora de imágenes transmitidas para su uso en periódicos a comienzos de los años 1920. Mucho camino se ha recorrido desde entonces, pero aún continua siendo una de las aplicaciones con mayor campo, desde el coloreado de películas antiguas al desarrollo de nuevas técnicas de fotografía por satélite.

Básicamente la televisión no se considera un sistema de procesamiento de imágenes, sino un sistema de transmisión de imágenes, aunque bien es cierto que hoy en día ambos se encuentran estrechamente unidos.

2.2.1. La búsqueda de un sensor de imágenes: la cámara de vídeo.

El procesamiento de imágenes sería imposible sin la existencia de un sensor de imágenes, en otras palabras, un ojo artificial. La historia de los sensores de imágenes está íntimamente ligada a la historia de la televisión que comenzó a principios del siglo XX.

En 1873, Joseph May, descubre la variación de la conductividad eléctrica del selenio cuando es expuesto a la luz. Con ello se abre por primera vez la posibilidad de convertir la información luminosa de una imagen en información eléctrica [PETE98]. Esto estimuló las mentes de varios inventores para hallar maneras de tomar imágenes y reproducirlas en lugares más distantes. En 1875, George Carey, propuso la realización de un panel de fotocélulas de selenio para tomar los valores de luz de una imagen y conectar cada fotocélula a una lámpara en otro panel distante (figura 1). Esta idea nunca fue llevada a cabo porque era necesario una gran cantidad de cables para conectar el panel receptor con el emisor, pero contiene un detalle básico, la imagen es dividida en pequeños elementos o puntos conocidos hoy día como *picture-element* ó *pixel*.

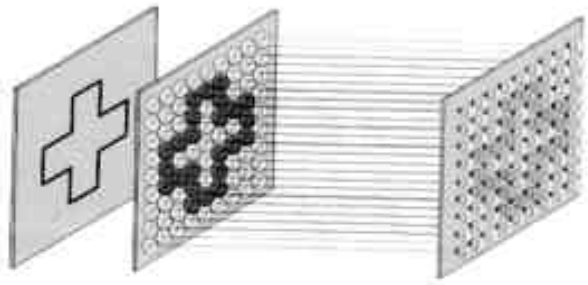


Figura 1. Propuesta de transmisión simultánea de G. Carey.

Un paso fundamental, aunque no siempre bien valorado, fue el establecido por Constantin Senlecq: la transmisión secuencial de imágenes (figura 2), utilizando una sola línea.

El estado de cada detector de selenio era leído con un contacto mecánico por un instante de tiempo, la señal se transmitía por un cable y otro contacto eléctrico activaba la

correspondiente lámpara en el lado emisor. Obviamente en aquellos tiempos no era posible construir contactos como el propuesto originalmente y el proyecto no era viable.

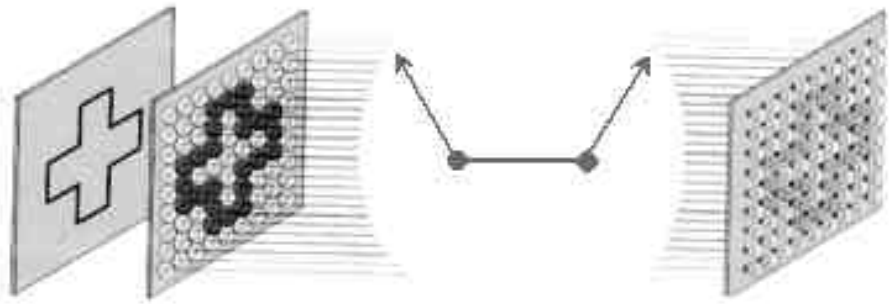


Figura 2. Propuesta de transmisión secuencial de C. Senlecq.

En 1884, Paul Nipkow, patenta un sistema de barrido mecánico de imágenes, el cuál puede observarse en forma simplificada en la figura 3. Es necesario señalar una diferencia crucial con respecto a la propuesta de G. Carey. En aquella la imagen era dividida en puntos o *pixels*, en la de Nipkow la imagen se divide en líneas continuas. Esta última es la base de la transmisión de imágenes analógicas de televisión, pero para el procesamiento digital de imágenes se utiliza la división por *pixels*.

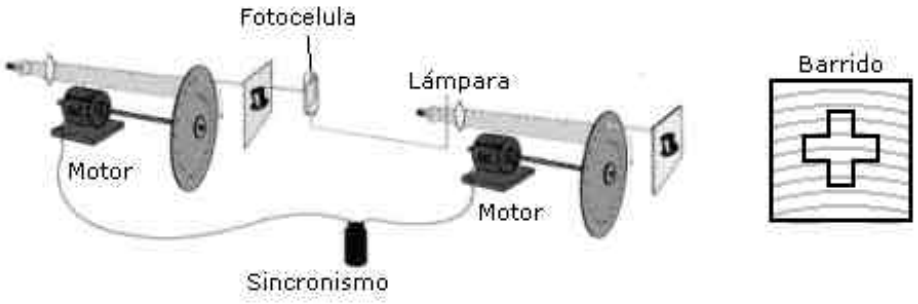


Figura 3. Sistema de barrido de imágenes de P. Nipkow.

Sin embargo la idea de Nipkow no era aún factible desde el punto de vista técnico. Las fotocélulas de selenio no eran capaces de responder con suficiente velocidad a la adquisición de imágenes por barrido, y no será hasta 1887 en que otro descubrimiento mejora significativamente el sistema de conversión de luz en electricidad. El descubrimiento en cuestión, el efecto fotoeléctrico, fue llevado a cabo por Heinrich Hertz.

Ya con los tubos fotoeléctricos era posible llevar a cabo la idea de Nipkow. Se construyeron algunos aparatos con carácter experimental, pero no dejaba de ser un sistema impracticable para aquella época por razones mecánicas (hoy en día existen cámaras basadas en este principio para aplicaciones especiales [MCBA00]).

En 1900 se celebra en París el primer Congreso Internacional de Electricidad y en él se utilizó por vez primera el término "*televisión*" para definir la transmisión de imágenes animadas mediante un sistema de comunicación [GARC86].

Otro paso significativo fue la idea de desarrollar todo el sistema de forma totalmente electrónica. La idea fue propuesta por A. Campbell Swinton en 1908. El dispositivo lector de imágenes propuesto era un tubo electrónico que leía la imagen desde una pantalla sensible por medio de un haz de electrones movido eléctricamente. El dispositivo para mostrar la imagen era otro tubo con una pantalla fluorescente que era activada también por un haz de electrones movido eléctricamente. La idea no fue viable hasta que en 1923, Vladimir Zworykin, inventa el "ionoscopio" (figura 4), el primer lector de imágenes animadas o cámara de vídeo. Ya se dispone del primer ojo electrónico propiamente dicho. Sin embargo, el ionoscopio tuvo que ser mejorado durante unos 12 años más antes de ser utilizado para las primeras transmisiones de televisión.

Figura 4. Sistema de lectura y reproducción de imágenes totalmente electrónico propuesto por V. Zworykin.

Al ionoscopio le siguieron muchas otras variedades de tubos electrónicos de vídeo tales como el super-emitrón (1934), el orthicón (1938), el image-orthicón (1946), el vidicón (1950), el plumbicón (1962), el saticón (1973) y el new-vicón (1980) [NAM99]. Todos estos tubos de vídeo registran niveles de luminosidad, por lo que para la visualización de imágenes en color es necesario colocar 3 tubos y un sistema de prismas, espejos y filtros de manera que cada uno de los tubos registre la luminosidad de uno de los colores básicos: rojo, verde y azul (figura 5) [BENS85a]. El sistema de cámaras a color con 3 tubos electrónicos comenzó a utilizarse en Estados Unidos en 1954.

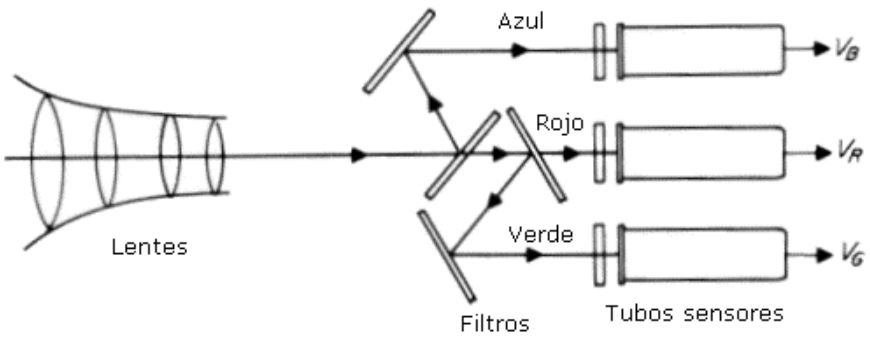


Figura 5. Visualización de imágenes en color.

A pesar de la larga y exitosa historia del tubo electrónico de vídeo, las investigaciones a partir de principios de los años 60 se centraban en la búsqueda de la cámara de estado sólido, con el objeto de solucionar los clásicos problemas de los tubos electrónicos: gran tamaño, fragilidad, envejecimiento, susceptibilidad a los campos eléctricos y magnéticos, etc.

El fruto de las investigaciones se comenzó a ver a principios de los años 70. En 1970, W. Boyle y G. Amelio de los laboratorios Bell inventaron el dispositivo de carga acoplada: *Charge Coupled Device* o CCD, para ser utilizado como memoria semiconductora pero que posteriormente (RCA Corp., 1973) sería utilizado como el sensor de vídeo por excelencia. Sin embargo, es necesario señalar también los sensores de imágenes CMOS que se desarrollaron paralelamente a los CCD y que gozan de gran popularidad por su reducido precio.

En la actualidad, la mayor parte de las cámaras de vídeo profesional y casero están basadas en dispositivos CCD, sin embargo en vista de los actuales avances en las tecnologías CMOS [ATME00], son muchos los que vaticinan un vuelco a la tecnología CMOS en un plazo relativamente corto [MAZO00]. Entre las ventajas que se esgrimen a favor de esta tecnología están las siguientes:

- Menor figura de ruido[1]. Hasta hace poco el desempeño de los circuitos CMOS en lo referente a figura de ruido era bastante pobre, pero en la actualidad éstos presentan niveles menores a los dispositivos CCD. Por ejemplo, las mejores figuras de ruido de los dispositivos CCD están por el orden de los 58 dB, y los mejores dispositivos CMOS logran hoy día niveles de 66 dB.

- Menores costos de producción. El proceso de producción de dispositivos CMOS es mucho más económico que el de los dispositivos CCD

- Alto grado de integración. Es posible una mayor integración de los componentes en la tecnología CMOS lo cual posibilita que en cuestión de muy pocos años se logren resoluciones mucho mayores que las obtenidas actualmente con la tecnología CCD (6 Mpixel). Ya se encuentran en producción dispositivos con resoluciones superiores a 1.3 Mpixel y se están desarrollando dispositivos de 4 Mpixel.

- Alta velocidad de salida. La velocidad de salida de imágenes de un dispositivo de captura de imágenes es proporcional a su resolución, y en los últimos años se ha convertido en un factor limitante que rápidamente está siendo superado. Así los actuales dispositivos de 1.3 Mpixels exhiben velocidades de 30 fps (cuadros por segundo) pero los próximos dispositivos CMOS de 4 Mpixel podrán alcanzar los 200 fps, convirtiéndolos así en los dispositivos más rápidos disponibles.

Aún los dispositivos CMOS tienen su desventaja más notable con respecto a los dispositivos CCD en lo referente a su sensibilidad. Los dispositivos CCD comerciales alcanzan sensibilidades de 0.05 lux, mientras que los mejores dispositivos CMOS tienen sensibilidades de 1 lux, aunque nuevas tecnologías tales como la iluminación trasera integral parecen ser la solución a esta desventaja.

[1] El término *figura de ruido* [NCS96] es un parámetro de mérito de un dispositivo, el cual compara el dispositivo con un dispositivo perfecto, es decir mide el ruido que introduce el dispositivo por las características de los materiales utilizados en su fabricación y no por el procesamiento que se desarrolla internamente.

Los sensores de imágenes normalmente están diseñados para dividir la imagen en *pixels* homogéneos en un sistema cartesiano de coordenadas, sin embargo, para aplicaciones de investigación también se desarrollan cámaras en coordenadas polares, donde la resolución en el centro de la imagen es superior a la resolución en la periferia [TIST94][PARD98]. También, en la misma línea están sensores donde un *pixel* está formado por 3 detectores ópticos desplazados para permitir una determinación del movimiento integrada, tal y como se cita en [KRAM96] y [ANDR91].

2.2.2. La Transmisión de imágenes

El primer intento con éxito de codificar y enviar imágenes fijas fue realizado por Casselli en 1862 [GARC86].

Tal y como ha sido mencionado, C. Senlecq fue el primero en establecer el concepto de que las imágenes debían transmitirse en forma secuencial a través de un solo canal. Pero a pesar de ser un concepto claro, quedan abiertas innumerables posibilidades tal y como se demostró a lo largo de la historia de la televisión, y que hoy en día aún seguimos arrastrando.

Las señales utilizadas para la transmisión de imágenes animadas reflejan en parte el sistema de barrido propuesto por P. Nipkow, con la salvedad de que no es un barrido curvo. La imagen se divide en líneas rectas, es decir, la discretización de la imagen es por líneas no por puntos. La resolución horizontal de la imagen está limitada por el ancho de banda del canal de transmisión. Otro parámetro de importancia al trabajar con imágenes animadas es el número de imágenes que se presentan por unidad de tiempo.

Tanto el número de líneas como el número de imágenes por unidad de tiempo utilizados en la transmisión de imágenes ha variado a lo largo de la historia, sin embargo, dos estándares son los que hoy día se utilizan en televisión, el de 525 líneas a 60 cuadros por segundo (estándar NTSC/EIA470 adoptado desde 1941 en Estados Unidos) y el de 625 líneas a 50 cuadros por segundo (PAL/SECAM/CCIR adoptado desde 1952 en Europa). Estos estándares fueron seleccionados ajustándose a características del ojo humano y a la manera en que es utilizado un aparato de televisión, entre ellos el ser visto desde una distancia mayor a siete veces la altura de la pantalla [SILB99]. Estas condiciones no son las mismas de un monitor de ordenador, por lo han surgido nuevos estándares que vienen a complicar el panorama.

La transmisión secuencial de imágenes requiere una serie de señales para enviar la información y el sincronismo de los sistemas involucrados. Básicamente tres señales pueden ser consideradas:

- La señal de información de nivel de brillo (luminancia) y color (crominancia).
- La señal de sincronismo horizontal que indica el comienzo de barrido de una nueva línea.
- La señal de sincronismo vertical que indica el comienzo de barrido de un nuevo cuadro.

En la señal analógica de vídeo estas tres señales se combinan en una sola, y se suele conocer con el nombre de “vídeo compuesto” (figura 6). Existen ciertas variaciones en la señal de

vídeo compuesto en lo referente a la transmisión de color, según sea NTSC, PAL o SECAM, pero aún así son bastante similares.

A pesar de su antigüedad, la señal de vídeo compuesto es aún hoy día la más utilizada para la transmisión de imágenes de vídeo. Es el estándar para cámaras, monitores, grabadores, etc.

La aparición de los sistemas digitales hace necesario un método de transmisión acorde a la nueva tecnología. A pesar de que comienzan a mencionarse sistemas de televisión digitales, en la actualidad las mayores aplicaciones están en los estudios digitales de televisión. En ellos se utiliza un sistema de señalización contenido en la recomendación CCIR-601 [BENS85b].

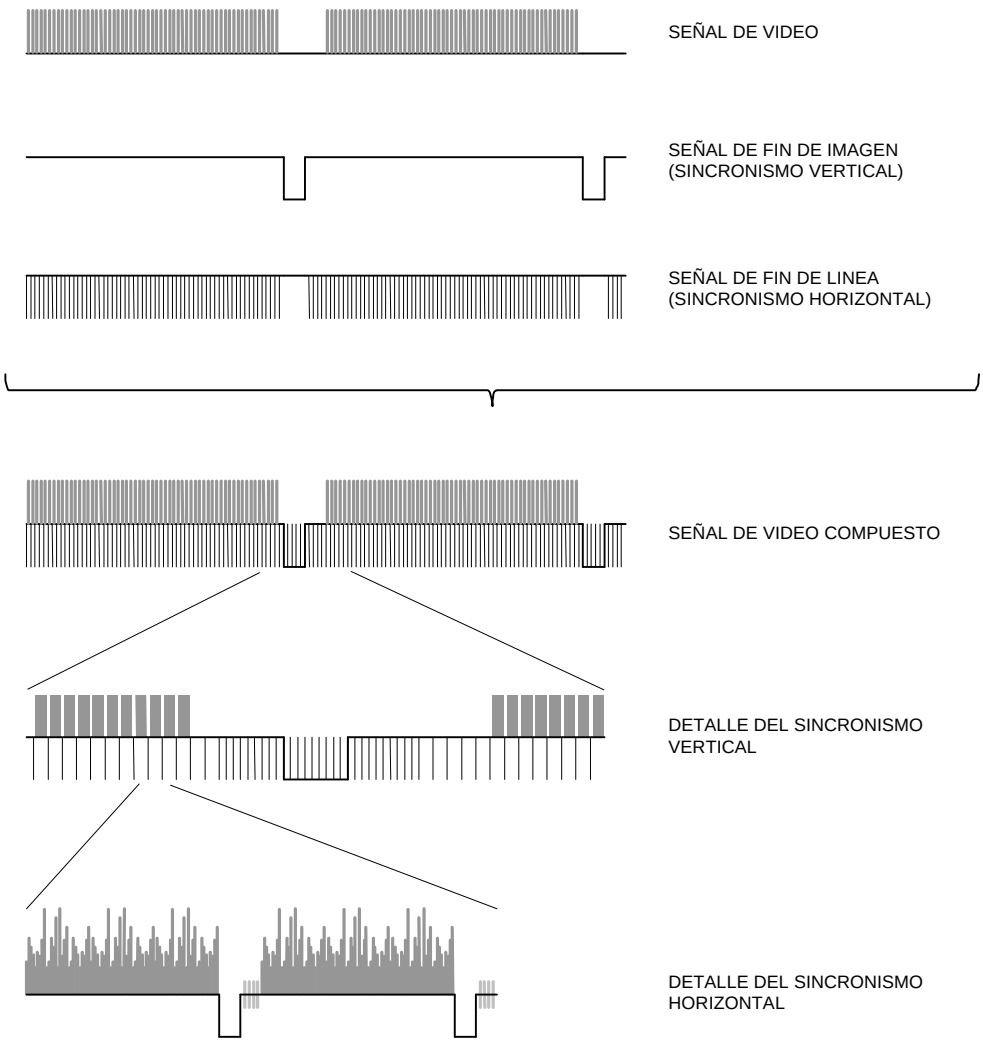


Figura 6. Señal de vídeo compuesto formada por la combinación de las señales básicas de vídeo.

El sistema de transmisión básico empleado para la transmisión digital de vídeo es el conocido como *bit-parallel* que utiliza 9 pares de conductores (cada señal se envía en forma diferencial para hacer menos susceptibles las señales a ruido externo). Ocho pares se utilizan para transmitir la información digital y un par adicional para transmitir el reloj básico de la transmisión.

En la tabla 1 se muestran los parámetros básicos del sistema de vídeo digital comúnmente utilizado en la actualidad en los estudios de televisión. En ella se observan ciertos parámetros que conviene explicar un poco pero sin entrar en mucho detalle.

Las imágenes en color en los sistemas de vídeo se han codificado tradicionalmente bajo los parámetros de *Luminancia* y *Crominancia*, de manera que los primeros sistemas de televisión a color fueran compatibles con los sistemas de televisión en blanco y negro. La *luminancia* se refiere a la parte de la imagen asociada con la luminosidad, y la *crominancia* se refiere a la asociada con el color. La señal de crominancia posee dos componentes, una lleva información de la diferencia entre el color rojo y la luminancia, y la otra lleva información de la diferencia entre el color azul y la luminancia. Con estas señales es posible construir las imágenes a color.

Tabla 1. Parámetros básicos del sistema de vídeo digital.

Parámetro	**Sistema de 525 líneas**	**Sistema de 625 líneas**
Número de líneas activas	480	576
Número de cuadros por segundo	30	25
Señales codificadas	Luminancia (Y) Crominancia (Rojo - Y, Azul - Y)	
Sistema de codificación	4:2:2	
Número de muestras de luminancia por línea horizontal	720	
Número de muestras de crominancia por línea horizontal	360	
Frecuencia de muestreo de la señal de luminancia	13.5 MHz	
Frecuencia de muestreo de las señales de crominancia	6.75 MHz	
Cuantización de señal de luminancia	Uniformemente en 220 niveles (8 bits)	
Cuantización de señal de crominancia	Uniformemente en 225 niveles (8 bits)	

También es necesario considerar que el ojo humano es más sensible a la luminancia que a la crominancia. Por ello, es usual que se transmita más información referente a la primera. La relación de muestras digitales tomadas para cada señal se suele expresar por una terna de números, los cuales expresan el número de muestras de luminancia tomadas, el número de muestras de crominancia con respecto al rojo y el número de muestras de crominancia respecto al azul. Así, la terna 4:2:2 expresa que se toman cuatro muestras de luminancia, dos muestras de crominancia con respecto al rojo y dos muestras de crominancia respecto al azul (figura 7).

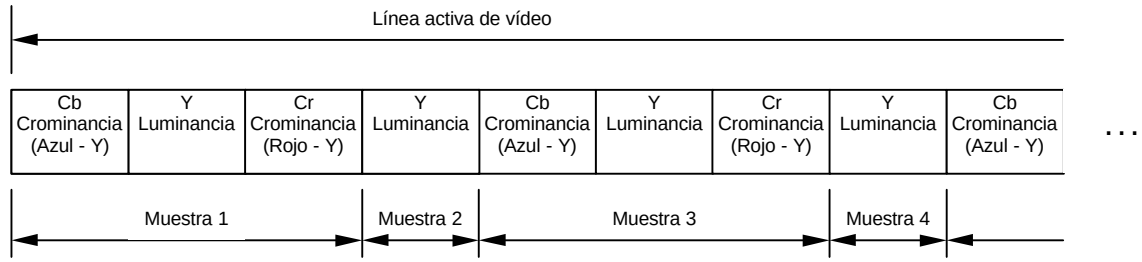


Figura 7. Muestras digitales de una señal de vídeo a color.

La información referente a los sincronismos horizontal y vertical se codifican en los mismos 8 pares de líneas que llevan las muestras digitales. Para ello, se utilizan dos códigos que no están permitidos para las señales de vídeo, el 00h y el FFh (obsérvese en la tabla 1 que los niveles de cuantización permitidos para las señales de 8 bits no alcanzan los 256 posibles). Utilizando estos códigos, se preparan grupos de bytes que contienen información referente a los sincronismos y otras informaciones auxiliares. En la figura 8 puede apreciarse el paquete de información enviado por cada línea de vídeo. Cada línea está compuesta de 1440 bytes de información referente al vídeo propiamente dicho (720 bytes de luminancia, 320 de crominancia referente al rojo y 320 de crominancia referente al azul) y 276 bytes de información de sincronismo y otros datos auxiliares.

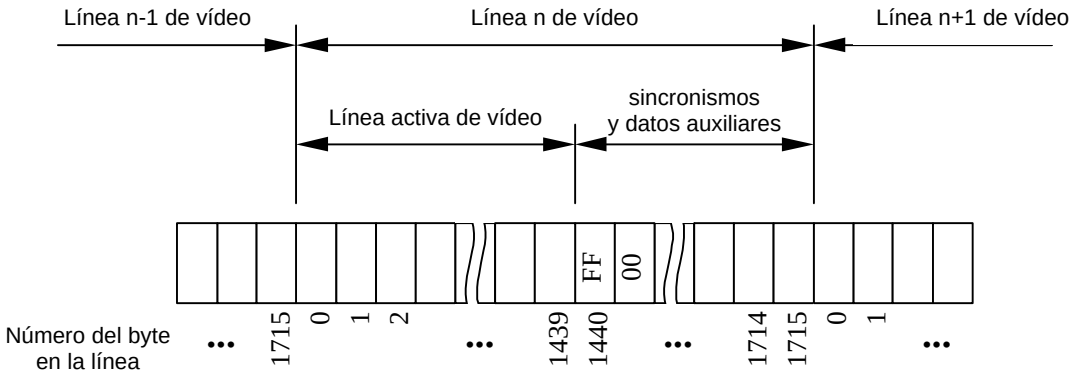


Figura 8. Transmisión de una línea de vídeo en forma digital.

Los métodos de transmisión analógico y digital discutidos se basan, tal y como se ha mencionado, en las señales de luminancia y crominancia que se utilizan por razones históricas de compatibilidad y de simplicidad para ciertas aplicaciones. Sin embargo, hoy día es frecuente encontrar aplicaciones basadas en las señales RGB (*Red-Green-Blue:* Rojo-Verde-Azul) principalmente en el campo de la informática.

En el caso de señales analógicas RGB, utilizadas principalmente en monitores, mediante tres señales similares a las de vídeo compuesto, se envían los tres colores primarios de los que se componen las imágenes.

2.2.3. La era digital y el procesamiento de imágenes

En un principio, la televisión era el eje fundamental de cualquier desarrollo referente a imágenes, pero la explosión de la informática ha traído consigo una variedad de formatos para la visualización de imágenes que influyen de mayor o menor grado en el procesamiento de imágenes. En la tabla 2 se pueden apreciar algunos de los formatos típicos de vídeo que se utilizan en la actualidad. En esta tabla se separan los formatos según sus aplicaciones fundamentales.

Tabla 2. Formatos típicos utilizados en vídeo digital.

Formato	**Descripción**	**Resolución** (*pixels:* horizontal x vertical)	**Refrescado** (Cuadros por segundo)
Sistemas de televisión			
CCIR 601	Televisión convencional	720 x 480 720 x 576	30 25
HDTV	Televisión de alta definición	1920 x 1152 1920 x 1152	60 50
Sistemas de compresión, transmisión y almacenado de imágenes			
SIF	Source Input Format Standard Interchange Format	352 x 240 352 x 288	30 25
MPEG-1		352 x 240 352 x 288	30 25
MPEG-2		720 x 480 720 x 576	30 25
Telecomunicaciones basadas en vídeo			
CIF	Common Intermediate Format	352 x 288	30
QCIF	Quarter Common Intermediate Format	176 x 144	30
Sistemas de informática			
VGA	Video Graphic array	640 x 480	60 a 75
SVGA	Super Video Graphic array	1024 x 768	45 a 72

Dos tecnologías emergentes, los gráficos por ordenador y la visión artificial han experimentado un creciente éxito en las últimas décadas como resultado de los avances en las arquitecturas de ordenadores, la computación distribuida, las capacidades de almacenamiento y las capacidades de las redes de ordenadores [JAIN97]. Los gráficos por ordenador involucran la visualización y síntesis de eventos y objetos complejos. La visión artificial se encarga de la interpretación automática de imágenes.

En 1989 el Gobierno de los Estados Unidos inició el Programa Federal de Computación de Alto Rendimiento (*The Federal High Performance Computing Program*). En éste se definía la visión artificial como:

"... el desarrollo de capacidades visuales de nivel humano para ordenadores y robots. La visión artificial requiere el procesamiento de imágenes, modelado de texturas y colores, procesamiento geométrico y racional, así como el modelado de objetos. Un sistema idóneo de

estructurada.”

Puede decirse que el procesamiento de imágenes surgió a raíz del programa espacial

60. En particular, se hizo necesaria la corrección de las imágenes de la luna transmitidas por la nave espacial Ranger 7 en 1964. El trabajo fue realizado por el
en Pasadena, California. Posteriormente se fueron desarrollando otros métodos para mejorar imágenes recibidas de otras misiones espaciales tales como y Apolo [GONZ96]

Del campo espacial, el procesamiento de imágenes pasó a nuevas áreas de aplicación: la medicina, la física, la astronomía, la arqueología o la industria. Procesados tales como el

son habituales hoy día. Todas estas aplicaciones van destinadas a la mejora de las imágenes para la interpretación humana. Un paso más allá en el procesamiento de imágenes consiste en

La interpretación de imágenes por máquinas autónomas requiere más capacidad de procesamiento y su desarrollo es aún incipiente, pero brinda grandes posibilidades futuras.

señalar las siguientes: el reconocimiento de caracteres, la inspección de productos industriales, el reconocimiento automático de huellas dactilares y la predicción del tiempo a

Aún son posibles muchas más aplicaciones de gran utilidad con la interpretación de imágenes por máquinas autónomas, pero el trabajo de investigación que se requiere es aún muy grande.

tridimensional del entorno son algunas de ellas.

En las secciones subsiguientes se hará una exploración estructurada de los conceptos y

el procesamiento de secuencias de imágenes que es el punto clave de esta Tesis.

2.3. Procesamiento de imágenes

Para el procesamiento de imágenes digitales es útil distinguir entre los dos tipos de *imagen* y la información tipo
[NOZA93].

Se dice que la información es del tipo cuando los datos que componen el paquete de información pueden organizarse en una matriz espacial. Como ejemplos se pueden citar la

conforma la imagen, o el campo de velocidades asignado a cada punto de una imagen, o la transformada de Fourier de una imagen.

Se dice que la información es del tipo *datos* cuando el paquete de información no puede ser considerado una matriz espacial. Como ejemplos se citan las coordenadas de los segmentos que componen una imagen, el nivel promedio de gris en una imagen, o el histograma de una imagen. El procesamiento de imagen puede clasificarse según el tipo de información de entrada y salida. De esta manera tenemos los siguientes casos:

El procesamiento es de tipo imagen-imagen cuando la entrada al proceso requiere una información del tipo imagen y la salida del mismo es una información del tipo imagen. Un ejemplo sería el cambio de niveles de gris de una imagen.

El procesamiento es de tipo imagen-dato cuando la entrada al proceso requiere una información del tipo imagen y la salida del mismo es una información del tipo dato. Un ejemplo sería la obtención del nivel promedio de gris de una imagen.

El procesamiento es de tipo dato-imagen cuando la entrada al proceso requiere una información del tipo dato y la salida del mismo es una información del tipo imagen. Un ejemplo sería la reconstrucción de una imagen a partir de las coordenadas tridimensionales de las aristas de los objetos.

El procesamiento es de tipo dato-dato cuando la entrada al proceso requiere una información del tipo dato y la salida del mismo es una información del tipo dato. Un ejemplo sería la determinación de las coordenadas del centro de un objeto representado por las coordenadas de sus aristas.

Eventualmente, pueden presentarse procesos que combinen, tanto en las entradas como en las salidas, imágenes y datos.

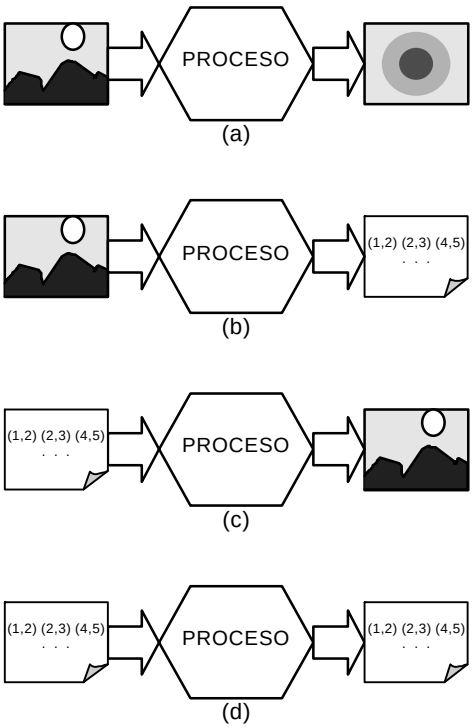


Figura 9. Clasificación de los algoritmos de procesamiento de imágenes: (a) imagen-imagen, (b) imagen-datos, (c) datos-imagen, (d) datos-datos.

El estudio de la visión dinámica puede considerarse en forma global como el procesamiento de una serie de imágenes cuyo resultado será una serie de datos. A la entrada del proceso se tiene una secuencia de imágenes, y a la salida se tienen datos tales como las coordenadas de los objetos, los parámetros que definen el movimiento de los objetos y en algunos casos una serie de datos que definen el entorno en un espacio tridimensional. Sin embargo el análisis de secuencias de imágenes no es una tarea sencilla de un solo paso, por lo cual es un hecho que el proceso tiene que ser dividido en subprocesos que pueden ser de tipos diversos: imagen-imagen, imagen-datos, o datos-datos (figura 9).

Las operaciones que se realizan sobre imágenes y que tienen como resultado otra imagen se pueden clasificar en tres grupos [VICE90]:

- **Operaciones puntuales:** Son aquellas en que el valor de cada *pixel G[x,y]* de la imagen resultante se obtiene a partir del valor del *pixel F[x,y]* de la imagen original, sin involucrar ningún otro *pixel*. Como ejemplos se pueden señalar la umbralización, el realce de contraste, la modificación del histograma, la conversión a pseudocolor y el cambio de color.

- **Operaciones locales:** Son aquellas en las que el valor de cada *pixel G[x,y]* de la imagen resultante se obtiene a partir de los valores del *pixel F[x,y]* y de sus vecinos en la imagen original. Eventualmente el concepto de operaciones locales puede ser extendido al campo de las tres dimensiones, es decir, incluyendo las secuencias temporales de imágenes. Entre operaciones locales básicas se pueden mencionar las convoluciones, las operaciones morfológicas y las operaciones booleanas locales. Como ejemplos se pueden señalar los filtros por convolución, los detectores de bordes, los uniformizadores de textura, etc.

- **Operaciones globales:** Son aquellas en las que el valor de cada *pixel G[x,y]* de la imagen resultante se obtiene de los valores del *pixel F[x,y]* y de todos los demás *pixels* en la imagen original. Como ejemplos de las operaciones globales se citan la transformada de Fourier, la del coseno, y sus respectivas antitransformadas.

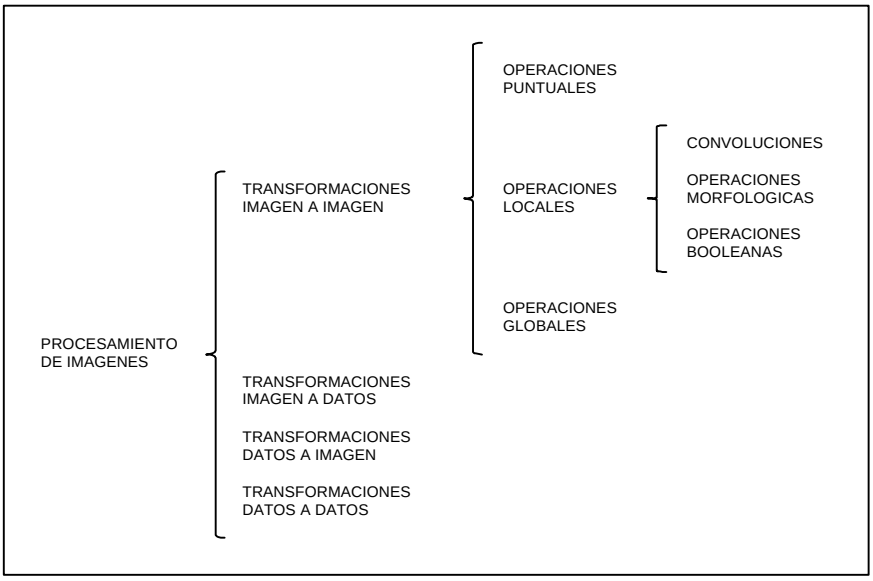


Figura 10. Tipos de procesamiento de imágenes.

Un sistema de visión artificial es un conjunto integrado de partes y procesos para el cual se

sensores disponibles, las restricciones de costo, velocidad de operación y el objetivo del sistema. En la se presenta el diagrama de bloques típico de un sistema de visión artificial en el que se esquematizan los procesos requeridos. Este diagrama es muy genérico

Figura 11. Representación esquemática de un sistema de visión artificial.

2.3.1. Secuencias de imágenes

Hasta el momento se ha hablado exclusivamente de procesamiento de imágenes individuales. Sin embargo, lo que más compete al estudio que se lleva a cabo en esta Tesis son las secuencias de imágenes, que muestran el movimiento de objetos en la escena.

Las imágenes en secuencia muestran los cambios que se producen en la escena a través del tiempo. En la sección 2.1 se hace referencia al número de imágenes por unidad de tiempo que se muestran en los sistemas de vídeo. Este número no es de ninguna manera una selección fortuita, sino que corresponde a la característica de persistencia de la visión humana [THIN99]. Este fenómeno de la persistencia de la visión fue descubierto el año 65 A.C. y consiste en la característica del ojo humano de almacenar por una décima de segundo la imagen de una escena iluminada con una luz brillante después de que la luz haya sido apagada. Por ello la sensación de movimiento que es observada cuando se presentan secuencias de imágenes a intervalos menores a una décima de segundo.

En vista de lo anterior, es necesario definir el término *tiempo real* aplicado al caso de las secuencias de imágenes.

Hay varias definiciones aplicadas a procesamiento en tiempo real y no existe un total consenso en una definición única. Sin embargo la definición siguiente puede aplicarse satisfactoriamente en el caso de las secuencias de imágenes.

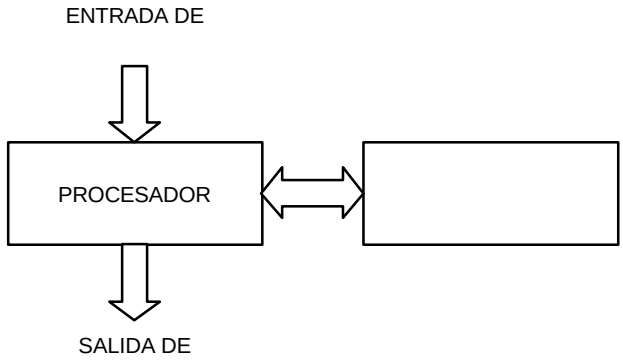
Se considera que un proceso se realiza en tiempo real cuando es capaz de obtener resultados aceptables en un tiempo predecible y suficientemente corto como para no perder la información asociada con los eventos que se procesan [JENS99].

Dado que el objetivo primordial del uso de las secuencias de imágenes es la visualización por el ser humano, se suele considerar *tiempo real* como la presentación de sucesivas imágenes a una velocidad tal que el ojo humano las interprete como imágenes en movimiento continuo. Esta velocidad suele ser de 25 a 30 imágenes por segundo, que es la utilizada comúnmente por las aplicaciones de vídeo.

2.3.2. Arquitecturas para el procesamiento de imágenes

La necesidad de procesar secuencias de imágenes en numerosas aplicaciones tanto científicas como industriales, a alta velocidad y a un costo aceptable, requiere un esfuerzo considerable dirigido al diseño de ingeniosas arquitecturas de *hardware*, bien sean basadas en procesadores de propósito general o en circuitos lógicos específicos [STOR89] [CROO99].

A pesar de los avances en la velocidad de los procesadores de propósito general actuales, aún continúan siendo lentos para aplicaciones de procesamiento de secuencias de imágenes, por lo menos cuando se utilizan en arquitecturas de procesador único (figura 12).



12. Arquitectura de procesador único.

En la actualidad parecen ser dos los caminos más viables en el procesamiento de secuencias de imágenes: el uso de poderosas arquitecturas de procesadores paralelos o el uso de arquitecturas segmentadas (*pipeline*) de procesadores o circuitos específicos (figura 13).

El uso de arquitecturas de procesadores paralelos ofrece una gran flexibilidad de programación y configuración pero por el momento resulta ser una posibilidad muy costosa para su utilización en la mayoría de las aplicaciones.

Las arquitecturas segmentadas, a pesar de ser menos flexibles que las arquitecturas basadas en procesador único o procesadores paralelos, producen excelentes resultados sin incurrir en grandes costos [ATHA95]. Por ello se utilizan en muchas aplicaciones de procesamiento de imágenes, especialmente en aquellas que se consideran como un preprocesamiento.

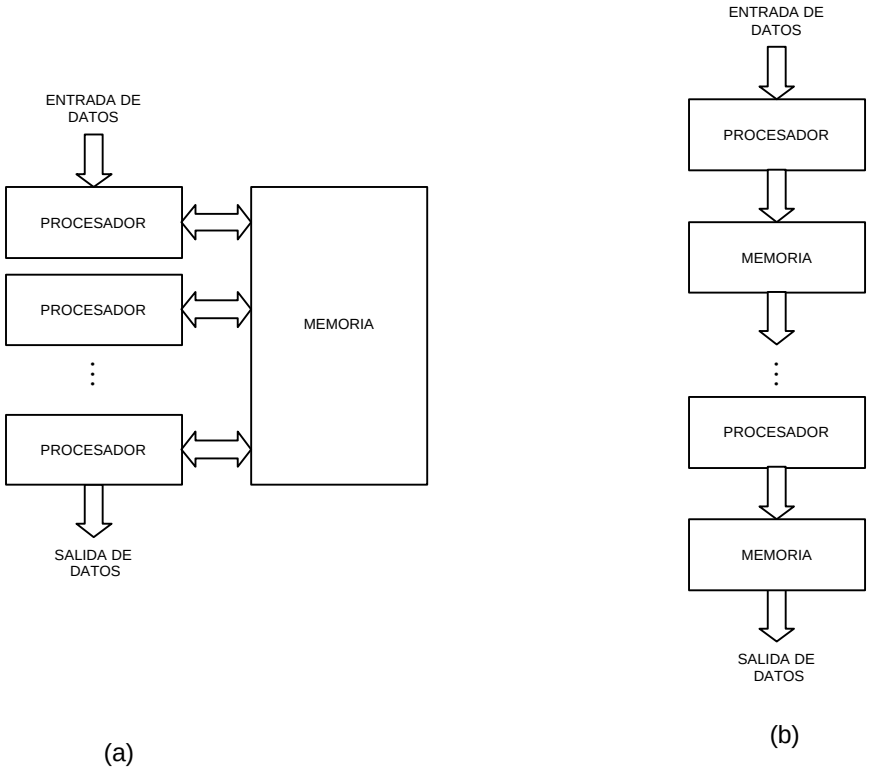


Figura 13. Arquitectura de procesadores paralelos (a) y arquitecturas segmentadas (b).

En una arquitectura basada en un procesador único, las diferentes subtareas en que se divide una tarea deben ser realizadas por el mismo procesador en tiempos distintos, y el tiempo total de procesamiento de la tarea es la suma de los tiempos de procesamiento de cada subtarea (figura 14). Para procesamientos cíclicos la velocidad máxima de entrada de datos a procesar está restringida por este tiempo total de procesamiento de la tarea.

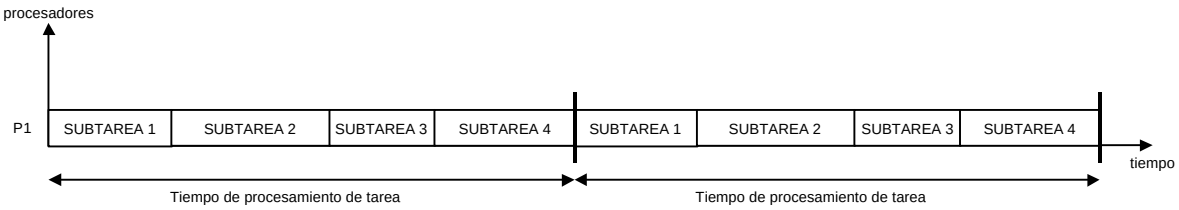


Figura 14. Procesamiento de tareas en una máquina de procesador único.

La arquitectura segmentada es una forma fácil y económica de realizar paralelismo temporal, y resulta particularmente útil para el procesamiento de imágenes, donde la información se presenta en forma serializada. El principio básico de la arquitectura segmentada es similar al de las líneas de montaje. La tarea debe dividirse en subtareas cada una de las cuales será ejecutada por una etapa de *hardware* especializada que actúe en concurrencia con las demás etapas [HWAN87]. En el diagrama espacio-tiempo de la figura 15 se observa cómo los distintos procesadores realizan las tareas asignadas de forma solapada. El tiempo de procesamiento de cada subtarea puede ser distinto, pero para que el sistema funcione de forma

síncrona en todas las etapas, todas ellas deben transferir sus datos a tiempos constantes, que definen el ciclo del sistema. En las arquitecturas segmentadas, el primer resultado del procesamiento se obtiene a un tiempo relativamente largo que viene fijado por la suma de los ciclos de tiempo de necesarios para efectuar todas las subtareas, pero a partir de ese momento, los resultados se obtienen cada ciclo de reloj. Ese tiempo inicial se denomina *latencia*.

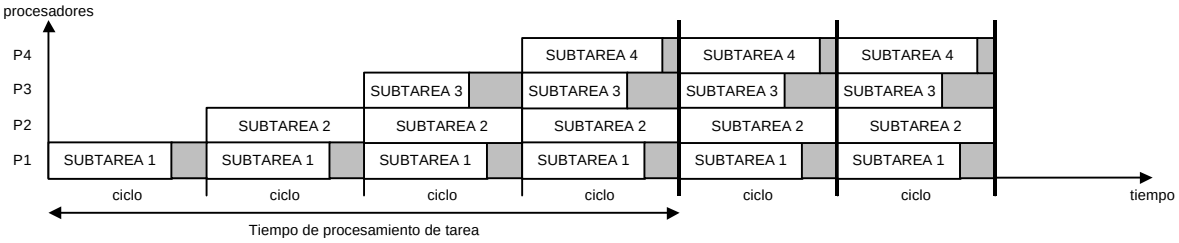


Figura 15. Diagrama espacio - tiempo de una arquitectura segmentada mostrando el solapamiento de subtareas.

En una arquitectura segmentada las etapas de procesamiento pueden ser grupos procesador-memoria como los de la figura 13b o circuitos combinacionales puros que efectúen operaciones aritmético-lógicas separadas mediante registros de acoplo (figura 16). Estos registros retienen los resultados intermedios entre las etapas. Tal y como se mencionó anteriormente es necesario que la transferencia de información entre las etapas se lleve a cabo en tiempos constantes, por ello los registros de acoplo son gobernados por un reloj común.

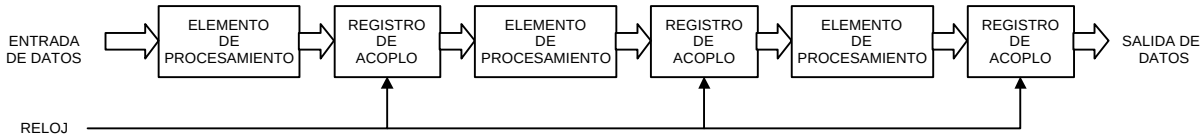


Figura 16. Estructura básica de una arquitectura segmentada.

Las arquitecturas segmentadas pueden diseñarse en estructuras ramificadas más complejas para resolver procesamientos más complejos [DATA99], inclusive pueden diseñarse en configuraciones realimentadas para permitir procesamientos elaborados (figura 17). También pueden realizarse estructuras configurables cambiando el flujo de datos entre los diferentes elementos de procesamiento.

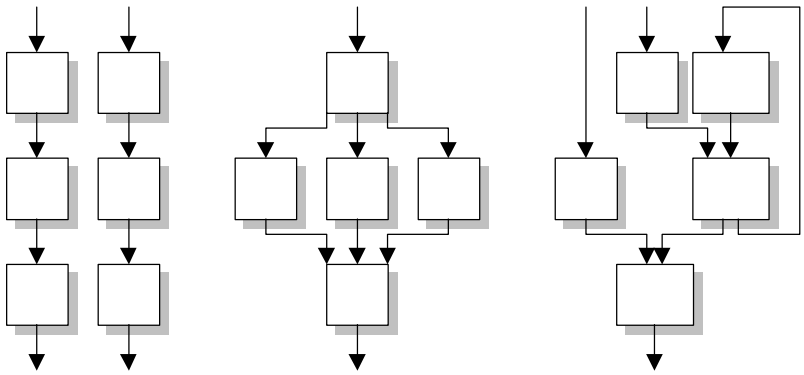


Figura 17. Estructuras complejas basadas en arquitecturas segmentadas.

La arquitectura segmentada resulta útil para el procesamiento de secuencias de imágenes por dos razones. En primer lugar, debido al carácter secuencial de la información que llega al sistema, y en segundo lugar porque muchas de las operaciones requeridas durante el procesamiento de imágenes no requieren el conocimiento completo del cuadro de la imagen. Son habituales en las arquitecturas segmentadas para el procesamiento de imágenes elementos procesadores del tipo mostrado en la figura 18, en la cual se representa un elemento que requiere 9 *pixels* de la imagen. Muchos de estos elementos de procesamiento son encontrados en la literatura referente al procesamiento de imágenes [STOR89] [BENI97] [NOZA91]. Algunas estructuras segmentadas utilizadas en el procesamiento de imágenes se citan a continuación.

McCubbrey y Laugheed [LOUG85] [MCCU85] desarrollaron una arquitectura segmentada para efectuar operaciones aritméticas, booleanas y morfológicas, utilizando para ello tres elementos procesadores y una memoria de imagen.

Luck [LUCK86] desarrolla un sistema en el cual divide las tareas de procesamiento en tres niveles. Un nivel bajo para operaciones tales como la detección de contornos, un nivel medio para operaciones como la transformada de Hough y un nivel alto para la determinación de detalles de la imagen. Para las operaciones de bajo nivel utiliza una arquitectura del tipo *segmentado.*

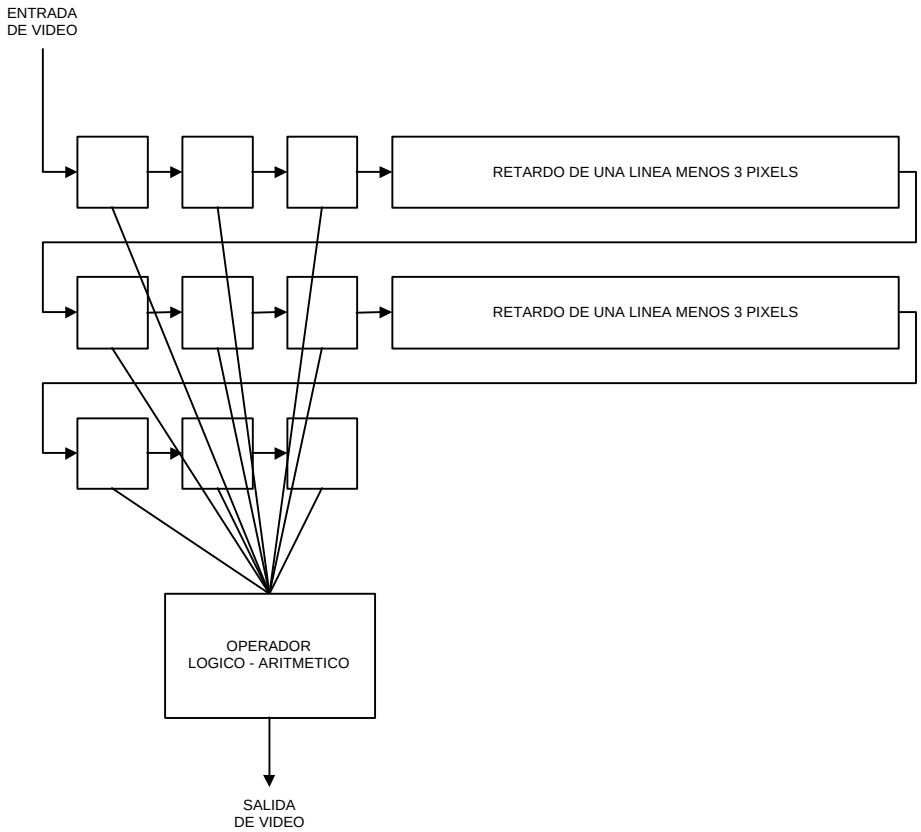


Figura 18. Elemento de procesamiento típico en procesamiento de imágenes.

Batchellor y Bowman [BATC86] [BOWM87] se encuentran con la necesidad de dividir las tareas de procesamiento de imágenes en dos: una de bajo nivel y otra de alto nivel. En el sistema desarrollado por Bowman, se utiliza un procesamiento de bajo nivel basado en una arquitectura segmentada y para el de alto nivel un procesador de propósito general.

2.4. El flujo óptico

El análisis del movimiento de los diferentes componentes de una escena puede ser abordado desde diferentes puntos de vista, pero básicamente pueden ser reducidos a dos: la determinación del *movimiento de objetos específicos* o la determinación del *flujo óptico*.

La determinación del *movimiento de objetos específicos*, es quizás más fácil de comprender intuitivamente, pero posiblemente más difícil de llevar a la práctica por medio de máquinas de visión artificial. Consiste en determinar en primer lugar los objetos que componen la escena y de los cuales se desea determinar el movimiento, y luego establecer las diferentes posiciones que adopta el objeto en el transcurso del tiempo.

La determinación del flujo óptico podría entenderse en una primera aproximación como la determinación del movimiento de cada punto en una escena[BEAU95] [NEGA98] [VERR89]. Sin embargo, dado que este tipo de determinación del movimiento es central en el desarrollo de la presente Tesis, es necesario ahondar más en el mismo.

En primer lugar, es necesario definir el concepto de *campo de movimiento*. Suponiendo una escena en el espacio tridimensional, el *campo de movimiento* es el campo vectorial de la velocidad de cada punto de la escena.

Cuando se observa una escena tridimensional por medio de un dispositivo de visión, bien sea un ojo o una cámara, se crea una representación bidimensional de la misma. De manera similar podemos definir un *campo de movimiento proyectado en el plano* en el que el *campo de movimiento*, que es esencialmente tridimensional, se proyecta sobre un plano para crear un campo bidimensional.

La recuperación del *campo de movimiento* parece ser una tarea esencial en muchos sistemas de visión, inclusive en la naturaleza. Sin embargo, el único dato disponible por medio de la visión es la variación espacial y temporal del patrón de brillo de la escena proyectada en un plano. De ellas sólo es posible obtener una aproximación del *campo de movimiento* que se denomina *flujo óptico*. Básicamente el *flujo óptico* puede definirse como el campo de movimiento del patrón de brillo de la escena proyectada.

Para ilustrarlo con mayor claridad se puede recurrir a la situación de la figura 19, en ella un cubo se desplaza con un movimiento horizontal y otro perpendicular a la pantalla de proyección. Se ha supuesto una proyección ortogonal para facilitar la explicación. Es obvio que todos los puntos del cubo se desplazan con el mismo movimiento, por tanto el *campo de movimiento* representaría todos los puntos del cubo con el mismo vector de velocidad asociado. Al proyectar el cubo sobre la pantalla, perdemos varias informaciones, primero la referente a los puntos del cubo que están por detrás, por que no hay forma de "proyectarlos"

por que son "tapados" por una de las caras del cubo. En segundo lugar, perdemos la noción del movimiento hacia la pantalla por ser una proyección ortogonal.

El movimiento horizontal es preservado en la proyección. Los puntos en el interior del cuadro tienen todos la misma velocidad y podemos por tanto definir un *campo de movimiento plano*. Ahora bien, considerando sólo lo que puede ser visto por los ojos o la cámara, se observa que un cambio de color gris-blanco se desplaza horizontalmente, y no es posible decir que los puntos en el interior o del exterior del cuadro se desplazan o no. Se pudiera decir que los puntos en el interior del cuadro se desplazan igual que los bordes o que en realidad son los puntos del exterior del cuadro que se desplazan con el borde del cuadro. No hay una forma certera de diferenciar un caso del otro. Lo que se puede ver es el cambio en los colores, es decir el *flujo óptico*, y de aquí se puede inferir algo del *campo de movimiento*, pero no deja de ser una aproximación.

El *campo de movimiento* y el *flujo óptico* son iguales sólo en el caso de que las variaciones espaciales del patrón de brillo correspondan a características estructurales de las superficies.

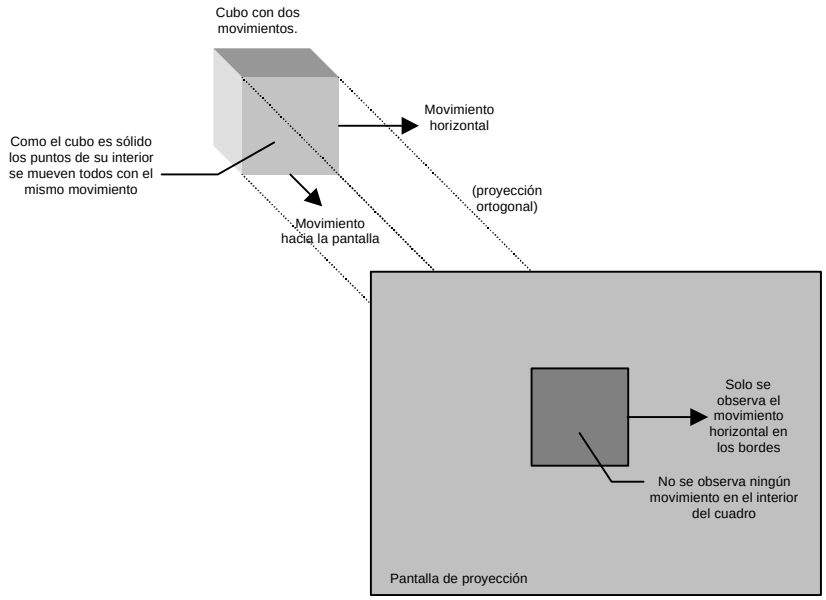


Figura 19. Ilustración de las diferencias entre *campo de movimiento* y *flujo óptico*.

Existe una tercera vía para el análisis del movimiento que puede ser considerada como un híbrido entre los dos planteados anteriormente, y que es necesario reseñar por su gran utilización actualmente. Este método suele denominarse en la literatura como *ajuste de bloques* [SANZ98]. Consiste en dividir la imagen en bloques uniformes y determinar el desplazamiento de los mismos en el tiempo. El procedimiento de división lo hace parecer a la determinación del *movimiento de objetos específicos*, pero en este caso no se divide la imagen en objetos específicos. También pudiera pensarse en este método como una determinación de *flujo óptico* de puntos muy grandes o muy baja resolución.

La utilización de una u otra forma de determinación del movimiento depende de las características de la aplicación en parámetros tales como tiempos máximos de procesamiento, densidad requerida de vectores de movimiento y tecnología a utilizar.

2.5. Aplicaciones de la determinación del flujo óptico

El movimiento es una fuente de información visual importante. La estimación del movimiento en imágenes, o en su sentido más estricto, del flujo óptico, tiene una gran cantidad de aplicaciones, que pueden clasificarse dentro de los siguientes grupos:

- **Recuperación de la estructura tridimensional a partir de secuencias de imágenes bidimensionales** (Estructura obtenida del movimiento, SFM), como es la detección de obstáculos para robots de navegación autónoma, la modelización del entorno, la adquisición automática de modelos para diseño asistido por ordenador (CAD), etc [VISE99].

- **Compresión y reconstrucción de secuencias de imágenes**, como son los casos de codificación y decodificación MPEG, y la reconstrucción de imágenes afectadas por ruido [SAHA00] [EGGL00].

- **Seguimiento y caracterización dinámica de objetos en movimiento** como en la detección de parámetros de tráfico de automóviles [BERT97] [FATH95], los sistemas de seguridad por visión artificial y en la previsión meteorológica.

Una imagen provee sólo información bidimensional del mundo tridimensional. Si se dispone de dos imágenes tomadas desde distintas ubicaciones espaciales, es posible modelar en parte el mundo tridimensional, en cuyo caso se dice que se dispone de dos imágenes estereoscópicas. Sin embargo, también desde secuencias de imágenes bidimensionales es posible extraer suficiente información como para modelar en cierto grado los objetos en movimiento (figura 20). La recuperación de la estructura tridimensional del entorno tiene amplios campos de aplicación en el diseño asistido por ordenador (CAD), ya que permitiría modelar objetos del mundo real en el lenguaje matemático de los ordenadores, para luego generar moldes o troqueles, analizar el acoplamiento entre piezas, etc. En este caso, el movimiento puede ser la cámara en vez del objeto, lo cual matemáticamente no tiene diferencia [SOAT98a] [SOAT98b].

La compresión de secuencias de imágenes es una de las aplicaciones más importantes de la estimación del movimiento a partir de imágenes y, además, es la aplicación más extendida. Las secuencias de imágenes están compuestas por objetos con diferentes movimientos. Cada cuadro de la secuencia de imágenes es muy parecida al anterior, a excepción de que algunos puntos de un cuadro cambian de posición para el siguiente, y de que pueden presentarse nuevos objetos en movimiento que entran en el campo de visión. De aquí que las aplicaciones de compresión de imágenes en movimiento se basen en indicar sólo los cambios de posición de aquellas partes de la imagen que cambian y mantener el resto [PARD94].

También con la información contenida en las secuencias de imágenes es posible reconstruir imágenes ruidosas, ya que parte de la información de una imagen se mantiene en la siguiente, posiblemente desplazada. Este caso es muy similar a la compresión de secuencias [LIMJ90]. También basados en el mismo principio se están comenzando a utilizar las secuencias de imágenes de baja resolución para obtener imágenes estáticas de alta resolución [EGGL00].

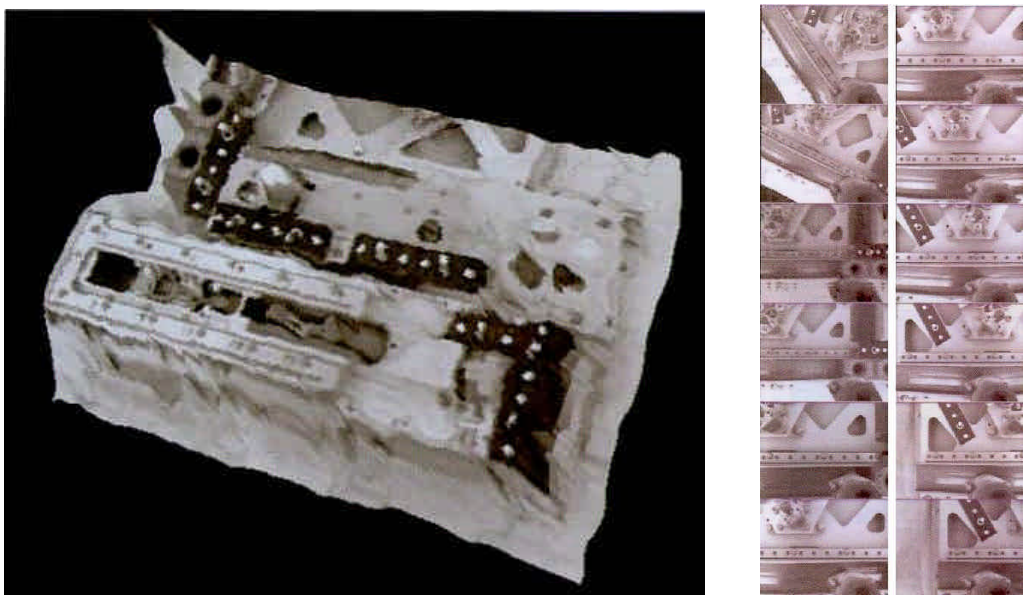


Figura 20. Imagen tridimensional obtenida a partir de una secuencia de imágenes en las pruebas del satélite ETS-VII [VISE99].

El seguimiento y caracterización dinámica de objetos en movimiento permite la utilización de la visión artificial en aplicaciones muy útiles y prometedoras en la que trabajan muchos investigadores. En aplicaciones como la mencionada con anterioridad de control de tráfico de automóviles [PARA00] [KOLL93], es posible ajustar automáticamente y a un bajo costo los sistemas de señalización en función de las características del tráfico. Pueden utilizarse también para aplicaciones de seguridad, donde un operario humano puede distraerse con facilidad. Entre las más recientes investigaciones está el seguimiento de rostros humanos para tareas como el reconocimiento de rostros, análisis de expresiones faciales y lectura de labios [LACA00] [ESSA97].

Otra aplicación muy utilizada en la actualidad del análisis de secuencias de imágenes de objetos en movimiento es el pronóstico meteorológico, con la diferencia de que las imágenes en este caso se producen a una velocidad mucho menor, y son complejas.

2.5.1. Recuperación de la estructura tridimensional

Sin duda la aplicación más interesante de la determinación del flujo óptico es la recuperación de la estructura tridimensional de las escenas. Si bien ya existen algunos incipientes ensayos como los ensayos realizados por la Agencia Espacial Europea con imágenes en las pruebas del satélite ETS-VII [VISE99] (figura 20), todavía es un área de aplicación poco estudiada. Trabajos en este sentido están siendo realizados por S. Avidan y A. Shashua [AVID00] del grupo *Vision Technology* de *Microsoft*, y P. Ho y R. Chung [HO00] del Departamento de Ingeniería Mecánica y Automatización de la *Chinese University of Hong Kong*.

2.5.2. Compresión de imágenes MPEG

La importancia de la compresión de imágenes es hoy día indudable. Por ello, muchos investigadores se afanan en esta labor. En la tabla 3 se presentan algunos datos que ponen de manifiesto los motivos para utilizar la compresión de imágenes, y en especial la compresión de secuencias de imágenes o vídeo [SAHA00]. En 1988, en respuesta a estas necesidades de compresión de vídeo, la Organización Internacional de Estándares (ISO) creó el Grupo de Expertos de Imágenes en Movimiento (MPEG), con el objetivo de establecer un formato común de codificación y almacenamiento de vídeo digital. Los estándares creados por este grupo de expertos no especifican las operaciones que deben efectuar los codificadores y decodificadores, sino la sintaxis del flujo de bits resultante.

Tabla 3. Comparación de tamaños de almacenamiento y ancho de banda requerido por diferentes datos multimedia.

Dato Multimedia	Tamaño / duración	Bits / Pixel ó Bits / Muestra	Tamaño sin comprimir (B: bytes)	Ancho de banda de transmisión (b: bits)	Tiempo de transmisión (utilizando un módem 28.8 Kbps)
Voz en calidad telefónica	10 seg	8 bps	80 KB	64 Kb/seg	22.2 seg
Imágenes en escala de grises	512 x 512	8 bpp	262 KB	2.1 Mb/image	1 min 13 seg
Imágenes a color	512 x 512	24 bpp	786 KB	6.29 Mb/image	3 min 39 seg
Imágenes médicas	2048 x 1680	12 bpp	5.16 MB	41.3 Mb/image	23 min 54 seg
Vídeo	640 x 480, 1 min (30 cuadros/seg)	24 bpp	1.66 GB	221 Mb/seg	5 días 8 h

El objetivo del proceso de codificación MPEG es comprimir secuencias de imágenes digitales sin pérdida aparente de calidad para disminuir de forma considerable el volumen de datos para transmisión o almacenamiento. Para ello se utilizan, simultáneamente, dos técnicas: la estimación del movimiento y la transformación del dominio.

Para la codificación de vídeo se están creando nuevos esquemas donde se persigue reducir el procesamiento a aquellas áreas de la imagen que lo requieren, tal y como son los contornos en movimiento [CINK98].

La compresión de vídeo basada en el movimiento de objetos produce mejores efectos que la codificación basada en bloques de tipo cuadros no adaptados a los objetos de la imagen, razón por la cual se observan bajas prestaciones en estándares tales como el MPEG y el H.261 a bajas velocidades. [HAN98]

2.5.3. Reconstrucción de imágenes

Son múltiples los casos en que se requiere extraer imágenes estáticas de secuencias de vídeo. Sin embargo, la calidad requerida para las imágenes de vídeo es bastante peor que la

requerida en imágenes estáticas. Esto es debido a la capacidad del cerebro humano de extraer la información redundante de las secuencias de imágenes. Utilizando el mismo principio, es posible extraer imágenes estáticas de alta resolución a partir de vídeo de baja resolución [EGGL00] (figura 21).

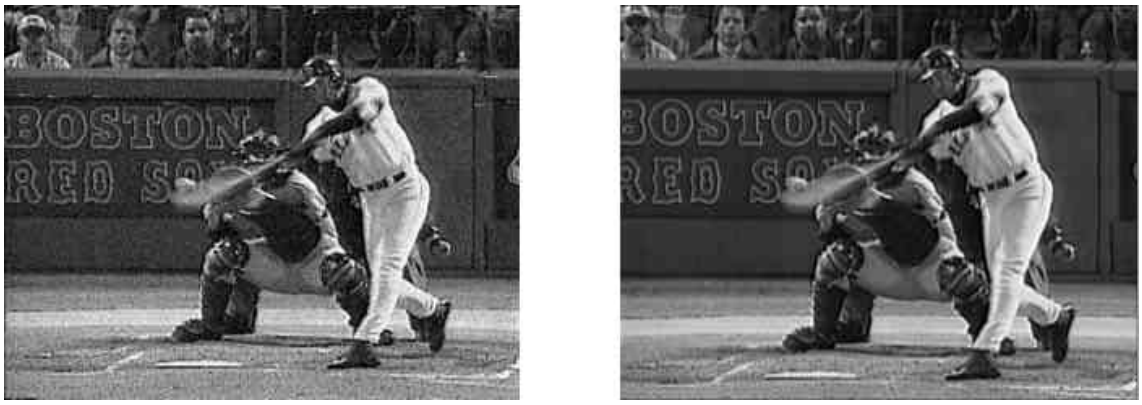


Figura 21. Obtención de imágenes de alta resolución a partir de secuencias de imágenes.

Con una tecnología basada en la determinación del movimiento por flujo óptico (figura 22) y desarrollada en el *MIT Media Lab*, ha sido posible obtener imágenes estáticas con una resolución de hasta 2000 x 3000 *pixels* a partir de imágenes de vídeo de baja resolución (640 x 480 *pixels*). Todo esto utilizando algoritmos *software* operando sobre plataformas Pentium.

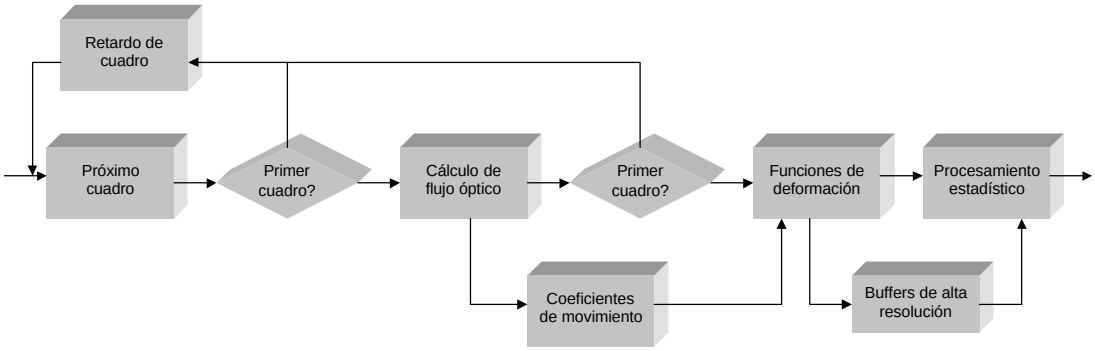


Figura 22. Algoritmo para la obtención de imágenes de alta resolución a partir de secuencias de imágenes.

2.5.4. Determinación de dinámica de fluidos

Los estudiosos de la dinámica de fluidos han tenido durante años que lidiar con el problema de determinar instantáneamente la velocidad puntual de fluidos sin alterarlo significativamente por la introducción de sensores [OFS00]. La información extraída de este campo de velocidades o flujo puede ser utilizada para mejorar el comportamiento de piezas

mecánicas inmersas en fluidos como puede ser el caso de la aerodinámica de un automóvil o el comportamiento de la hélice de un barco.

Por el momento la técnica menos intrusiva sería la adquisición de imágenes de partículas en el fluido (figura 23). Este método se conoce con el nombre de *Particle Image Velocimetry* (PIV). El fluido a estudiar se "siembra" con partículas colorantes o "humo" y mediante una iluminación adecuada al caso, se toman secuencias de imágenes y se procesan para determinar el campo de movimiento, que no es otra cosa en este caso que el flujo óptico.

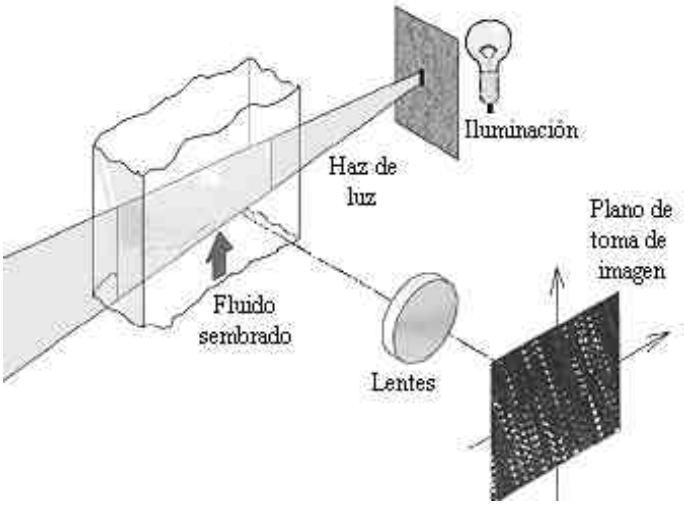


Figura 23. Determinación del campo de movimiento de un fluido por el método de Particle Image Velocimetry (PIV).

La empresa Optical Flow Systems, Ltd., formada de la Universidad de Edimburgo, es pionera en este campo. Diseña y comercializa sistemas para la medición de fluidos basados en adquisición de imágenes (figura 24). Sus sistemas, aunque responden a las necesidades de cada cliente en particular, están basados en una arquitectura centrada en un procesador de propósito general del tipo PC, y permiten un procesamiento relativamente rápido de unas 4 o 5 imágenes por segundo. Todo el procesamiento se realiza por medio de *software*. Los clientes de esta empresa dan una idea de los campos donde es necesaria una medición de velocidades de fluídos: Departamento de Agricultura de los Estados Unidos, BMW, Rolls-Royce, etc.

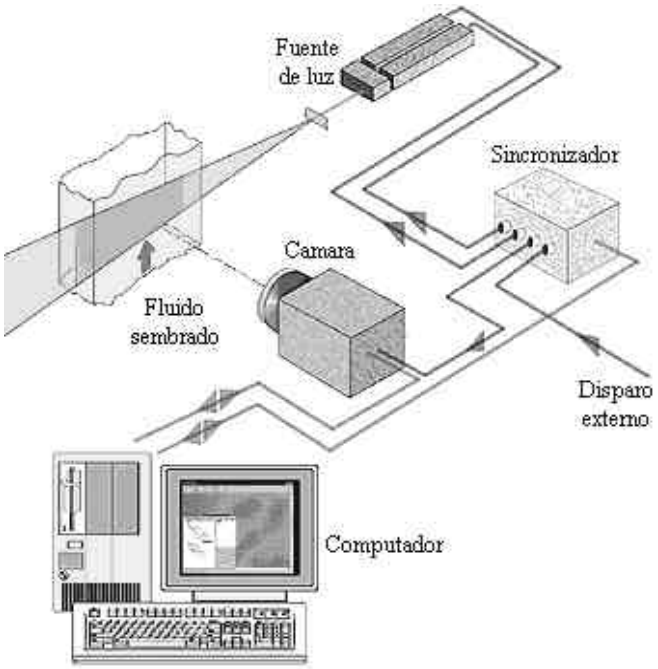


Figura 24. Sistema para *Particle Image Velocimetry* (PIV) de Optical Flow Systems Ltd.

2.6. Métodos para la determinación del flujo óptico

Se han propuesto una gran cantidad de algoritmos para la determinación del *flujo óptico* en imágenes de vídeo. Cada autor establece algunas mejoras o nuevas formas de resolver el problema que supone la extracción del *flujo óptico* de una serie de imágenes. Sin embargo, básicamente se utilizan dos formas de acometer el problema, con diversas variaciones sobre ellos:

La *segmentación de las imágenes* consiste en identificar una serie de características locales (objetos, bloques, segmentos, etc) en una imagen, tratar de obtener las correspondencias con la otra imagen, y por último determinar los movimientos. Estas técnicas adolecen del problema de que no generan un campo vectorial muy denso y de que su complejidad crece con el número de particularidades con el que cuenta la imagen [SCHW95].

El *método de los gradientes* espacio-temporales consiste en determinar los cambios (gradientes) espaciales y temporales del patrón de grises de la imagen y a partir de ellos obtener el flujo óptico. Este método tiene como base el hecho de que en los cambios de intensidad de la imagen está contenida la información del movimiento de los objetos. Tiene la ventaja de que genera un campo vectorial muy denso (un vector por *pixel*) que resulta muy útil en algunas aplicaciones. Sin embargo adolece del problema de que es muy sensible al ruido en las imágenes y que no puede determinar el flujo bajo ciertas circunstancias.

Para la estimación del movimiento en una primera aproximación, es necesario tener por lo menos un par de imágenes consecutivas, sin embargo, el uso de más imágenes aporta una

mayor información y, por ello, puede obtenerse una estimación más real del movimiento [SHAR90].

2.6.1. Método de la segmentación

A diferencia de los métodos de determinación de movimiento en las imágenes por gradientes espacio-temporales, los métodos de determinación por segmentación de imágenes son mucho más variados en su desarrollo. Sin embargo, podemos establecer tres métodos básicos: los algoritmos de segmentación en líneas, los algoritmos de segmentación en bloques y los algoritmos de segmentación en grupos o *clusters*. Dado que los objetivos de la presente Tesis están más encaminados a la determinación del *flujo óptico*, no se profundizará aquí sobre estos algoritmos, sólo se presentará una breve idea de la forma de llevarlos a cabo.

Los algoritmos de segmentación en líneas extraen el contorno de los objetos, modelan éstos como segmentos de líneas y determinan los parámetros de movimiento basándose en el desplazamiento de los segmentos entre las sucesivas imágenes. Ejemplos de éstos algoritmos se pueden encontrar en [GU96], [ARAP97] y [ZHAN95]. Estos algoritmos extraen en primer lugar los contornos de los objetos contenidos en la imagen. Para ello utilizan operadores por convolución u operadores morfológicos detectores de bordes. Luego determinan las coordenadas de cada uno de los segmentos de líneas que conforman los objetos en las imágenes, determinan la correspondencia entre los segmentos de las imágenes consecutivas y, por último, determinan el movimiento de aquellos segmentos cuya posición ha cambiado.

Los algoritmos de segmentación en bloques modelan las imágenes dividiéndolas en rectángulos (u otros polígonos) de tamaños variables que se adaptan a áreas de nivel de gris (o color) más o menos homogéneo. El movimiento se determina basándose en el desplazamiento de los centros de los rectángulos entre las sucesivas imágenes. Ejemplos de estos algoritmos se pueden encontrar en [BLAC96], [SCHW95] y [PANJ95]. Estos algoritmos, por lo general, dividen las imágenes en una serie de bloques “semilla”, y luego vuelven a dividir estos bloques en bloques más pequeños. El proceso continúa hasta que se cumplen ciertas restricciones en cuanto a la dispersión de niveles de gris (o color) dentro de los bloques. Luego se procede a buscar las correspondencias entre los bloques de dos imágenes consecutivas y, por último, se determina el movimiento de aquellos bloques que cambiaron de posición. Estos bloques suelen ser, por lo general, polígonos simples tales como rectángulos.

Los algoritmos de segmentación de grupos (*clusters*) que modelan las imágenes como áreas amorfas adaptadas a áreas de la imagen de características más o menos homogéneas en cuanto a color, nivel de gris o textura. El movimiento se determina basándose en el desplazamiento de los "centros de masa" de cada una de las áreas correspondientes entre dos imágenes consecutivas. Ejemplos de estos algoritmos se pueden encontrar en [KOTT94], [PARD94] y [UCHI94]. Estos algoritmos utilizan procesos de simplificación de imágenes agrupando áreas donde el color o la textura son similares. De esta manera las imágenes quedan divididas en áreas amorfas pero de color o textura uniformes. Luego se procede a determinar los “centros” de cada área, hallar las correspondencias entre las áreas de las imágenes consecutivas y, por último, a determinar el movimiento entre los centros de áreas correspondientes de dos imágenes consecutivas.

Todos los algoritmos de segmentación tienen la característica común de generar un campo vectorial de flujo óptico poco denso: un vector por línea, bloque o grupo. Es posible asignar el

mismo vector a todos los *pixels* dentro de un cuadro o área, pero es una generalización que puede no ser útil en todos los casos.

También los algoritmos de segmentación adolecen del problema de que son computacionalmente muy exigentes. Los procesos de buscar las correspondencias entre segmentos, bloques o áreas de una imagen a otra no son sencillos. Además, cuando una imagen está compuesta por múltiples objetos, las segmentaciones suelen ser numerosas y, por tanto, el cálculo extremadamente laborioso.

2.6.2. Método de los gradientes

Tal y como se mencionó anteriormente, los métodos para la determinación del flujo óptico por medio de gradientes se basan en la determinación de las variaciones espaciales y temporales del patrón de grises entre las imágenes de la secuencia [GONZ00] [HORN86b]. Por ello, el método requiere un mínimo de textura de los objetos, es decir que exista un gradiente espacial. También, el hecho de que el método mide sutiles cambios en el tiempo del patrón de grises, obliga a que los desplazamientos entre imágenes sean pequeños, o que guarden relación con la *suavidad* de la textura.

Matemáticamente, los cambios se valoran con el uso de los gradientes, es decir los diferenciales espaciales y temporales. De ahí que el método reciba el nombre de método de los gradientes o diferencial.

La información que se obtiene de éste método corresponde a la velocidad instantánea de cada *pixel* de la imagen. Sin embargo, tiene varias limitaciones que obligan a recurrir a ciertas restricciones con las cuales es posible hacer una determinación relativamente buena del flujo óptico en casos no extremos.

Dado que para la determinación del flujo óptico con este método es necesario calcular diferenciales, es necesario señalar que éstas tienen un "comportamiento brusco" cuando se encuentran con cambios bruscos en la el patrón de intensidades de la imagen. En el caso de secuencias de imágenes, este tipo de cambios se producen con frecuencia cuando las imágenes son afectadas por el ruido. Por ello es usual utilizar filtros para suavizar las imágenes. Sin embargo en la solución aquí desarrollada este efecto de suavizado se obtiene por el hecho de utilizar imágenes distintas en la secuencia, es decir, no se trabaja una y otra vez sobre las mismas imágenes [HORN81].

Existen diversas variedades del método de gradientes propuestas por múltiples autores entre los que se pueden citar a H. Nagel [NAGE92], P. Werkhoven y J. Koenderink [WERK90], F. Bartolini et al. [BART97], y M. Otte y H. Nagel [OTTE95]. En esta Tesis se ha optado por el método básico de B. Horn y B. Schunck [HORN81] con ligeras variaciones para ser adaptado a su realización en *hardware*. En las secciones siguientes se hará el análisis matemático de este método básico de los gradientes.

2.7. Determinación del flujo óptico por gradientes

2.7.1. Desarrollo teórico.

Una secuencia de imágenes puede ser representada como una función de intensidad luminosa dependiente de la posición y tiempo:

(2.1) $$E(x, y, t)$$

donde *x* e *y* representan las coordenadas espaciales de un punto de la imagen y *t* representa el tiempo.

Si se cumplen que la iluminación de los objetos en las imágenes se mantiene constante en el tiempo, que no existen objetos que se oculten unos a otros y que no aparecen nuevos objetos en la imagen, entonces, cada punto de la imagen se mueve a consecuencia del movimiento de los objetos, y ello puede expresarse matemáticamente como:

(2.2) $$E(x, y, t) = E(x + dx, y + dy, t + dt)$$

Si se expande el segundo termino de la ecuación 2.2 por medio de la serie de Taylor[1] centrada en el punto (*x,y,z*):

(2.3) $$E(x, y, t) = E(x, y, t) + \frac{\partial E(x, y, t)}{\partial x}(x + d_x - x) + \frac{\partial E(x, y, t)}{\partial y}(y + d_y - y) + \frac{\partial E(x, y, t)}{\partial t}(t + d_t - t) + ...$$

o lo que es lo mismo:

(2.4) $$E(x, y, t) = E(x, y, t) + \frac{\partial E(x, y, t)}{\partial x}(d_x) + \frac{\partial E(x, y, t)}{\partial y}(d_y) + \frac{\partial E(x, y, t)}{\partial t}(d_t) + e$$

donde ε representa los términos de orden superior.

Asumiendo que los términos de orden superior son despreciables y restando *E(x,y,z)* en ambos lados de la ecuación, se obtiene:

(2.5) $$0 = \frac{\partial E(x, y, t)}{\partial x}(d_x) + \frac{\partial E(x, y, t)}{\partial y}(d_y) + \frac{\partial E(x, y, t)}{\partial t}(d_t)$$

Dividiendo la ecuación por δ_t y simplificando ligeramente la notación utilizada, se obtiene:

[1] Teorema de Taylor: sea f una función con derivadas de todos los órdenes en algún intervalo (a-r,a+r). La función f en dicho intervalo puede ser representada por la serie de potencias:

$$f(x) = f(a) + f'(a)(x - a) + \frac{f''(a)}{2!}(x - a)^2 + \frac{f'''(a)}{3!}(x - a)^3 + ...$$

(2.6) $$\frac{\partial E}{\partial x}\frac{d_x}{d_t}+\frac{\partial E}{\partial y}\frac{d_y}{d_t}+\frac{\partial E}{\partial t}=0$$

En el límite en que δ_t tiende a cero, se obtiene:

(2.7) $$\frac{\partial E}{\partial x}\frac{dx}{dt}+\frac{\partial E}{\partial y}\frac{dy}{dt}+\frac{\partial E}{\partial t}=0$$

donde:

$\frac{dx}{dt}=u$ $\frac{dy}{dt}=v$ son las velocidades horizontal y vertical de los *pixels*, y

$\frac{\partial E}{\partial x}=E_x$ $\frac{\partial E}{\partial y}=E_y$ $\frac{\partial E}{\partial t}=E_t$ son los gradientes espacio-temporales de la imagen

Finalmente, se presenta la ecuación del flujo óptico de la siguiente manera:

(2.8) $$E_x u+E_y v+E_t=0$$

2.7.2. El problema de apertura

La ecuación de flujo óptico tiene dos incógnitas *(u,v)*, ya que los demás parámetros *(E_x, E_y, E_t)* pueden ser determinados a partir de la secuencia de imágenes. Dado que sólo se tiene una ecuación con dos incógnitas los valores de la velocidad no pueden ser determinados unívocamente. Si se cambia la forma de la ecuación (2.8) a una presentación vectorial, es posible interpretar el resultado:

(2.9) $$\vec{\nabla}E\cdot\vec{V}=-E_t$$

En esta forma se observa que la ecuación es un producto escalar entre el vector velocidad y el gradiente de la intensidad luminosa, por lo que se concluye que sólo puede ser determinado el vector velocidad en la dirección del gradiente espacial de la luminosidad. Este problema se conoce con el nombre de "problema de apertura". No es un problema exclusivo de ésta ecuación sino que un problema inherente a la visión y que se da inclusive en la visión humana.

Para explicar el problema de apertura de una manera menos matemática, vamos a suponer un objeto, un cuadro negro, que se desplaza en forma diagonal delante de una ventana de tamaño menor al objeto (figura 25). Si hay partes horizontales y verticales diferenciadas (gradientes espaciales verticales y horizontales) del objeto, es posible determinar sus vectores de movimiento. Si sólo se observa una parte vertical (u horizontal) del objeto, sólo podrá observarse el movimiento perpendicular a dicha parte, el movimiento en el otro sentido no podrá ser determinado. Por último, si de la ventana no se observa ninguna parte diferenciada no es posible determinar ningún movimiento.

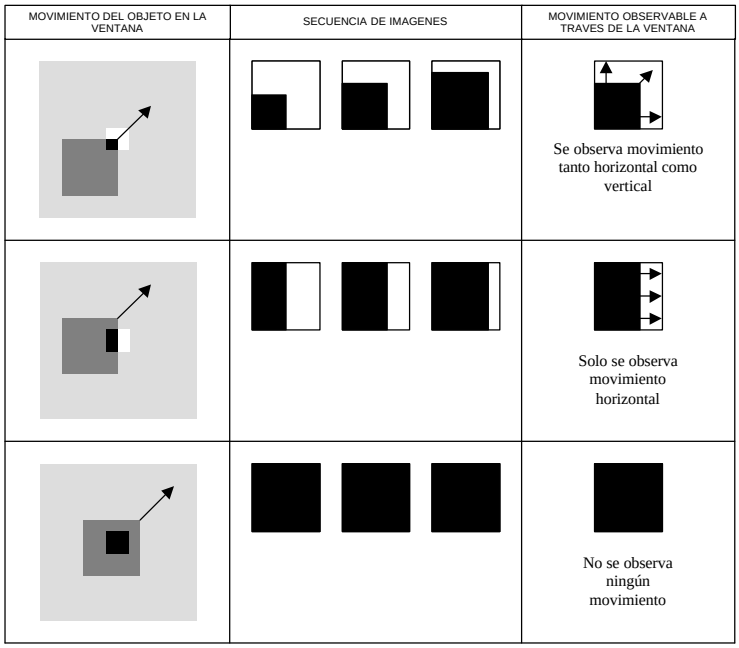


Figura 25. Problema de apertura.

En el caso de imágenes digitales, el tamaño de la ventana viene siendo determinado por el tamaño y el número de los *pixels* utilizados para calcular los gradientes. Para estimar el movimiento en esos casos en que el problema de apertura se hace significativo, es necesario recurrir a alguna manera de "aumentar la apertura".

A diferencia de la ecuación de flujo óptico que se obtiene por una lógica matemática bastante exacta, la ecuación adicional requerida para solucionar el problema de apertura es obtenida generalmente de considerar ciertas aproximaciones válidas en mayor o menor grado.

En particular, se suele suponer que los puntos cercanos en una imagen deben tener movimientos similares. Esta condición se conoce con el nombre de "suavidad en el flujo óptico", y no está lejos de la realidad. De hecho, si cada punto de la imagen tuviera un movimiento independiente y aleatorio, no sería posible determinar su movimiento [HORN81].

2.7.3. Restricciones de suavidad en el flujo óptico.

Tal y como se ha mencionado, es necesario agregar una ecuación adicional para poder resolver satisfactoriamente la ecuación de flujo óptico, y tal ecuación es la que pone restricciones en los cambios de velocidad entre puntos cercanos de la imagen. Matemáticamente el cambio de velocidades entre puntos cercanos se representa por las derivadas parciales:

$$u_x = \frac{\partial u}{\partial x} \qquad u_y = \frac{\partial u}{\partial y} \qquad v_x = \frac{\partial v}{\partial x} \qquad v_y = \frac{\partial v}{\partial y}$$

El cambio del módulo de la velocidad de un punto a otro se puede expresar por medio de una ecuación cuadrática:

(2.10) $$u_x^2 + u_y^2 + v_x^2 + v_x^2 = e_b{}^2$$

donde $\varepsilon_b{}^2$ representa el cambio cuadrático del modulo de la velocidad, que debe ser aproximadamente cero para cumplir con la restricción de que los cambios en la velocidad deben ser suaves.

Mediante sistemas utilizados en el cálculo de errores, se pueden combinar la ecuación de restricción de suavidad del flujo óptico (2.10) y la ecuación de flujo óptico (2.8), pero para ello es necesario en primer lugar modificar ligeramente la ecuación de flujo óptico. En la ecuación 2.5 se han despreciado los términos de orden superior ε, que ahora se vuelven a retomar de manera que la ecuación de flujo óptico queda convertida a:

(2.11) $$E_x u + E_y v + E_t = e_a$$

Ahora es posible combinar las ecuaciones en un sistema de cálculo de errores, agregando una ecuación adicional en la que se establece el error total como la suma de los demás errores y agregando un factor de ajuste λ^2 para compensar el peso de cada error.

(2.12) $$e_t^2 = e_a^2 + l^2 e_b^2$$

Finalmente, el sistema de ecuaciones queda en la forma siguiente:

(2.13) $$\begin{cases} E_x u + E_y v + E_t = e_a \\ u_x^2 + u_y^2 + v_x^2 + v_y^2 = e_b{}^2 \\ e_a^2 + l^2 e_b^2 = e_t^2 \end{cases}$$

En este sistema de ecuaciones es necesario minimizar el error total de la imagen, es decir, tratar de hacerlo cero. El error de toda la imagen será:

(2.14) $$e_i^2 = \iint_I e_t^2 dx\, dy$$

Combinando el sistema de ecuaciones 2.13 en la ecuación 2.14 se obtiene la siguiente ecuación integral:

(2.15) $$e_i^2 = \iint_I (E_x u + E_y v + E_t)^2 + l^2 (u_x^2 + u_y^2 + v_x^2 + v_y^2)\, dx\, dy$$

La tarea de minimizar esta ecuación de error es un problema de cálculo de variaciones que se resuelve por medio de las ecuaciones de Euler. En particular la ecuación 2.15 es una integral de la forma:

(2.16) $$\iint_I F(u, v, u_x^2, u_y^2, v_x^2, v_y^2)\, dx\, dy$$

donde $$F = (E_x u + E_y v + E_t)^2 + l^2 (u_x^2 + u_y^2 + v_x^2 + v_y^2)$$

y cuyas ecuaciones de Euler correspondientes son:

$$\begin{cases} F_u - \frac{\partial}{\partial x} F_{u_x} - \frac{\partial}{\partial y} F_{u_y} = 0 \\ F_v - \frac{\partial}{\partial x} F_{v_x} - \frac{\partial}{\partial y} F_{v_y} = 0 \end{cases} \tag{2.17}$$

Las soluciones a estas ecuaciones diferenciales de Euler representan la solución a la minimización de la ecuación 2.15. En este caso en particular, las soluciones de Euler son:

$$\begin{cases} \nabla^2 u = l^2 (E_x u + E_y v + E_t) E_x \\ \nabla^2 v = l^2 (E_x u + E_y v + E_t) E_y \end{cases} \tag{2.18}$$

donde ∇^2 representa el operador Laplaciano, definido como:

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \tag{2.19}$$

2.7.4. Estimando el Laplaciano

Para determinar el Laplaciano de un array bidimensional como el de una imagen digitalizada se utiliza la convolución con determinadas máscaras (sección 2.3). Se han propuesto varias de estas máscaras [GONZ96] [ARAN89] [HORN86b], pero una buena aproximación se obtiene con la propuesta por Horn y Schunck [HORN81] y que se representa en la figura 26.

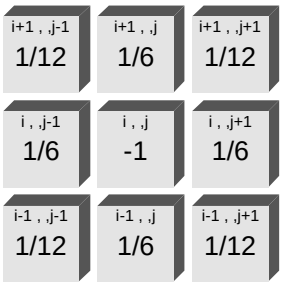


Figura 26. Máscara para la estimación del Laplaciano.

A partir de esta máscara se puede obtener una conveniente aproximación al Laplaciano expresada matemáticamente de la siguiente manera:

$$\begin{cases} \nabla^2 u = k(\bar{u} - u) \\ \nabla^2 v = k(\bar{v} - v) \end{cases} \tag{2.20}$$

donde *k* tiene el valor de 3. Las variables $\bar{u}$ y $\bar{v}$ que se denominarán *coeficientes de promediado*, se definen en forma discreta como sigue:

(2.21)
$$\begin{cases}\bar{u}_{i,j} = \frac{1}{6}\left(u_{i-1,j} + u_{i,j+1} + u_{i+1,j} + u_{i,j-1}\right) + \frac{1}{12}\left(u_{i-1,j-1} + u_{i-1,j+1} + u_{i+1,j+1} + u_{i+1,j-1}\right) \\ \bar{v}_{i,j} = \frac{1}{6}\left(v_{i-1,j} + v_{i,j+1} + v_{i+1,j} + v_{i,j-1}\right) + \frac{1}{12}\left(v_{i-1,j-1} + v_{i-1,j+1} + v_{i+1,j+1} + v_{i+1,j-1}\right)\end{cases}$$

y su representación en forma de máscara sería la mostrada en la figura 27.

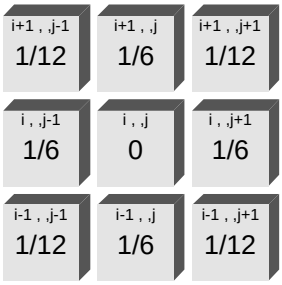


Figura 27. Máscara para la estimación de los *coeficientes de promediado* ($\bar{u}$ y $\bar{v}$).

2.7.5. Ecuaciones iterativas del flujo óptico

Para obtener unas ecuaciones iterativas del flujo óptico se combinan las ecuaciones 2.20 y 2.18. Para facilitar las operaciones, se vuelven a escribir las ecuaciones con un ligero cambio de notación:

(2.22)
$$b^2 \equiv \frac{1}{l^2}$$

(2.23 (2.18))
$$\begin{cases} b^2\nabla^2 u = (E_x u + E_y v + E_t)E_x \\ b^2\nabla^2 v = (E_x u + E_y v + E_t)E_y \end{cases}$$

(2.24 (2.20))
$$\begin{cases} \nabla^2 u = k(\bar{u} - u) \\ \nabla^2 v = k(\bar{v} - v) \end{cases}$$

Uniendo ambos grupos de ecuaciones, y definiendo $a^2 \equiv kb^2$ se obtiene:

(2.25)
$$\begin{cases} a^2(\bar{u} - u) = (E_x u + E_y v + E_t)E_x \\ a^2(\bar{v} - v) = (E_x u + E_y v + E_t)E_y \end{cases}$$

Sustituyendo adecuadamente las ecuaciones 2.25 entre sí, se obtienen finalmente las ecuaciones iterativas del flujo óptico:

(2.26)
$$\begin{cases} u = \bar{u} - E_x \dfrac{E_x\bar{u} + E_y\bar{v} + E_t}{a^2 + E_x^2 + E_y^2} \\ v = \bar{v} - E_y \dfrac{E_x\bar{u} + E_y\bar{v} + E_t}{a^2 + E_x^2 + E_y^2} \end{cases}$$

El conjunto de ecuaciones 2.26 son la pieza fundamental del desarrollo de esta Tesis, y cobrarán especial importancia en capítulos posteriores.

3

Capítulo 3

Materiales y métodos

La herramienta que alarga la mano del hombre
es también un instrumento de visión.
Revela la estructura de las cosas
hace posible el volver a unirlas en
combinaciones nuevas llenas de imaginación

Jacob Bronowski
El ascenso del hombre, 1973

3.1. Señalización entre módulos segmentados

Para poder utilizar una arquitectura segmentada al máximo de su potencial debe definirse una forma de señalización coherente con la aplicación, en nuestro caso el procesamiento de imágenes. Desde este punto de vista, el concepto involucrado en la transmisión de imágenes de televisión analógica, estudiado en la sección 2.2.2, es más útil que el utilizado en el formato de transmisión digital. Es decir, se separan las señales de sincronismo vertical horizontal y la imagen en sí misma.

Siguiendo este concepto básico se propone un formato digital de señalización común para los módulos de procesamiento de imágenes.

La conexión se realiza por medio de 4 señales de control y el bus de datos. El número de bits fijados para el dato determina el número de niveles de brillo posibles. En vídeo digital varios estudios han determinado que 8 bits son suficientes para una calidad muy aceptable, aunque algunos fabricantes especifican 10 bits para solucionar problemas debidos al procesamiento [QUAN98] [VIDE98]. Dado a que el procesamiento de 10 bits resultaría mucho más exigente en recursos, se opta por seleccionar un bus de datos de 8 bits.

Por otro lado, dado que la información de flujo óptico se concentra básicamente en los niveles de brillo de la imagen, para el sistema de determinación de flujo óptico sólo se requiere la información de nivel de gris o brillo de la imagen, por lo que se omite la información de color. De esta manera el interfaz completo que conecta este módulo de arquitectura segmentada se realiza por medio de las siguientes señales:

- **Reloj de sincronismo de los datos (CLK).** Indica con el flanco de subida que el dato puede ser tomado.

- **Dato válido (DV).** Indica que los datos son válidos y que pueden ser tomados en el próximo flanco de subida del reloj (CLK).

- **Sincronismo horizontal (HS).** Señala con el flanco de subida el comienzo de una nueva línea horizontal en la imagen.

- **Sincronismo vertical (VS).** Señala con el flanco de subida el comienzo de un nuevo cuadro de la imagen.

- **Bus de datos (D0-D7).** Datos sobre el nivel de gris de cada punto de la imagen. Son 8 bits, por lo que se representan 256 niveles de gris. El nivel más brillante (blanco) es representado por el dato FFh y el nivel más oscuro (negro) por el dato 00h.

En la figura 28 se muestran las señales de la interfaz de vídeo en su perspectiva temporal. La figura es una representación simplificada donde se presentan sólo 8 pulsos de sincronismo horizontal entre dos pulsos de sincronismo vertical, lo que indica que se trata de una imagen de sólo 8 líneas de resolución.

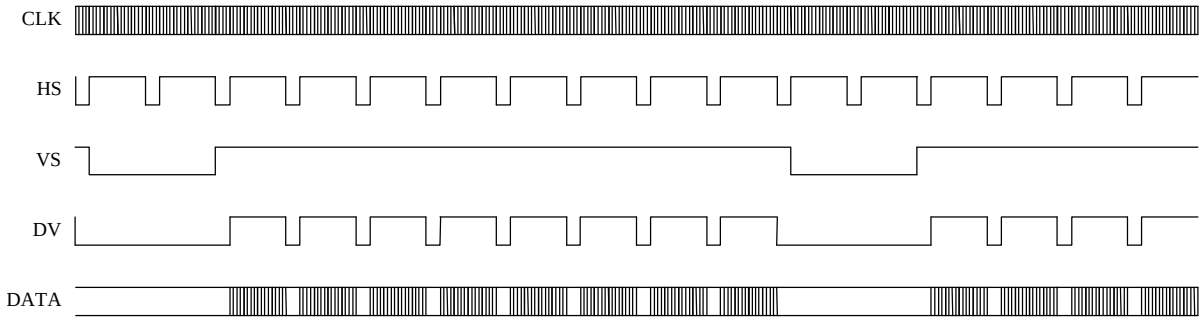


Figura 28. Señales digitales para imágenes de vídeo.

Hasta este momento, ninguna de las señales está restringida a alguna resolución específica de las imágenes. De hecho, el límite de resolución estaría fijado por las características del sistema que procesará las señales y por las características de la fuente de las imágenes. En el caso aquí expuesto, las imágenes serán obtenidas de una cámara blanco y negro convencional. Estas cámaras tienen la particularidad de generar campos *entrelazados*, es decir, las imágenes contienen alternativamente información de dos posiciones ligeramente desplazadas en sentido vertical. Esto es particularmente problemático para la determinación del flujo óptico, ya que se busca obtener el desplazamiento de *pixels* en imágenes y la determinación de este desplazamiento con imágenes desplazadas por sí mismas representa un problema de difícil solución.

Pero el procesamiento de la imagen puede realizarse utilizando campos alternativos. En la tabla 2 se indicó la resolución de las imágenes de televisión digitales, específicamente para el sistema europeo de televisión se especifica una resolución de 720 x 576 *pixels* a una velocidad de 25 cuadros por segundo. Ahora bien, estos cuadros son dos imágenes entrelazadas, es decir, cada imagen tiene una resolución máxima de 720 x 288 *pixels* y se presentan 50 imágenes por segundo.

Con el objetivo de disponer de "*pixels* cuadrados" es conveniente que la resolución tanto vertical como horizontal sean las mismas: 288 x 288 *pixels*. Considerando además que los bordes de la imagen no contienen información útil (se utilizan para generar bandas negras mientras se establecen los sincronismos) y que por razones digitales se prefieren resoluciones que son potencias de dos, la resolución más adecuada parece ser de 256 x 256 *pixels*.

Para obtener la resolución mencionada, es necesario un reloj con una frecuencia de 5 MHz (200 ns de período). Los datos son tomados en el flanco de subida de la señal de reloj (CLK) y cuando la señal de dato válido (DV) está activa. Las relaciones temporales entre el reloj y las demás señales se presentan en la figura 29.

Existen también ciertas relaciones que se deben cumplir entre las señales de sincronismo horizontal, datos válidos y el sincronismo horizontal. Estas relaciones se definen en la figura 30.

Es necesario establecer que, si bien, para el sistema que se desarrolla en esta Tesis, se requiere una resolución de 256 x 256 y esto fija la frecuencia de reloj en 5 MHz, el interfaz aquí propuesto sigue siendo válido para cualquier otra resolución o frecuencia de reloj. De hecho, en las simulaciones del sistema se trabajará con resoluciones mucho menores para agilizar las pruebas.

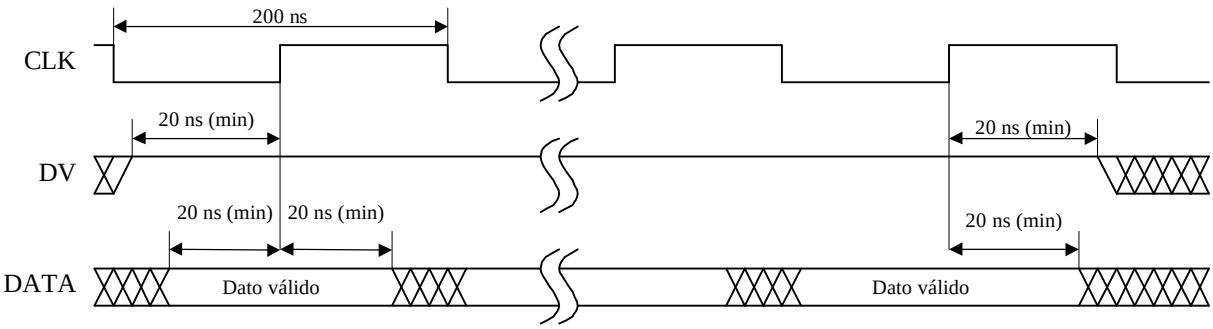


Figura 29. Relación entre señales digitales CLK, DV y DATA, para imágenes de vídeo.

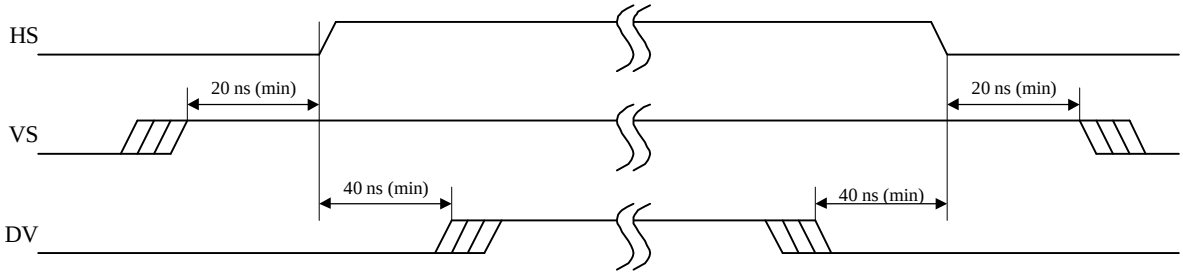


Figura 30. Relación entre señales digitales HS, VS y DV para imágenes de vídeo.

3.2. Herramientas diseñadas para esta Tesis

Para poder realizar los diferentes análisis y pruebas del sistema para la determinación del flujo óptico que se pretende desarrollar en esta Tesis se hizo necesario desarrollar ciertas herramientas. Las principales son las siguientes:

- Banco de pruebas para procesamiento digital de imágenes en VHDL.
- Analizador de algoritmos para determinación del flujo óptico

3.3. Banco de prueba en VHDL

Durante la realización de la presente Tesis hubo la necesidad de desarrollar una metodología que permitiera probar el sistema para la extracción de flujo óptico con el uso de VHDL. Previendo el uso de esta herramienta para cualquier otra aplicación de procesamiento de secuencias de imágenes que fuera desarrollada utilizando el mismo sistema de segmentación aquí referido, se creó una herramienta muy genérica basada en dos módulos VHDL

[ZULO99a] [BIDA99]. Uno de los módulos VHDL simula una cámara digital de vídeo y el otro módulo simula un monitor digital de televisión (figura 31). Esto que permite la utilización de un único lenguaje a lo largo de todo el proceso de diseño.

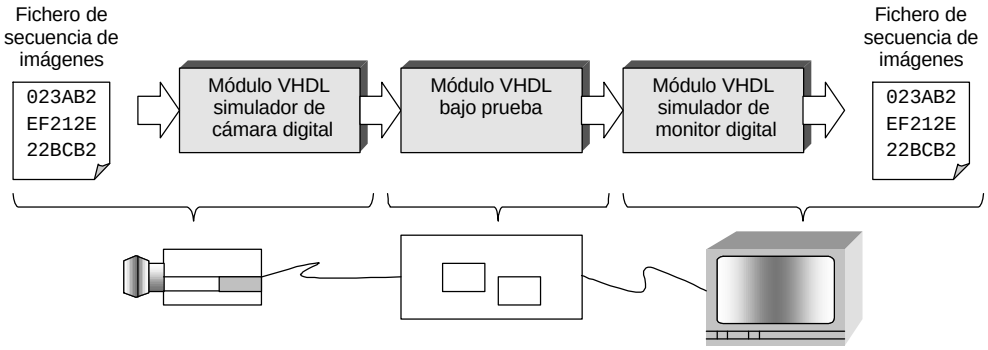


Figura 31. Banco de prueba VHDL.

Las herramientas de diseño en lenguaje VHDL para FPGAs y PLDs incorporan varias utilidades para la prueba de los sistemas a desarrollar, basados en el uso de ficheros con una serie de estímulos o vectores de prueba. Sin embargo, el procesamiento de imágenes requiere el manejo de gran cantidad de datos que no es posible generar y analizar utilizando los tradicionales vectores de prueba. Con el uso de este banco de prueba se pretende eliminar la necesidad de estos vectores de prueba y sustituirlos por un bloque VHDL que lea ficheros conteniendo secuencias de imágenes y las utilice para generar los estímulos necesarios para el sistema bajo prueba.

De la misma manera, los sistemas de procesamiento de imágenes generan una gran cantidad de información cuyo estudio a través de la información proporcionada por las herramientas de diseño en VHDL, basadas en diagramas de tiempo, resulta de muy poca utilidad. Nuevamente, el banco de pruebas aquí propuesto permite tomar las señales generadas por el módulo bajo prueba y producir ficheros de imágenes que son una forma más útil de estudiar la información referente a procesamiento de imágenes.

La simulación de funcionamiento en un banco de pruebas como el aquí propuesto requiere el uso de un simulador de VHDL que permita una descripción comportamental del sistema. Los programas para el desarrollo de FPGAs en lenguaje VHDL no suelen permitir el uso de descripciones comportamentales: una cámara o un monitor no pueden ser integrados en una FPGA. Por ello, para la simulación de un modelo utilizando el banco de pruebas de vídeo se utilizará el programa de simulación de VHDL *ModelSim* [MODE97]. Este programa de simulación es capaz de interpretar todos los comandos definidos por el estándar VHDL [IEEE94].

Para crear el banco de prueba fue necesario crear y definir un formato de ficheros que pudiera ser útil para su procesamiento por medio de VHDL y también un sistema con el cual pudieran ser analizados estos ficheros.

En esta Tesis el sistema a desarrollar tiene como salida un campo vectorial bidimensional, por ello genera dos salidas en formato de imágenes digitales, que representan las componentes vertical y horizontal del flujo óptico (esto se estudiará con más detalle en la sección 4.1.2).

Dado que las salidas tienen el formato de imágenes digitales, pueden ser conectadas al banco de pruebas utilizando dos módulos de monitor tal y como se muestra en la figura 32.

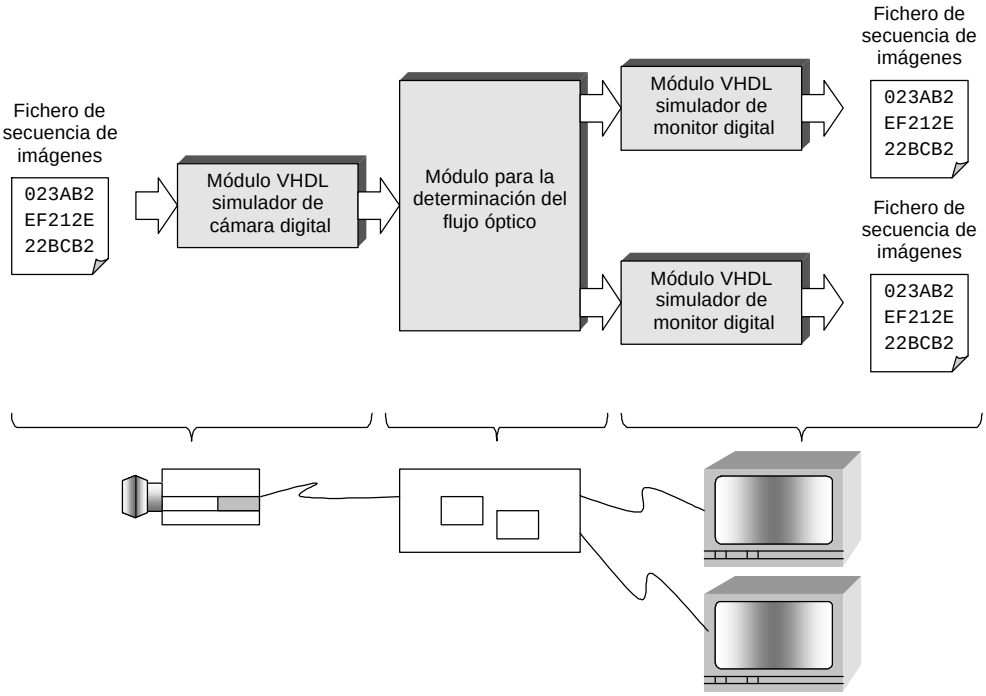


Figura 32. Banco de prueba VHDL usado en la determinación del flujo óptico.

3.3.1. El formato *hex*

El lenguaje de descripción de *hardware* no permite procesar ficheros de imágenes *bitmap* o similares, ya que fue diseñado para lecturas y escrituras de ficheros basados en caracteres ASCII. Por ello se hace necesario definir un formato especial de ficheros de imágenes donde sólo se utilicen caracteres ASCII.

La manera más obvia y sencilla de representar información binaria mediante caracteres ASCII es utilizando un formato de tipo *hex* [ZULO99b]. Los caracteres *hex* (0-9 A-F) son rápida y fácilmente convertidos a binario por los simuladores VHDL, a pesar de que el fichero en este formato requiere dos veces más bytes que un fichero *bitmap* para codificar la misma imagen.

Por el momento, las necesidades de simulación sólo han ido encaminadas al uso de imágenes en niveles de gris. Por lo que el formato aquí descrito sólo contempla este tipo de imágenes. Sin embargo, para proyectos a futuro ya se está trabajando en un formato similar con información de color.

En el formato *hex* aquí propuesto dos caracteres *hex* representan el nivel de gris de cada *pixel* de la imagen. Esto significa que es posible representar una escala de 256 niveles de gris por *pixel*. El nivel más blanco será representado por los caracteres *FF* y el más oscuro por los caracteres *00*. Adicionalmente es necesario codificar en el mismo fichero los sincronismos de vídeo presentes en una secuencia de imágenes.

El sincronismo horizontal, o cambio de línea de la imagen, será codificado por los caracteres ASCII *Carriage Return* ($^{C}_{R}$) y *Line Feed* ($^{L}_{F}$). Por otro lado, el sincronismo vertical, o cambio de cuadro de imagen, será codificado con un asterisco (*).

La codificación de una secuencia de imágenes ejemplo puede ser observada en la figura 33. Para simplificar el ejemplo se ha seleccionado una secuencia de imágenes de 16 x 16 *pixels*. Tal y como se mencionó, el nivel de gris de cada *pixel* se codifica como un par de caracteres. Obsérvese que este fichero contiene sólo la información requerida para la generación de las señales y no contiene información alguna sobre otras características tales como dimensiones de la imagen.

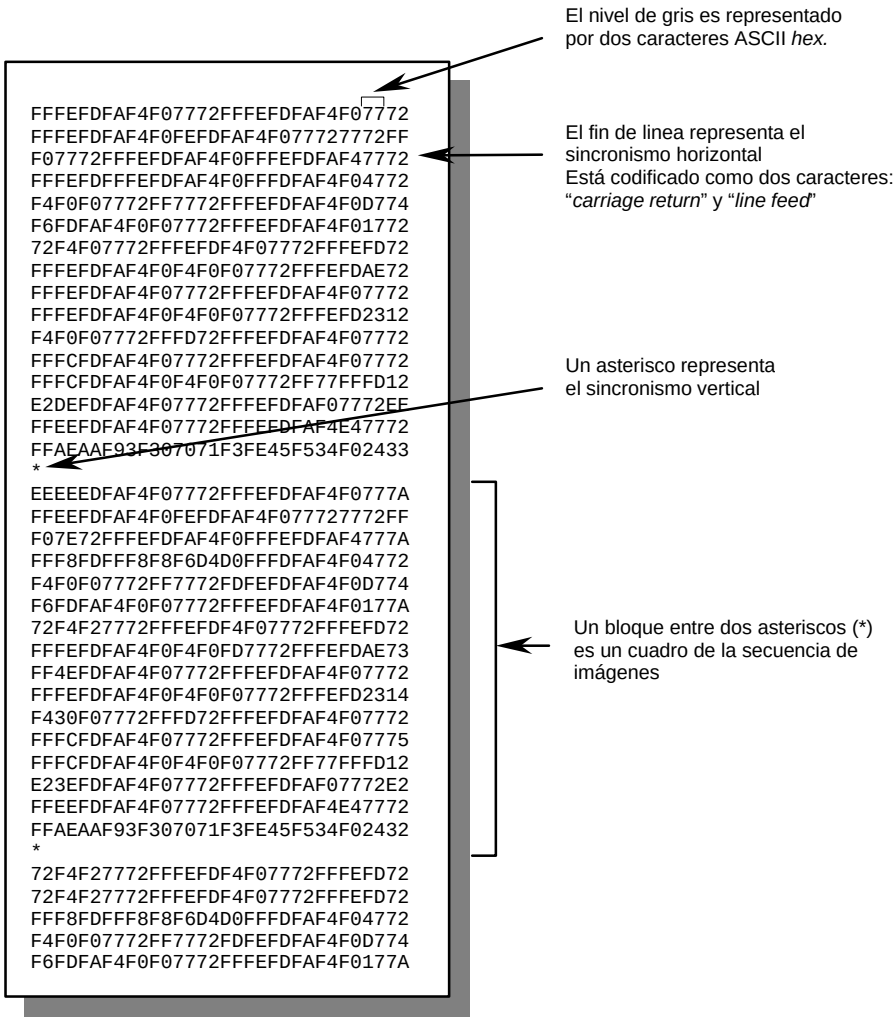


Figura 33. Ejemplo de un fichero *hex* conteniendo una secuencia de imágenes.

Para el sistema de determinación de flujo óptico, se generarán dos ficheros *hex*: uno para las magnitudes horizontales y otro para las verticales. Sin embargo, la magnitud es un número con signo y esto no puede ser representado fácilmente en el formato propuesto. Por ello se adopta una transformación que desplaza el nivel 0 a 128. Esto es, la magnitud cero será representada con el número hexadecimal 80, El mayor vector hacia la derecha o hacia arriba es representado con el número hexadecimal FF y el mayor vector hacia la izquierda o hacia abajo es representado con el número hexadecimal 01.

3.3.2. Convirtiendo y visualizando ficheros de imagen *hex*

Para la creación y visualización de los ficheros *hex* de imágenes se ha diseñado un programa específico: *Visor*. En su modo de creación de ficheros *hex*, el programa el programa *Visor* toma imágenes en los formatos *bitmap* (.bmp) y *Standard Interchange Format* (.sif) y los convierte al formato *hex*. En modo de visualización de ficheros de formato *hex* permite visualizar imágenes o campos vectoriales en el caso de información referente al flujo óptico.

Dado que el banco de pruebas se ha diseñado pensando en secuencias de imágenes, el programa *Visor* permite seleccionar múltiples archivos de imagen para poder formar un fichero en formato *hex* con secuencias de imágenes utilizables por la simulación en VHDL. También es posible tomar una sola imagen para salvarla como una serie de imágenes fijas en el formato *hex*.

El formato *Standard Interchange Format* está diseñado para almacenamiento de secuencias de imágenes de vídeo de 240 líneas de 352 puntos para NTSC (60 campos por segundo) y de 288 líneas de 352 puntos para PAL y SECAM (50 campos por segundo). Cada cuadro de la imagen es almacenado en un fichero independiente. En este formato, se incluye información tanto de nivel de gris como de color, pero tal y como se mencionó antes, sólo se hará uso de la información de nivel de gris.

En el formato *bitmap*, las imágenes deben ser en niveles de gris, el programa *Visor* no es capaz de tomar imágenes *bitmap* de color y convertirlas a nivel de gris.

Para la integración de todas las herramientas en una misma plataforma de trabajo, el programa *Visor* fue diseñado para ser utilizado en sistemas *Windows* utilizando el lenguaje de programación *Microsoft Visual BASIC*.

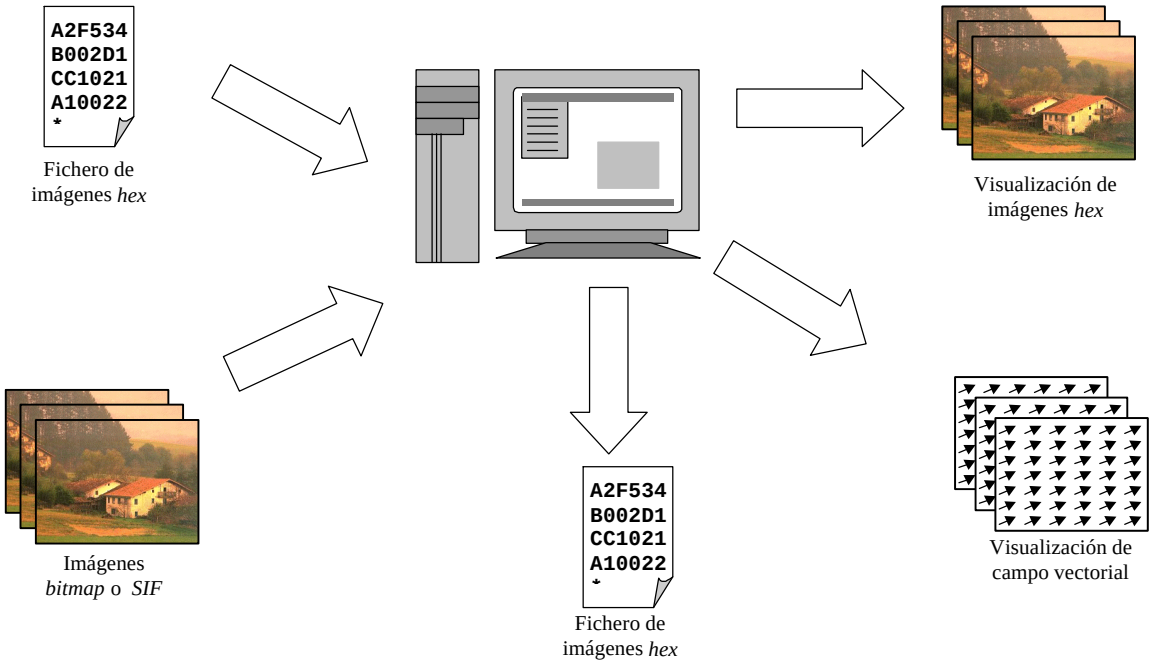


Figura 34. Programa *Visor:* conversión entre formatos y visualización.

3.3.3. Módulos VHDL del banco de pruebas

Fundamentalmente, el banco de pruebas VHDL consta de dos módulos: el módulo *camera* y el módulo *monitor*.

El módulo *camera* simula una cámara digital en niveles de gris. Las imágenes de entrada están codificadas en formato *hex*. Este módulo genera las señales de vídeo digital requeridas para el procesamiento, tal y como fueron descritas con anterioridad: datos, sincronismo horizontal, sincronismo vertical, dato válido y reloj de muestreo de datos.

En el apéndice C1 se muestra el código VHDL del módulo *camera*. Los datos de entrada, es decir, la secuencia de imágenes, se encuentra almacenada en un fichero denominado "inputvideo.hex". Esta denominación puede ser cambiada pasando el nombre del fichero como un parámetro genérico. De igual manera utilizando una serie de parámetros genéricos pueden ser cambiados otros parámetros de funcionamiento del módulo, tales como el período de reloj y las longitudes temporales de los sincronismos.

El módulo *monitor* simula un monitor digital. Las imágenes se reciben en el módulo *monitor* en el mismo formato de vídeo digital generado por el módulo *camera*, y es convertida y almacenada en un fichero en formato *hex*.

El código VHDL del módulo *monitor* se muestra en el apéndice C2 . Está compuesto por tres procesos que se encargan de escribir los códigos del nivel de gris, los códigos de sincronismo vertical y los códigos de sincronismo vertical, respectivamente.

3.4. Analizador de algoritmos

Para analizar los algoritmos a utilizar en el sistema para la determinación del flujo óptico se desarrolló una herramienta *software* con la que fuera posible describir los algoritmos matemáticos y visualizar su resultado de una manera fácil. Esta herramienta fue denominada "analizador de algoritmos" y fue realizada utilizando dos herramientas de desarrollo: el *MATLAB* y el Visual C.

Por una parte, el cálculo de la estimación de movimiento es complejo: requiere la manipulación de gran cantidad de datos y cálculos matriciales extensos. Estas tareas son propicias para llevarse a cabo con un lenguaje de cálculo matricial como es el *MATLAB*. Este lenguaje permite desarrollar los algoritmos en un tiempo menor al que se requeriría utilizando lenguajes del tipo C o Fortran.

Por otro lado, el programa además de rápido y exacto, debe ser funcional, sencillo e intuitivo a la hora de utilizarlo. En este caso el uso de *MATLAB* supondría en cierto modo una desventaja ya que implicaría que el usuario estuviera familiarizado con ese entorno de trabajo. En este sentido la solución más adecuada es desarrollar una aplicación en entorno Windows, de apariencia y modo de funcionamiento básico similar a otras muchas aplicaciones, y a las que los posibles usuarios ya están adaptados.

Con estas premisas en consideración, se desarrolló el programa analizador de algoritmos, con una estructura donde se combinan las ventajas ofrecidas por ambos lenguajes de programación, es decir, el *MATLAB* y el C++. En la figura 35 se muestra la estructura del programa analizador de algoritmos, donde se observan 3 bloques básicos: el *MATLAB* que procesará los algoritmos, la interfaz gráfica de usuario (GUI: *Graphical User Interface*) y el API (*Application Program Interface*) que se encarga de la interacción entre el *MATLAB* y la interfaz gráfica de usuario.

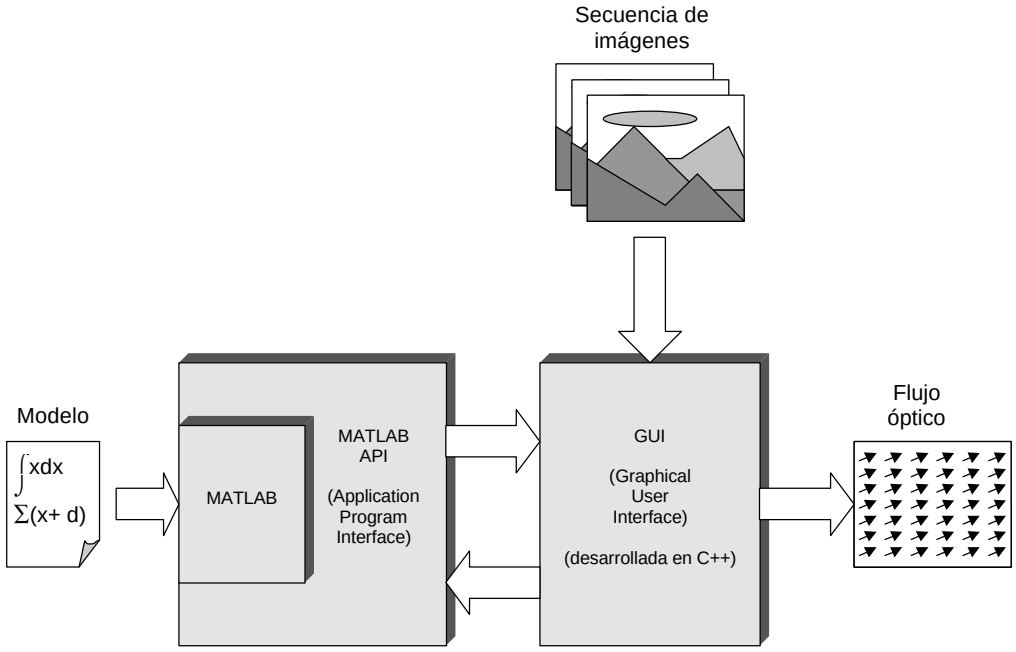


Figura 35. Programa analizador de algoritmos.

Tal y como se mencionó, el *MATLAB* se encarga de la interpretación y procesamiento de los algoritmos. Estos son introducidos en el lenguaje propio de *MATLAB*. En la figura 36 se observan algunas de las rutinas de cálculo de flujo óptico utilizadas durante los ensayos. Estas rutinas se refieren al cálculo de las ecuaciones iterativas del flujo óptico de Horn y Schunck. En la figura 37 se representa en forma esquemática la mencionada rutina de prueba. Si es necesario considerar varias imágenes de una secuencia, la imagen actual es considerada como imagen previa en la próxima iteración, y la nueva imagen será recibida por el *MATLAB* a través de la interfaz API.

```
% Computing R and P factors
D = Ex.^2 + Ey.^2 + lambda^2;


% Computing P
P = Ex.*FlujoX + Ey.*FlujoY +Et;


% Computing Rx
R=P./D;


% Calculate the optical flow
FlujoX = FlujoX - Ex.* R;
FlujoY = FlujoY - Ey.* R;
```

```
% Calculating x,y and t gradients:
Ex=zeros(NLPC,NPPL);
Ex_=diff(XW,1,2);
Ex(:,1:NPPL-1)=Ex_(:,:,1)+Ex_(:,:,2);
Ex=Ex+[zeros(1,NPPL);Ex(1:NLPC-1,:)];
Ex(1,:)=0;

Ey=zeros(NLPC,NPPL);
Ey_=-diff(XW,1,1);
Ey(2:NLPC,:)=Ey_(:,:,1)+Ey_(:,:,2);
Ey=Ey+[Ey(:,2:NPPL) zeros(NLPC,1)];
Ey(:,NPPL)=0;

Et=diff(XW,1,3);

Ex=Ex/4;Ey=Ey/4;Et=Et/4;
```

Figura 36. Rutinas de cálculo de flujo óptico.

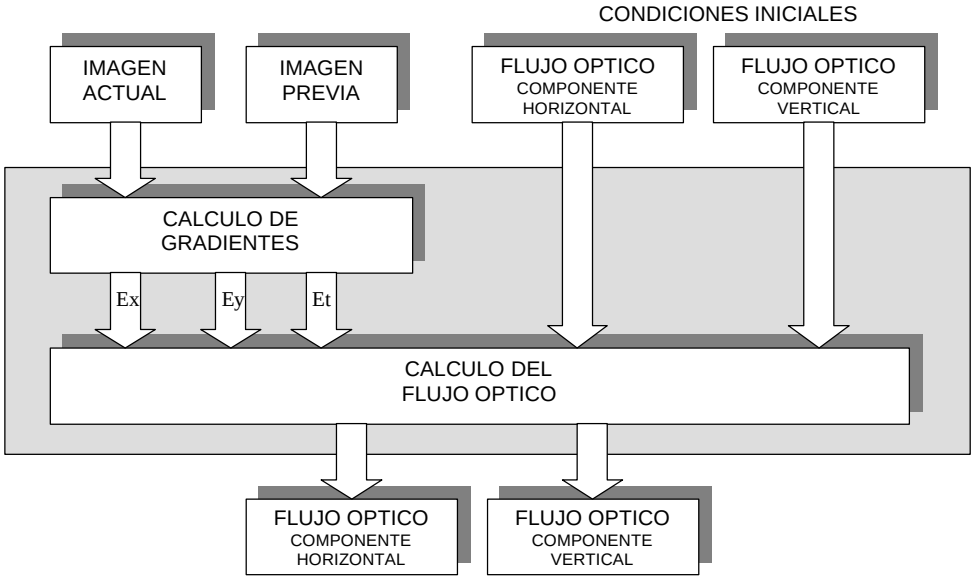


Figura 37. Representación esquemática de las rutinas de cálculo de flujo óptico.

El *interfaz gráfico de usuario* (GUI: *Graphic Program Interface*) se ha desarrollado en lenguaje C++ utilizando la biblioteca de clases MFC de Microsoft, lo que hace que el programa tome la apariencia característica de un programa de Windows. Para ello se ha utilizado la herramienta de desarrollo Microsoft Visual C++ 5.0.

La interfaz gráfica de usuario GUI es la que permite una interacción con el usuario de una manera cómoda. La pantalla de presentación al usuario se muestra en la figura 38 y en ella se distinguen 4 secciones o ventanas. Una primera ventana muestra el directorio de trabajo de donde se tomarán los ficheros de imágenes en formato *bitmap*. La segunda ventana muestra la lista de ficheros *bitmap* seleccionados para constituir la secuencia de imágenes. La tercera ventana muestra una de las imágenes seleccionadas en la lista correspondiente a la secuencia. Por último está la ventana de resultados donde aparecen, superpuestos a la última imagen procesada, los vectores calculados de flujo óptico utilizando líneas coloreadas que se destacan fácilmente sobre las imágenes en niveles de gris.

Figura 38. Pantalla de la interfaz gráfica de usuario.

El API es una herramienta incluida en el paquete *MATLAB* con la cual es posible el desarrollo de aplicaciones como la aquí descrita, permitiendo el uso de dos lenguajes tan distintos como el *MATLAB* y el C++. El API es básicamente una librería que incluye los recursos de enlace entre el *MATLAB* y una aplicación externa.

4

Capítulo 4

Arquitectura para la determinación del flujo óptico

Toda frase que yo emita
habrá de ser considerada por vosotros
no como una aseveración
sino como una pregunta

Niels Bohr

4.1. Realización en bloques de las ecuaciones de flujo óptico

4.1.1. Definición de la arquitectura

Para la obtención del flujo óptico a través de las ecuaciones formuladas por Horn y Schunck es necesario estudiarlas con cierto detalle para determinar la arquitectura más conveniente para cumplir con los objetivos de desarrollar una arquitectura que proporcione un flujo continuo de datos en tiempo real.

Las ecuaciones de interés (2.22) fueron desarrolladas en el capítulo 2 de esta Tesis, y para propósitos de comodidad se vuelven a reproducir aquí:

(4.1 (2.22))
$$\begin{cases} u = \bar{u} - E_x \dfrac{E_x\bar{u} + E_y\bar{v} + E_t}{a^2 + E_x^2 + E_y^2} \\ v = \bar{v} - E_y \dfrac{E_x\bar{u} + E_y\bar{v} + E_t}{a^2 + E_x^2 + E_y^2} \end{cases}$$

donde $\bar{u}$ y $\bar{v}$ están definidos sobre el resultado de la ecuación según las ecuaciones mostradas a continuación y que también son representadas gráficamente en la máscara de la figura 39.

(4.2 (2.21))
$$\begin{cases} \bar{u}_{i,j} = \frac{1}{6}\left(u_{i-1,j} + u_{i,j+1} + u_{i+1,j} + u_{i,j-1}\right) + \frac{1}{12}\left(u_{i-1,j-1} + u_{i-1,j+1} + u_{i+1,j+1} + u_{i+1,j-1}\right) \\ \bar{v}_{i,j} = \frac{1}{6}\left(v_{i-1,j} + v_{i,j+1} + v_{i+1,j} + v_{i,j-1}\right) + \frac{1}{12}\left(v_{i-1,j-1} + v_{i-1,j+1} + v_{i+1,j+1} + v_{i+1,j-1}\right) \end{cases}$$

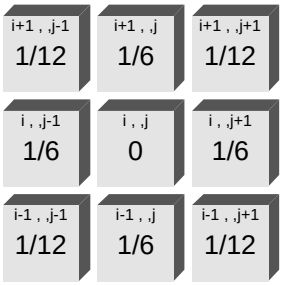


Figura 39. Máscara para la estimación de $\bar{u}$ y $\bar{v}$.

Lo más relevante de las ecuaciones 4.2 es que son iterativas, es decir, que el resultado depende de un resultado anterior. Inicialmente se asume un resultado hipotético, que bien puede ser un valor cero para todo el flujo óptico de una secuencia de imágenes. Con ello se

obtiene el primer resultado, el cual se introduce nuevamente en la ecuación (calculando los términos $\bar{u}$ y $\bar{v}$) para obtener el segundo resultado, y así sucesivamente hasta que los valores del flujo óptico converjan a valores estables.

En el procesamiento de una serie de imágenes, es preciso obtener el valor de los gradientes. Para ello se requiere tener por lo menos dos imágenes, ya que lo requiere el cálculo del gradiente temporal. Por ello, una de las posibles arquitecturas que pudiera plantearse sería la representada en la figura 40.

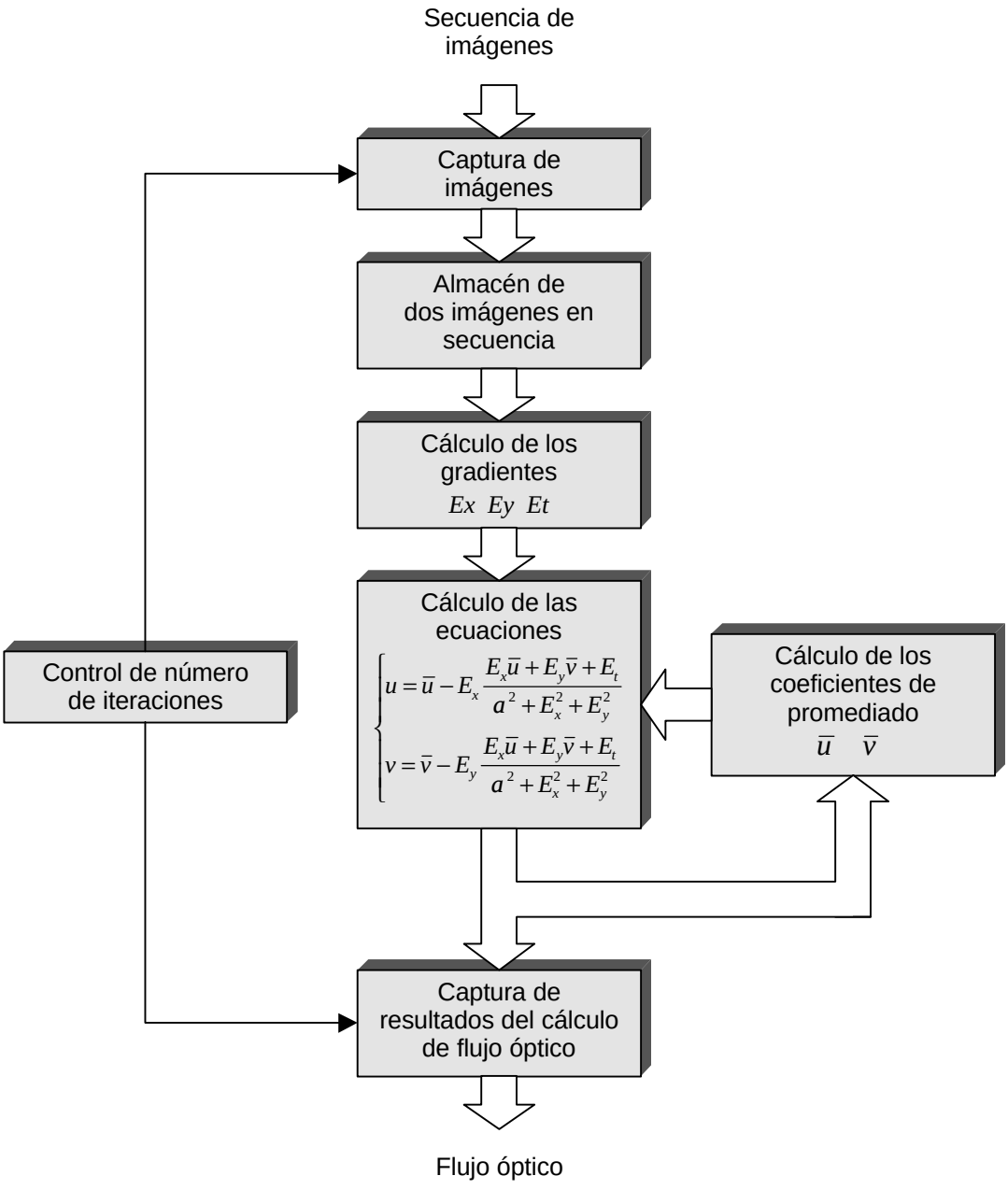


Figura 40. Posible arquitectura para el cálculo de flujo óptico utilizando las ecuaciones iterativas.

En esta arquitectura, se capturan dos o más imágenes de una secuencia de vídeo y se almacenan para su posterior procesamiento. Durante la etapa de procesamiento, se toma la

información de tales imágenes y se calculan los gradientes, los cuales permanecerán constantes durante todo el proceso iterativo. El cálculo de las ecuaciones se lleva a cabo por medio de un proceso iterativo que se repite un número fijo de veces, el cual debe ser ajustado para obtener una buena aproximación al flujo óptico. Cuando termina el proceso iterativo, se da salida al resultado, se capturan otras dos o más imágenes y se comienza nuevamente otro proceso iterativo.

Una arquitectura de este tipo ha sido propuesta por P. Cobos y F. Monasterio-Huelin [COBO98] y con ella han logrado procesar secuencias de imágenes de 50 x 50 *pixels* de resolución a una velocidad de 19 cuadros por segundo. En este sistema emplean un proceso de tres iteraciones para calcular el flujo óptico.

Pero para resultados medianamente aceptables se requiere un gran número de iteraciones de manera que los valores de flujo óptico converjan adecuadamente. Aunque no se ha hecho un estudio riguroso de la convergencia de las ecuaciones con secuencias de imágenes de diversos tipos, es habitual encontrar en las resoluciones de éstas ecuaciones por *software* utilizando entre 50 y 500 iteraciones [GHOS96] [LAPL96] [MITI88] [NESI95].

El trabajo de Cobos y Monasterio-Huelin nos sirve para analizar la magnitud del problema utilizando la arquitectura en cuestión. La utilización de imágenes de 256 x 256 *pixels*, como se propone en esta tesis requiere una mejora de 35 veces en el tiempo de procesamiento. El aumento del número de imágenes a 25 por segundo como se ha planteado, requiere 1.3 veces. Y finalmente, el aumento de iteraciones a un número más razonable de 64 requiere una mejora de 21 veces en el tiempo de procesamiento, para una mejora total de casi 1000 veces.

Es muy improbable que los dispositivos electrónicos puedan mejorar a corto o medio plazo en una escala tan grande como para que sea viable una arquitectura como la planteada. Por ello es necesario buscar la solución por otro lado. En el capítulo 2 se pudieron analizar una serie de arquitecturas útiles para el procesamiento de imágenes a alta velocidad, y en base a una extensiva aplicación de las mismas, se establece una arquitectura que será la aportación principal de esta Tesis.

La idea principal para desarrollar una arquitectura viable se encuentra en el mismo texto original de Horn y Schunck [HORN81]. En el mismo se menciona que las iteraciones, en lugar de hacerse para un solo cuadro de la imagen, pueden hacerse a lo largo de los sucesivos cuadros, ya que los cambios de un cuadro a otro son pequeños, y los resultados obtenidos para cuadros anteriores son válidos para los nuevos y ya comienzan a alcanzar una convergencia. Adicionalmente, también se menciona la mayor inmunidad al ruido de un sistema de esta naturaleza.

Este método ha sido aplicado en muchas realizaciones en *software* [GHOS97] [MEMI98] [NG98] [ONG99] [TIST96] [XIE96]. Ahora se propone el desarrollo del mismo con un *hardware* de aplicación específica. A grandes rasgos, la arquitectura aquí propuesta se presenta en la figura 41. Tal y como se ha señalado en capítulos anteriores, la información contenida en las secuencias de imágenes pueden ser transmitidas en forma serial y en muchos casos esa información serial puede ser procesada por arquitecturas segmentadas.

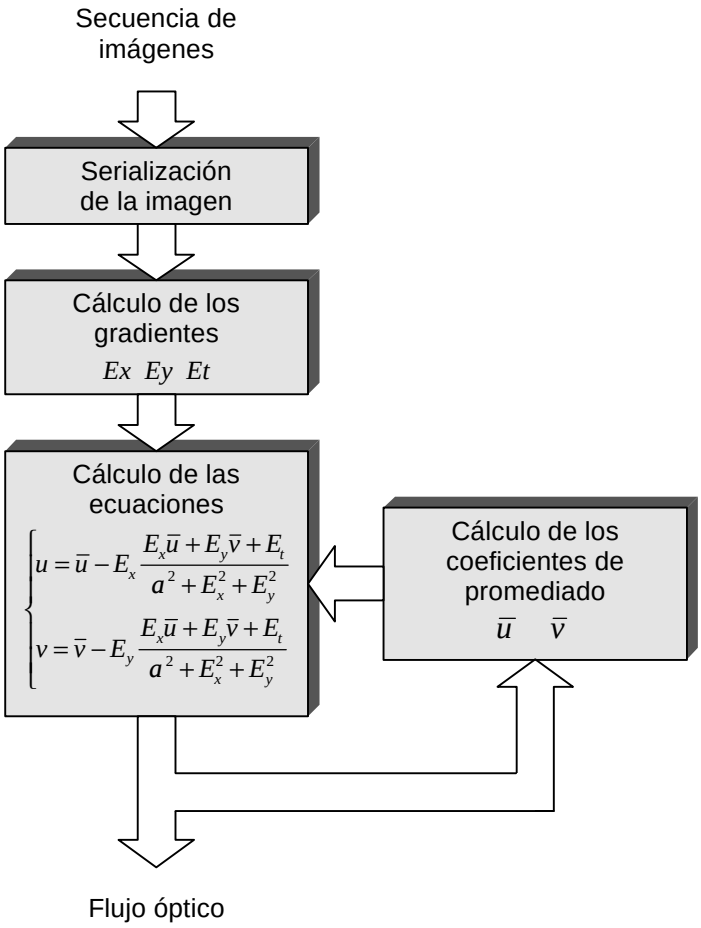


Figura 41. Posible arquitectura para el cálculo de flujo óptico utilizando las ecuaciones iterativas.

En la arquitectura segmentada propuesta, las imágenes deben fluir *pixel* a *pixel*, y el cálculo se realiza por etapas. De esta manera, mientras se calcula parte de la ecuación del flujo óptico para un *pixel*, se calculan los gradientes para otro *pixel*. Los resultados fluyen a la misma velocidad en que los *pixels* de la imagen entran al sistema, con una latencia de algo más de un cuadro, para permitir el cálculo de los gradientes temporales que requieren la información del cuadro actual y el cuadro anterior de la imagen.

La realimentación requerida para introducir la iteratividad en las ecuaciones se obtiene a partir del mismo flujo óptico de la salida. De esta manera, la restricción de suavidad se obtiene a partir de los resultados anteriormente obtenidos para el flujo óptico.

En los apartados siguientes se construirá paso a paso la arquitectura del sistema para el cálculo del flujo óptico, detallando cada una de las partes que lo componen y la relación entre los mismos.

4.1.2. Entradas y salidas del sistema

Antes de entrar en detalles sobre los bloques que constituyen el sistema es menester detenerse a ver cuáles son las entradas y salidas del sistema y sus características más relevantes.

En primer lugar, es dado que la técnica de procesamiento propuesta es enteramente digital, la entrada de las imágenes al sistema debe ser digital y los *pixels* deben fluir en forma serial. Por otro lado, tal y como se observa en las ecuaciones iterativas que se pretenden implementar, la información requerida de las imágenes viene expresada en términos del nivel de gris. Por ello se utilizará una fuente de señales de vídeo de niveles de gris suprimiendo la información de color.

Las entradas del sistema deben cumplir con las características descritas en la sección 3.1 referente a la señalización entre módulos segmentados, y que son las siguientes:

- Señal digital en nivel de gris. Cada *pixel* codificado con un nivel de gris de 8 bits, representando con 255 (FFh) el nivel de máximo brillo (blanco) y con cero (00h) el nivel mínimo (negro).
- Señal de reloj para entrada de *pixels* de 5 MHz, es decir, los *pixels* fluyen con un período de 200 ns.
- Imágenes con una resolución de 256 x 256 *pixels.*
- Flujo de 50 imágenes por segundo.
- Señales de sincronización vertical y horizontal

Para determinar el tipo de salida del sistema es necesario conocer lo que se pretende obtener del sistema: el flujo óptico. Este es un campo vectorial, por lo que cada punto del plano de imagen requiere de un par de números que lo definan: el módulo y la dirección, o las componentes vertical y horizontal de los vectores. Por ello puede decirse que por cada imagen de entrada existen dos "imágenes" de salida.

En la arquitectura propuesta en esta Tesis se dispone de dos salidas que representan las componentes vertical y horizontal de los vectores de velocidad del flujo óptico. Las señales en ambas salidas del sistema pueden ser del mismo formato que las señales a la entrada, es decir, la información referente a la velocidad de cada uno de los *pixels* fluye a una velocidad de 5 MHz e incluye las señales de sincronismo horizontal y vertical.

Los *pixels* en la imagen de entrada son representados en 256 niveles de gris codificados numéricamente en 8 bits. Las componentes horizontal y vertical del flujo óptico, pueden tomar tanto valores positivos como negativos, por ello, conservando los 8 bits utilizados en el formato de las señales que se consideran, es posible representar valores comprendidos entre +127 y -127 *pixels*/cuadro. Este rango de valores es suficientemente grande para representar los valores prácticos que se obtienen de las imágenes.

Concluyendo, las señales a la salida del sistema cumplen con las siguientes características:

- Dos salidas digitales de 8 bits que representan la velocidad horizontal y vertical del flujo óptico en cada *pixel* de la imagen. Estas están codificadas con valores comprendidos entre -127 representando con uno (01h) y +127 representado con 255 (FFh).
- Señal de reloj para la salida de los valores del flujo óptico en cada *pixel* de 5 MHz, es decir, la información fluye con un período de 200 ns.

- El flujo óptico se obtendrá con una resolución de 256 x 256 vectores por imagen.
- Una estimación de flujo óptico por imagen, es decir 50 estimaciones por segundo.

En la figura 42 se hace una representación detallada de las características de las señales de entrada y salida del sistema para la determinación del flujo óptico.

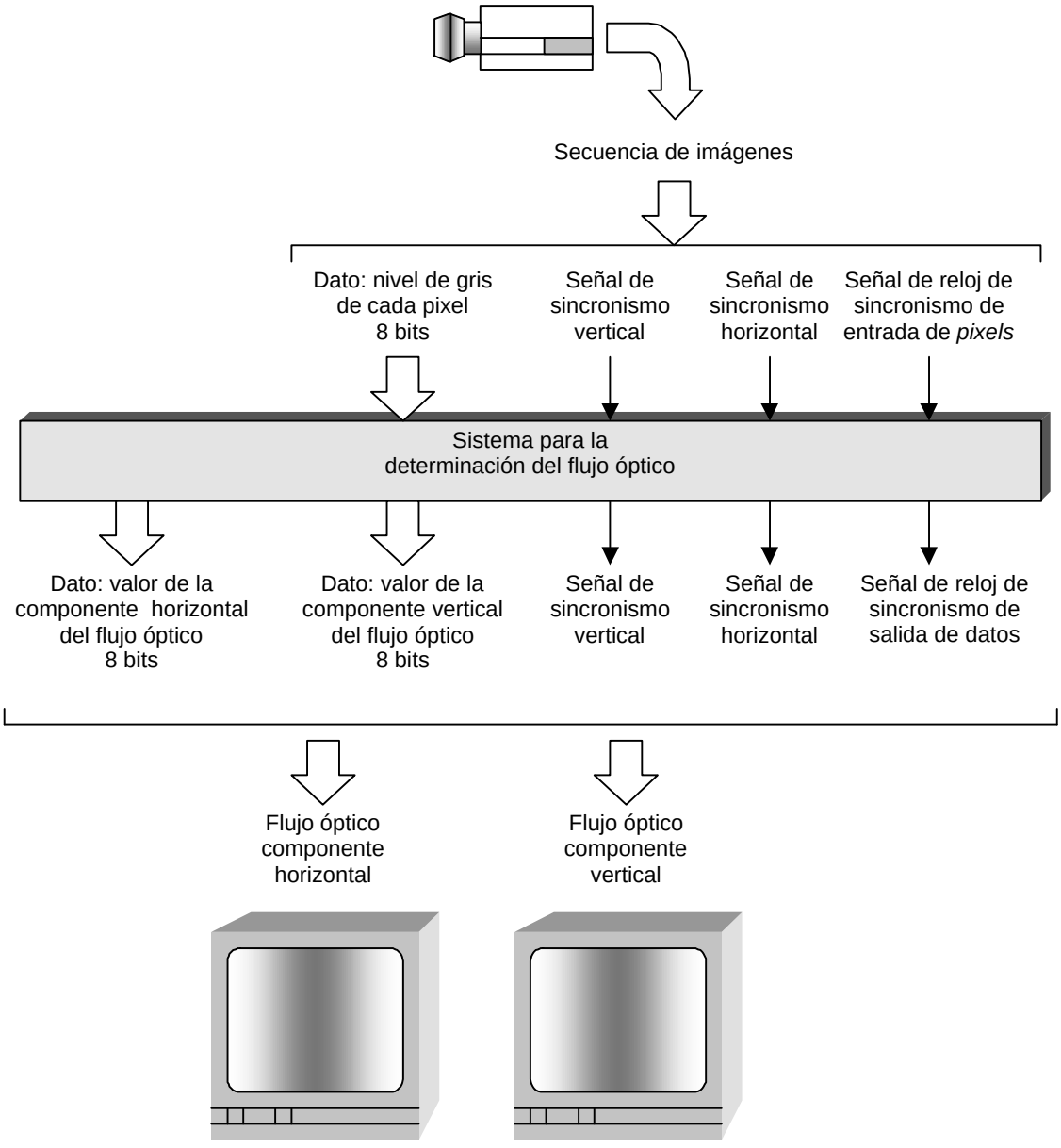


Figura 42. Señales de entrada y salida del sistema para la determinación del flujo óptico.

4.1.3. Descomposición en partes de las ecuaciones de flujo óptico

Para implementar las ecuaciones de flujo óptico (2.22) es necesario dividir las mismas en bloques de cálculo de manera que el problema se limite a desarrollar los bloques resultantes, esto facilitará el diseño del sistema, así como también las pruebas parciales del mismo.

Lo primero que salta a la vista en las ecuaciones de flujo es el extensivo uso que se hace de los gradientes espaciales y temporales E_x , E_y , E_t:

$$\begin{cases} u = \bar{u} - \boldsymbol{E}_x \dfrac{\boldsymbol{E}_x\bar{u} + \boldsymbol{E}_y\bar{v} + \boldsymbol{E}_t}{a^2 + \boldsymbol{E}_x^2 + \boldsymbol{E}_y^2} \\ v = \bar{v} - \boldsymbol{E}_y \dfrac{\boldsymbol{E}_x\bar{u} + \boldsymbol{E}_y\bar{v} + \boldsymbol{E}_t}{a^2 + \boldsymbol{E}_x^2 + \boldsymbol{E}_y^2} \end{cases} \tag{4.3}$$

Los gradientes son básicamente los cambios de nivel de gris en cada punto de la imagen. Por ello, para obtener un gradiente es necesario efectuar una resta de nivel de gris entre *pixels* adyacentes. Para poder disponer de la información de *pixels* adyacentes se requiere de una arquitectura como la estudiada en el apartado 2.3.2 de esta Tesis y representada en la figura 18. Sin embargo en este caso se requiere una adaptación de la misma para poder calcular los gradientes temporales: es necesario incluir también la información de dos cuadros de la secuencia de imágenes. Por ello se define un "cubo" de *pixels* (figura 43) para el cálculo de los gradientes. Es necesario aclarar que el cálculo del gradiente puede realizarse entre varios cuadros de la imagen, con lo cual se reduciría la sensibilidad del sistema al ruido, pero ello complicaría seriamente el resto de los bloques del sistema. Por esto se elige un cubo de 2 x 2 x 2 *pixels.*

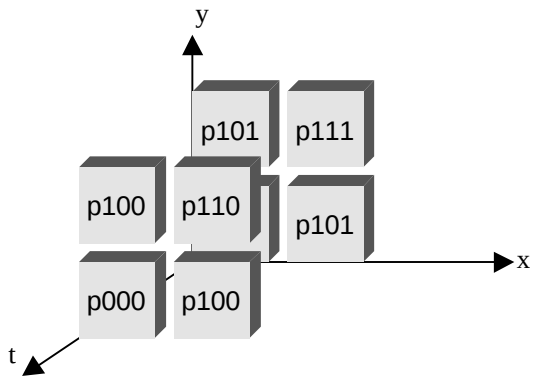


Figura 43. Cubo de *pixels* requerido para el cálculo de los gradientes espacio-temporales.

En la figura 44 se representa un esquema de la arquitectura necesaria para tomar de la secuencia de imágenes los *pixels* requeridos para el cálculo de los gradientes espacio-temporales. Básicamente es una estructura que almacena *pixels* y los suministra con diferentes retardos.

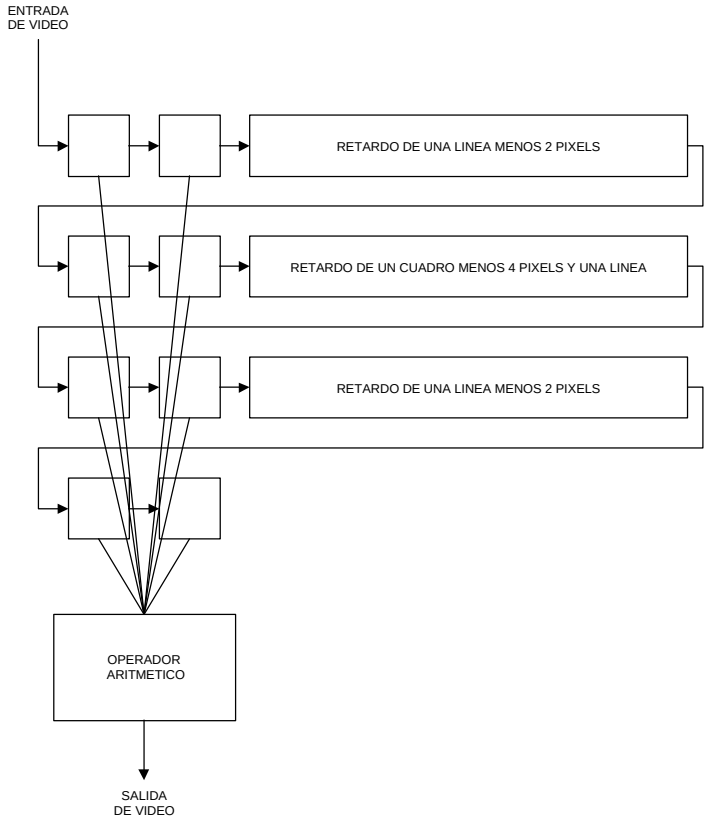


Figura 44. Estructura para la obtención de *pixels* para el cálculo de gradientes a partir de dos cuadros de una secuencia de imágenes.

De esta manera para obtener los gradientes se tienen dos bloques principales, uno encargado de suministrar los *pixels* requeridos y otro encargado de efectuar los cálculos con los mismos para obtener los gradientes (figura 45).

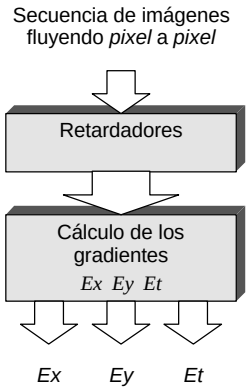


Figura 45. Diagrama esquemático para el cálculo de gradientes espacio-temporales.

Otro término que se repite en las ecuaciones es el que se denominará *Término D* y que aparece representado a continuación:

$$\begin{cases} u = \bar{u} - E_x \dfrac{E_x\bar{u} + E_y\bar{v} + E_t}{a^2 + E_x^2 + E_y^2} \\ v = \bar{v} - E_y \dfrac{E_x\bar{u} + E_y\bar{v} + E_t}{a^2 + E_x^2 + E_y^2} \end{cases}$$

Término D

Término D

(4.4)

$$D \equiv a^2 + E_x^2 + E_y^2 \tag{4.5}$$

Para calcular el *Término D* sólo se requieren los gradientes espaciales *(E_x, E_y)* y el término α. Este último, como se verá más adelante, es una constante.

En las ecuaciones también es posible calcular otro término que se presenta con frecuencia y que se denominará *Término P* y que se representa seguidamente:

$$\begin{cases} u = \bar{u} - E_x \dfrac{E_x\bar{u} + E_y\bar{v} + E_t}{a^2 + E_x^2 + E_y^2} \\ v = \bar{v} - E_y \dfrac{E_x\bar{u} + E_y\bar{v} + E_t}{a^2 + E_x^2 + E_y^2} \end{cases}$$

Término P

Término P

(4.6)

$$P \equiv E_x\bar{u} + E_y\bar{v} + E_t \tag{4.7}$$

A diferencia del *Término D*, el *Término P* depende no sólo de los gradientes espaciales *(E_x, E_y)*, sino que también del gradiente temporal *(E_t)* y los coeficientes de promediado ($\bar{u}$, $\bar{v}$). Estos últimos son los parámetros que hacen que la ecuación sea iterativa y su determinación se considera más adelante. En la figura 46 se representan los bloques requeridos en el sistema para calcular los *Términos P y D*, que se han venido discutiendo.

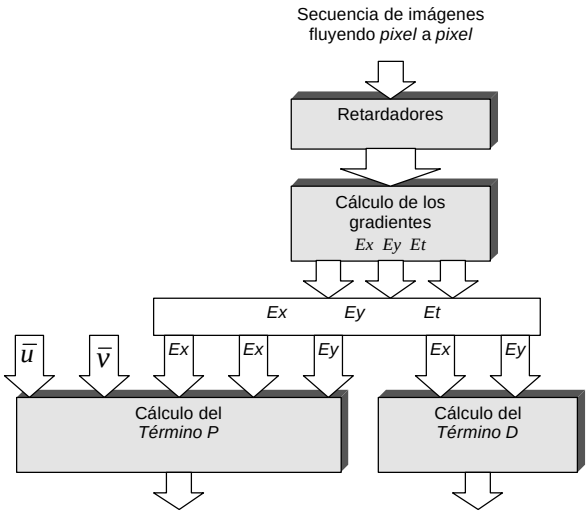


Figura 46. Diagrama esquemático para el cálculo de los *Términos P y D*.

Llegados a este punto, se pudiera pensar que el próximo término a calcular sería la división de los *Términos P y D* ya que es un término común para las dos ecuaciones que conforman el conjunto:

$$\begin{cases} u = \bar{u} - E_x \dfrac{E_x\bar{u} + E_y\bar{v} + E_t}{a^2 + E_x^2 + E_y^2} \\ v = \bar{v} - E_y \dfrac{E_x\bar{u} + E_y\bar{v} + E_t}{a^2 + E_x^2 + E_y^2} \end{cases} \tag{4.8}$$

Sin embargo, es necesario considerar que los cálculos realizados con arquitecturas *hardware* deben ser realizados preferiblemente utilizando aritmética entera, ya que generalmente requieren menos recursos. Por ello, en este caso, para no reducir las prestaciones del sistema que se desarrolla y para optimizar los recursos, se debe efectuar primero la multiplicación de los *Términos P* por los gradientes espaciales *(E_x, E_y)* y luego efectuar la división. De ello surgen los *Términos P_x y P_y*, representados a continuación:

$$\begin{cases} u = \bar{u} - E_x \dfrac{E_x\bar{u} + E_y\bar{v} + E_t}{a^2 + E_x^2 + E_y^2} \\ v = \bar{v} - E_y \dfrac{E_x\bar{u} + E_y\bar{v} + E_t}{a^2 + E_x^2 + E_y^2} \end{cases} \tag{4.9}$$

Término Px

Término Py

$$P_x \equiv E_x(E_x\bar{u} + E_y\bar{v} + E_t) \tag{4.10}$$

$$P_y \equiv E_y(E_x\bar{u} + E_y\bar{v} + E_t) \tag{4.11}$$

Una vez obtenidos los *Términos P_x y P_y* es posible efectuar la división entre éstos y el *Término D*, para obtener los *Términos R_x y R_y* que se definen tal y como se muestra a continuación:

$$\begin{cases} u = \bar{u} - E_x \dfrac{E_x\bar{u} + E_y\bar{v} + E_t}{a^2 + E_x^2 + E_y^2} \\ v = \bar{v} - E_y \dfrac{E_x\bar{u} + E_y\bar{v} + E_t}{a^2 + E_x^2 + E_y^2} \end{cases} \tag{4.12}$$

Término Rx

Término Ry

$$R_x \equiv \frac{P_x}{D} = \frac{E_x(E_x\bar{u} + E_y\bar{v} + E_t)}{a^2 + E_x^2 + E_y^2} \tag{4.13}$$

$$R_y \equiv \frac{P_y}{D} = \frac{E_y(E_x\bar{u} + E_y\bar{v} + E_t)}{a^2 + E_x^2 + E_y^2} \tag{4.14}$$

Con los *Términos R*, el último paso de la estimación es ya el cálculo de las velocidades horizontal y vertical del flujo óptico:

(4.15) $$u = \bar{u} - R_x = \bar{u} - \frac{E_x(E_x\bar{u} + E_y\bar{v} + E_t)}{a^2 + E_x^2 + E_y^2}$$

(4.16) $$v = \bar{v} - R_y = \bar{v} - \frac{E_y(E_x\bar{u} + E_y\bar{v} + E_t)}{a^2 + E_x^2 + E_y^2}$$

En la figura 47 se muestra de manera esquemática todos los bloques necesarios para implementar las ecuaciones 4.14 y 4.15.

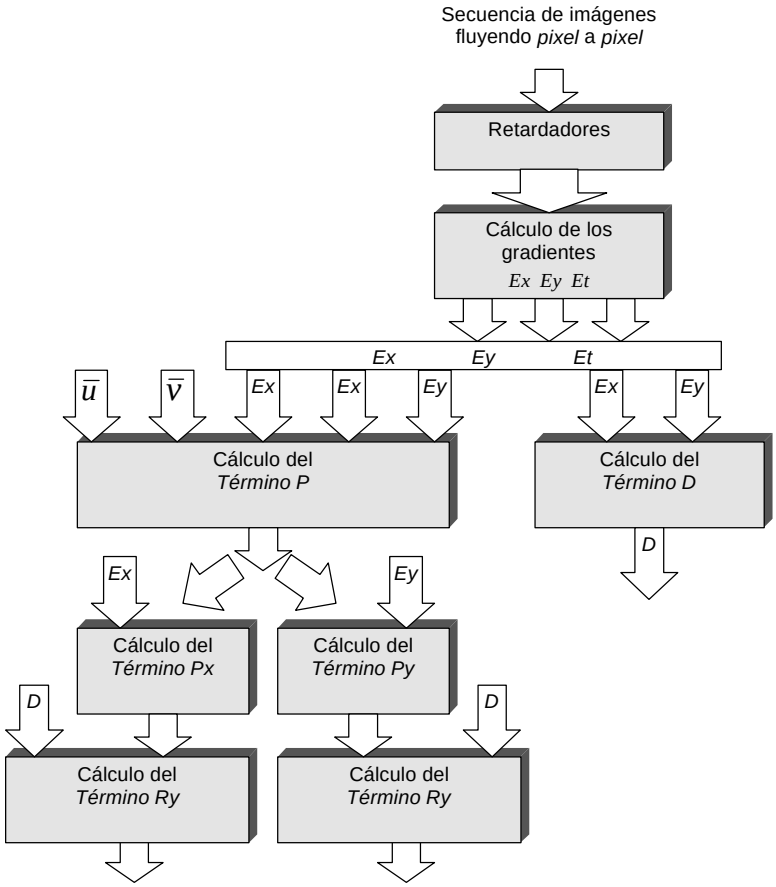


Figura 47. Diagrama esquemático para el cálculo de las ecuaciones de flujo óptico sin incluir la iteratividad.

En el camino sólo se ha dejado pendiente el cálculo de los coeficientes de promediado ($\bar{u}$, $\bar{v}$). Este cálculo fue definido en la sección 2.7.4 con las ecuaciones 2.21, y su determinación por medio de una máscara fue representada en la figura 27 que se implementa por medio de la estructura mostrada en la figura 18. Por comodidad, ambas figuras se reproducen en la figura 48. Es necesario acotar que son dos los coeficientes de promediado, uno para las velocidades horizontales y otro para las velocidades verticales, por lo que el sistema debe incorporar dos bloques iguales para la obtención de tales coeficientes.

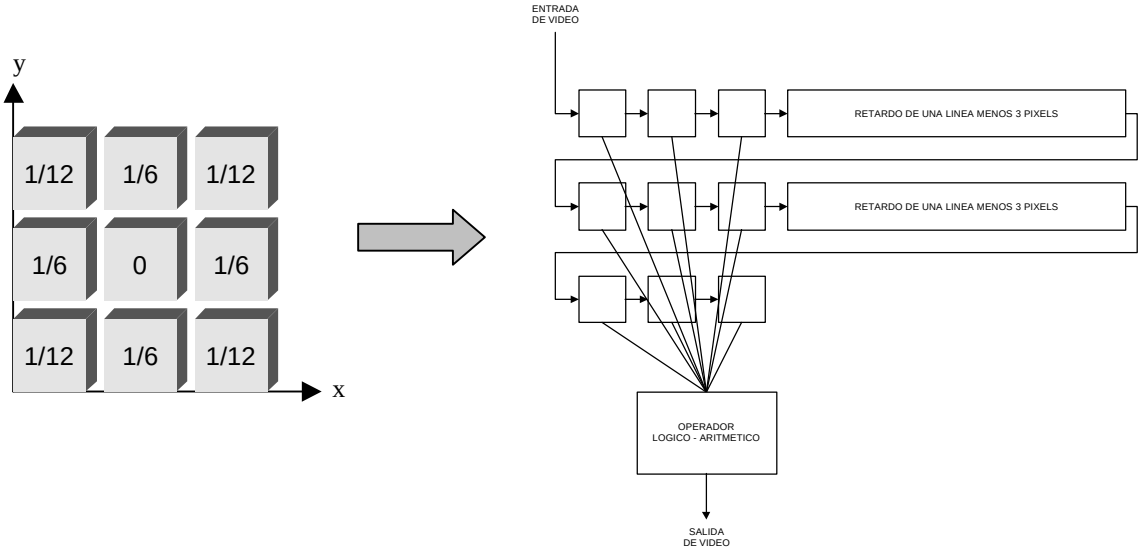


Figura 48. Máscara para la obtención de las variables de promediado y arquitectura asociada.

Es necesario señalar que los coeficientes de promediado deben ser introducidos en las ecuaciones de forma iterativa de manera que, primero debe realizarse una determinación de velocidades, a continuación se calculan los coeficientes de promediado para esas velocidades y por último deben introducirse en el sistema para la próxima determinación de velocidades. Esto implica que los coeficientes calculados a medida que se realiza la determinación de velocidad deben ser retenidos o almacenados hasta el comienzo de la próxima determinación. Esto debe realizarse con unas memorias de tipo FIFO.

Todos los cálculos se llevan a cabo sin menospreciar ninguno de los bits que es posible obtener. Sin embargo, una vez determinadas las velocidades, estas deben estar comprendidas entre ciertos valores. Los resultados de las ecuaciones de flujo óptico vienen expresados en *pixels* por cuadro, es decir, en las mismas unidades que fueron calculados los gradientes. Los valores muy grandes en las estimaciones de velocidad indicarían que existe una discontinuidad o ruido que debe ser suavizado, por lo que limitar el valor de las mismas ayuda a que los resultados de suavizado converjan más rápidamente a valores más reales. En el sistema desarrollado en esta Tesis, a fin de aprovechar las posibilidades de la arquitectura se limita la dispersión de valores de velocidad entre -127 y +127, esto es, una representación en ocho bits.

El cálculo de los coeficientes de promediado requiere que los valores de velocidad estén codificados en complemento a dos, pero las salidas del sistema tienen una codificación distinta, tal y como fue definido en la sección 4.1.2. Por ello se requiere una conversión entre la codificación complemento a dos y la codificación de las salidas. En la figura 49 se observa un diagrama esquemático de todos los elementos que componen el sistema.

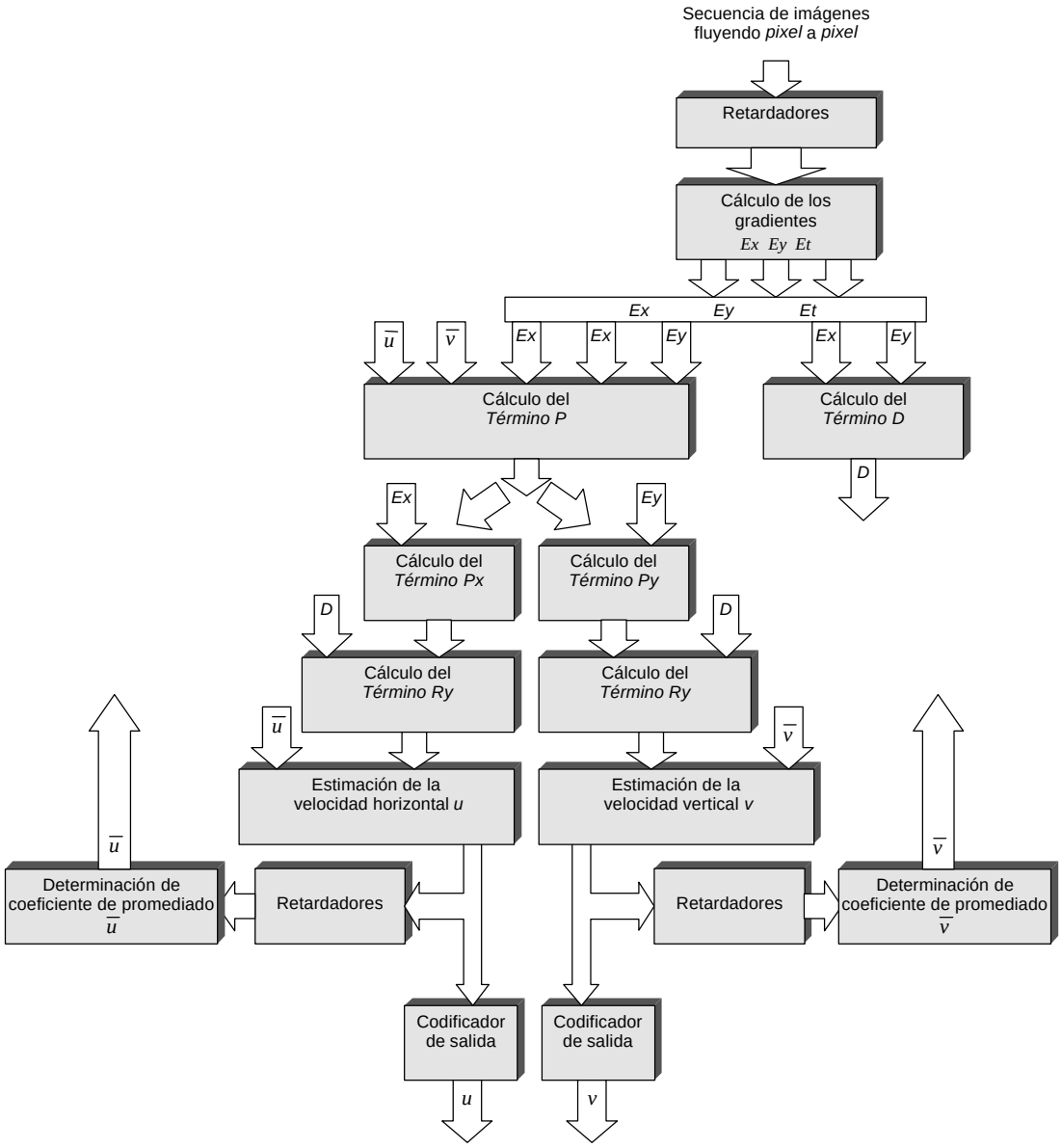


Figura 49. Diagrama esquemático del sistema para la determinación del flujo óptico.

4.2. Diagrama general de bloques

Con el diagrama esquemático de la figura 49 es posible determinar los bloques necesarios para llevar a cabo el sistema. Tratando de mantener una estructura modular durante todo el diseño, se divide el sistema en bloques que serán diseñados y estudiados independientemente. En la figura 50 se observa el diagrama de bloques del sistema, señalándose además las líneas que conectan a los mismos. Dado que la información que se pasa de un bloque a otro es numérica, en muchas conexiones se detalla el número de líneas requeridas para expresar en forma binaria el número y en forma separada se agrega un bit para el signo. De esta manera para expresar un número de 8 bits y signo se señala 8+1.

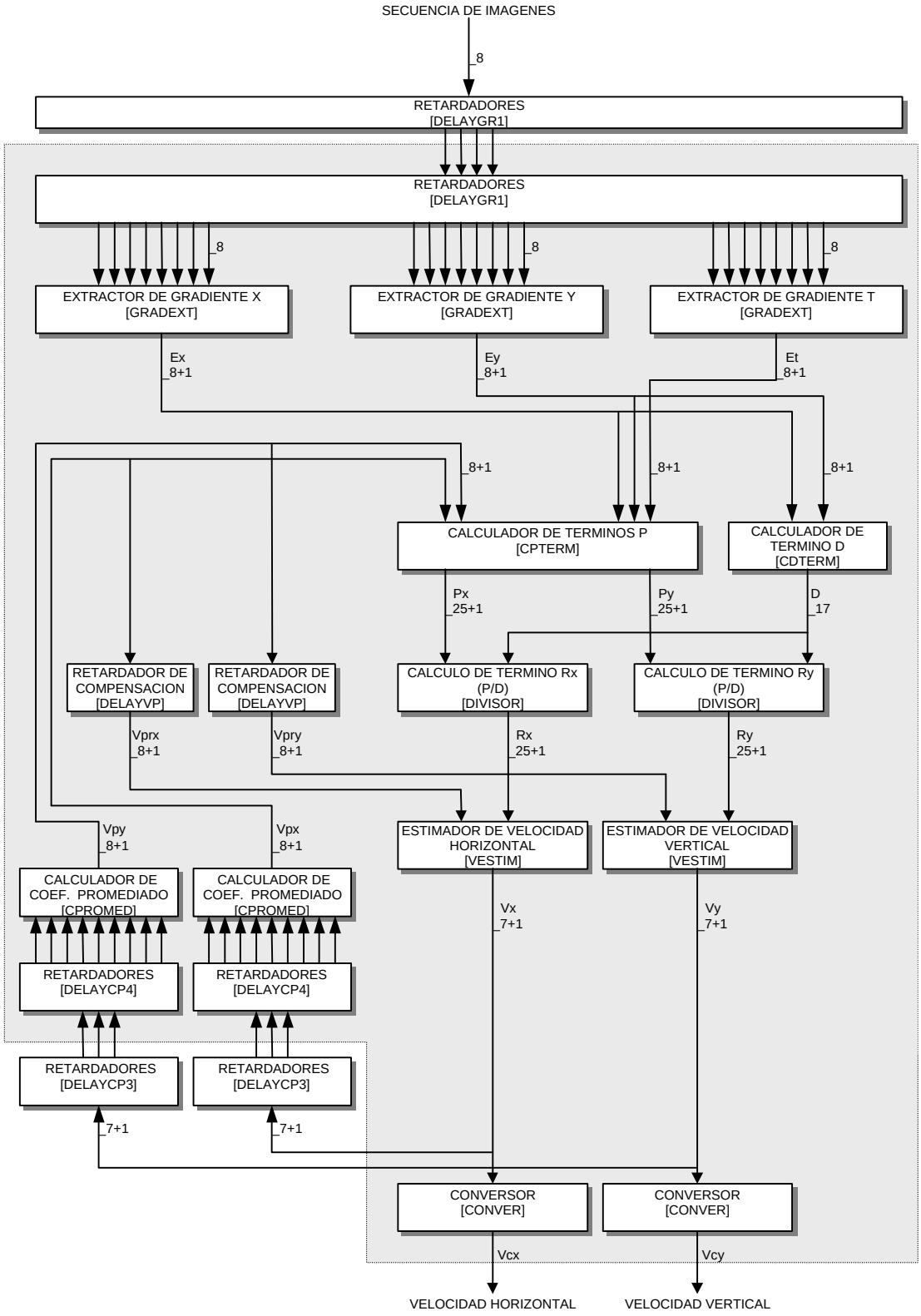


Figura 50. Diagrama general de bloques del sistema.

Para el desarrollo en VHDL de la arquitectura propuesta se ha modularizado toda la descripción en varios ficheros los cuales se muestran en la tabla 4. Cada uno de estos módulos se analizará detalladamente en las secciones siguientes. También en la misma tabla 4 se agrega una columna donde se indica el apéndice donde se lista la descripción del módulo en lenguaje VHDL.

Tabla 4. Módulos VHDL creados para describir el sistema.

Módulo	Descripción	Apéndice
Cdterm	Cálculo de término D	B1
Conver	Conversor	B2
Cpromed	Calculo de coeficiente de promediado	B3
Cpterm	Cálculo de términos P	B4
Ctrfifo	Control de FIFOs	B5
Delaycp1	Retardador de coeficientes de promediado	B6
Delaycp2	Retardador de coeficientes de promediado	B7
Delaycp3	Retardador de coeficientes de promediado	B8
Delaygr1	Retardador de gradiente primario	B9
Delaygr2	Retardador de gradiente secundario	B10
Delaysi	Retardador de un bit	B11
Delayvp	Retardador de compensación	B12
divip03	Divisor por 3 para cálculo de coeficientes de promediado	B13
Divisor	Divisor para cálculo de términos R	B14
Fpgablk1	Bloques a ser incluidos en la FPGA	B15
Gradext	Extractor de gradientes	B16
Memfifo	Descripción memorias FIFO	B17
Multsgn1	Multiplicador con signo	B18
Multsgn2	Multiplicador con signo	B19
Multsgn3	Multiplicador con signo	B20
Sincroni	Generador de señales de reloj y sincronismo	B21
System	Sistema y banco de pruebas	B22
Vestim	Estimador de velocidad	B23

4.2.1. Bloques retardadores para el cálculo de gradientes

Tal y como se mencionó en la sección 4.1.3, y se expresó gráficamente en la figura 44 y en la figura 45, para la determinación de los gradientes espacio-temporales de la imagen es necesario almacenar *pixels* de dos imágenes de la secuencia. Ocho de esos *pixels* deben ser suministrados de forma simultánea a la etapa de cálculo de gradientes.

Para realizar la arquitectura señalada en la figura 45, se utiliza una configuración de memorias FIFO y registros que se representa en la figura 51. Como se observa en dicha figura, se requiere una FIFO de 64 Kbytes para almacenar un cuadro de imagen de 256 x 256 *pixels*. También se requieren dos FIFOs de 256 bytes para almacenar una línea de 256 *pixels*. Por ultimo se requieren cuatro registros para almacenar *pixels*.

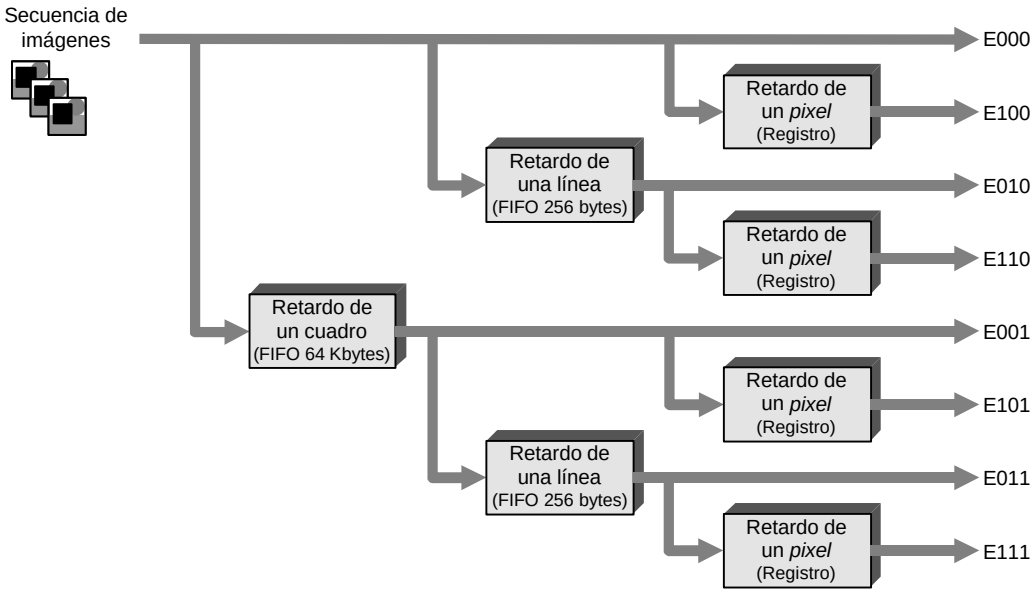


Figura 51. Conexión de FIFOs y registros para la obtención de *pixels* para el cálculo de gradientes.

El almacenamiento de un cuadro de imagen de 256 x 256 *pixels* requiere de una memoria FIFO de 65536 bytes, y el almacenamiento de una línea, 256 bytes. La utilización de esta cantidad de memoria dentro de las FPGA comerciales [ALTE01] [XILI99] es por el momento inviable. En la tabla 5 se presentan las capacidades de memoria presentes en varias de las FPGA actuales. Obsérvese que en dicha tabla las capacidades se presentan en número de bits y no en bytes, por ello es necesario señalar que para almacenar un cuadro de programa se requieren 524288 bits, y para una línea 2048 bits. Además, la utilización de esta memoria en algunos casos requiere el uso de otros recursos presentes en la FPGA. Por estas razones se ha tomado la decisión de considerar que el circuito de retardo de cuadros y líneas se realice fuera de la FPGA, utilizando para ello memorias FIFO comerciales (figura 52).

Tabla 5. Capacidades de memoria presentes en algunas FPGA comerciales.

Fabricante	Dispositivo	Máxima capacidad de memoria RAM (Kbits)
Altera	APEX EP20K1500C	442 368
Altera	APEX 20K400	212 992
Altera	FLEX 10K200	98 304
Xilinx	XC4085	100 352
Xilinx	Virtex XCV1000	131 072
Xilinx	Spartan XC2S200	57 344

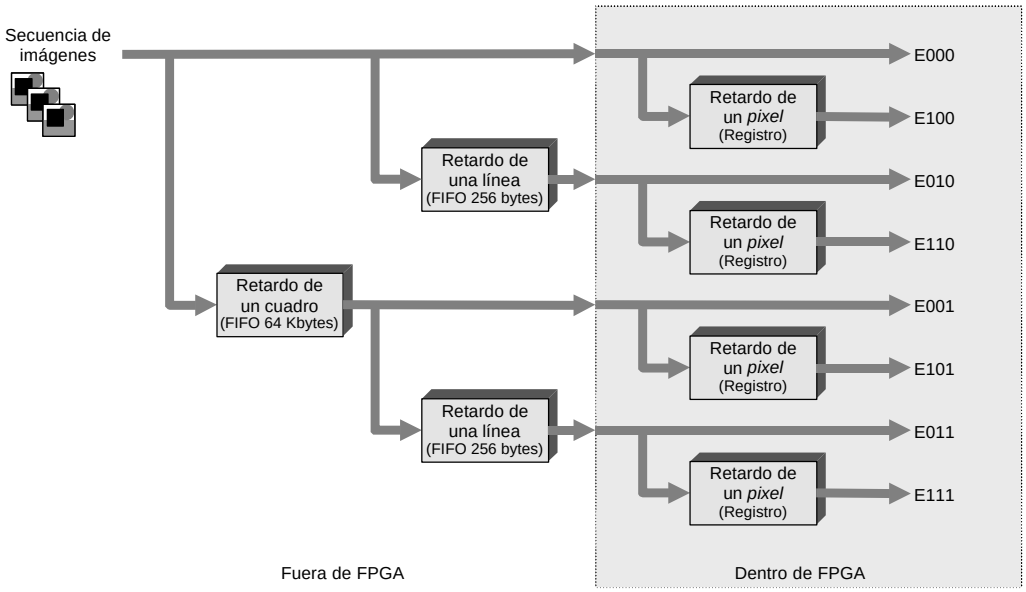


Figura 52. Separación entre bloques a desarrollar fuera de la FPGA (utilizando FIFOs) y dentro de la FPGA (utilizando registros).

Obviamente, la decisión de utilizar memorias FIFO comerciales afecta la manera en que debe ser descrita con VHDL esta parte del sistema. Mientras otras partes deberán describirse con el objetivo de sintetizarlas, las FIFOs comerciales se describirán por su modelo comportamental.

Se considera aquí el uso de memorias FIFO tales como las fabricadas por *Integrated Device Technology, INC* (serie IDT72xx) [IDT00] o *Cypress Semiconductor*, (IDT7208) [CYPR00] (Véase el Apéndice A). Estas han sido modelizadas para esta Tesis de la manera mostrada en el apéndice B17.

Para la descripción VHDL de los bloques retardadores de gradientes se crearon 3 módulos VHDL: memfifo, delaygr1 y delaygr2, los cuales se muestran en los apéndices B9, B10 y B17.

4.2.2. Bloque de cálculo de gradiente

A partir de los *pixels* suministrados por el bloque de retardadores analizado en la sección anterior se pueden calcular los gradientes espacio-temporales (E_x, E_y, E_z). Estos gradientes son básicamente diferencias promediadas entre los valores de grises de los *pixels* en cada una de las direcciones ortogonales posibles: horizontal, vertical y temporal. Seguidamente se presenta el cálculo de estos gradientes en forma de ecuaciones y siguiendo la nomenclatura introducida en la figura 43 y que se reproduce nuevamente a continuación (figura 53):

$$
\begin{aligned}
E_x &= \tfrac{1}{4}\left(E_{000} - E_{100} + E_{010} - E_{110} + E_{001} - E_{101} + E_{011} - E_{111}\right) \\
E_y &= \tfrac{1}{4}\left(E_{110} - E_{100} + E_{010} - E_{000} + E_{111} - E_{101} + E_{011} - E_{001}\right) \\
E_x &= \tfrac{1}{4}\left(E_{001} - E_{000} + E_{011} - E_{010} + E_{101} - E_{100} + E_{111} - E_{110}\right)
\end{aligned}
\tag{4.17}
$$

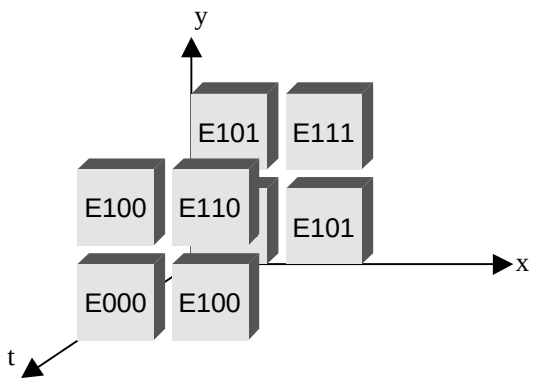


Figura 53. *Pixels* de dos imágenes utilizados en el cálculo de los gradientes espacio-temporales.

Las ecuaciones para el cálculo de los tres gradientes espacio-temporales son muy similares entre sí. Las operaciones a realizar son las mismas y lo que varía son los operandos. Por ello es posible crear un bloque estándar para el cálculo de gradiente que se utilizaría en cada uno de los casos. Este bloque se presenta en la figura 54.

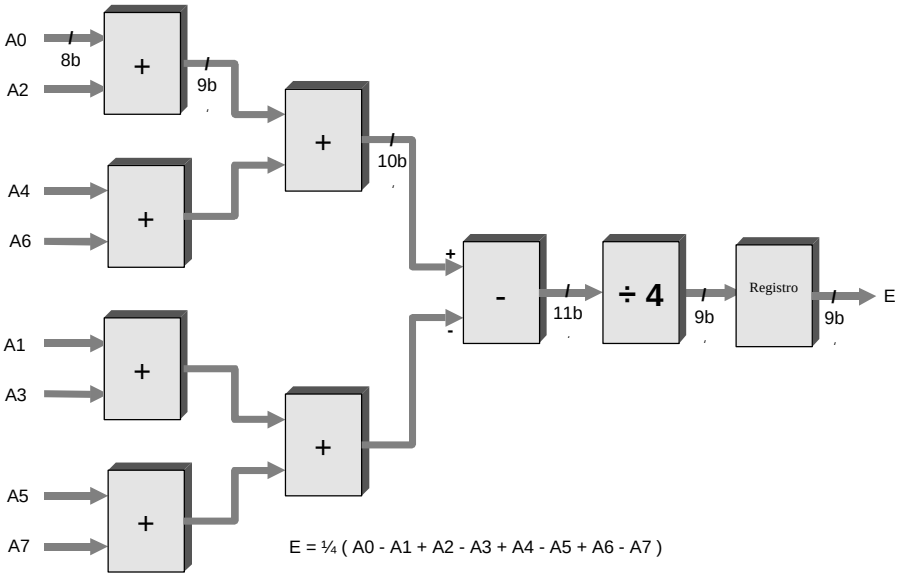


Figura 54. - Módulo de cálculo de gradiente en una dirección.

Teniendo en cuenta las operaciones que se llevan a cabo, el signo del resultado y el número de bits en que se recibe los *pixels* de la imagen es necesario prever una salida con un bit más que la entrada para el signo. En la figura 54 se observan las diferentes dimensiones en bits de los datos a medida que son calculados. El resultado final, es decir, el gradiente, se representa con 8 bits más el signo (9 bits).

Al construir el modelo en VHDL del bloque de cálculo de gradiente, se presentan ciertas diferencias entre los distintos compiladores, relacionadas con el número de bits de las entradas y las salidas de los operadores. La mejor manera de cumplir con los requerimientos

de todos los compiladores es extender el número de bits de la entrada al mismo número de bits que debe tener el resultado.

Para mantener la arquitectura segmentada del sistema, la salida de este registro se hace pasar por un registro que sincronizará el resultado con el reloj del sistema y permitirá independizar el funcionamiento de la tecnología utilizada en su realización. En el apéndice B16 se presenta la descripción VHDL del bloque de cálculo de gradiente.

Finalmente, los tres gradientes necesarios, horizontal, vertical y temporal, son calculados utilizando tres veces el bloque de cálculo de gradiente (figura 55), siguiendo las directrices del conjunto de ecuaciones (4.17).

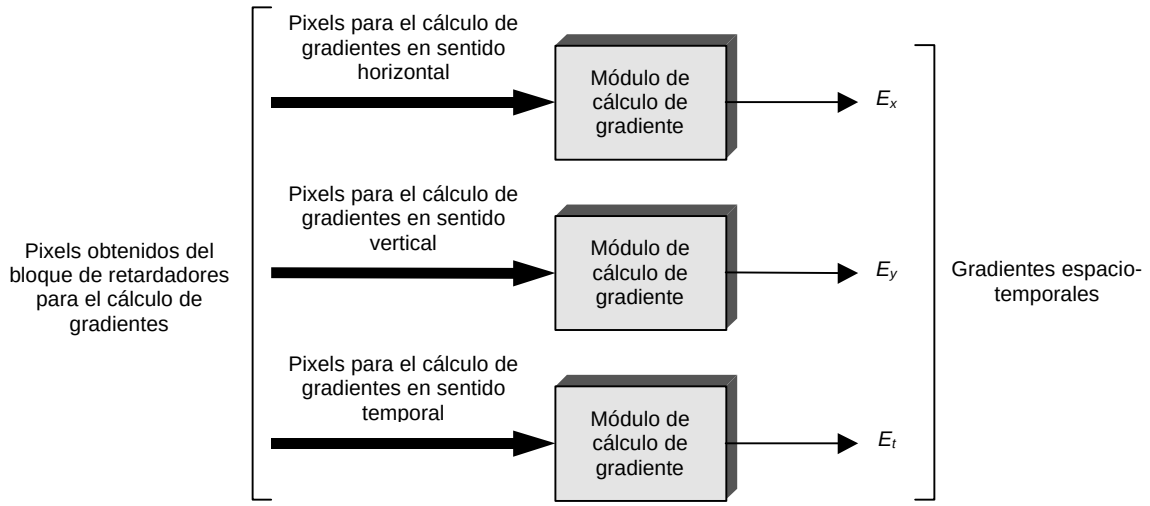


Figura 55. Cálculo de los gradientes espacio-temporales.

4.2.3. Bloque de cálculo de término D

En la sección 4.1.3 se definió el término D como parte de la ecuación de flujo óptico por medio de la ecuación (4.5) y que se reproduce aquí por conveniencia:

(4.5) $$D \equiv a^2 + E_x^2 + E_y^2$$

El término α^2 es utilizado para salvar la indeterminación que se produce en aquellos casos en que los gradientes espaciales son ambos cero. Este caso se produce a causa del problema de apertura analizado en la sección 2.7.2. Sólo se da en las zonas de la imagen donde el nivel de gris de la imagen es uniforme y por tanto no existe información relativa al movimiento de los *pixels*. El valor de α^2 permite agilizar la convergencia de la ecuación iterativa de flujo óptico reduciendo el peso de la ecuación básica de flujo óptico aumentando el peso de la restricción de suavidad. Un valor grande de α^2 agiliza la convergencia pero reduce la precisión de la determinación del flujo óptico. Los valores utilizados suelen ser pequeños y cercanos a la unidad [LAPL96]. En algunos casos se suele efectuar una variación dinámica del valor de α^2 tal y como lo proponen S. Ghosal y P. Vanek [GHOS96].

En el caso del sistema digital aquí analizado, podemos lograr un ajuste mejor variando dinámicamente el valor de α^2 porque es posible conocer en todo momento el valor de la suma de gradientes, e introducir la corrección sólo cuando ello haga falta, es decir:

$$\begin{cases} D = a^2 & \text{si} \quad E_x^2 + E_y^2 = 0 \\ D = E_x^2 + E_y^2 & \text{si} \quad E_x^2 + E_y^2 > 0 \end{cases} \tag{4.18}$$

Dado que en todo el sistema se hace uso de la aritmética binaria entera, el valor de α^2 se hace igual a la unidad. De esta manera el bloque de cálculo del término D aquí desarrollado tiene una ecuación como la que sigue:

$$\begin{cases} D = 1 & \text{si} \quad E_x^2 + E_y^2 = 0 \\ D = E_x^2 + E_y^2 & \text{si} \quad E_x^2 + E_y^2 > 0 \end{cases} \tag{4.19}$$

Los gradientes recibidos tienen módulo y signo, es decir, son representaciones binarias con signo en módulo 2. Sin embargo, los resultados deben ser positivos. El número de bits del resultado está en consonancia con el número de bits de los gradientes. Así el número final de bits será de 17 bits sin signo:

[Número de bits del resultado] = 2 * [número de bits del gradiente incluyendo el signo] - 1

Para el cálculo del término D se requiere del uso de bloques multiplicadores, ya que es necesario elevar los valores espaciales de gradientes al cuadrado. Para ello pudieran utilizarse las megafunciones propias del fabricante de la FPGA, pero para que el modelado en VHDL sea independiente de la implementación, se ha optado por preparar un bloque de multiplicación [BROW99] independiente del paquete que sea utilizado para la síntesis y simulación del sistema.

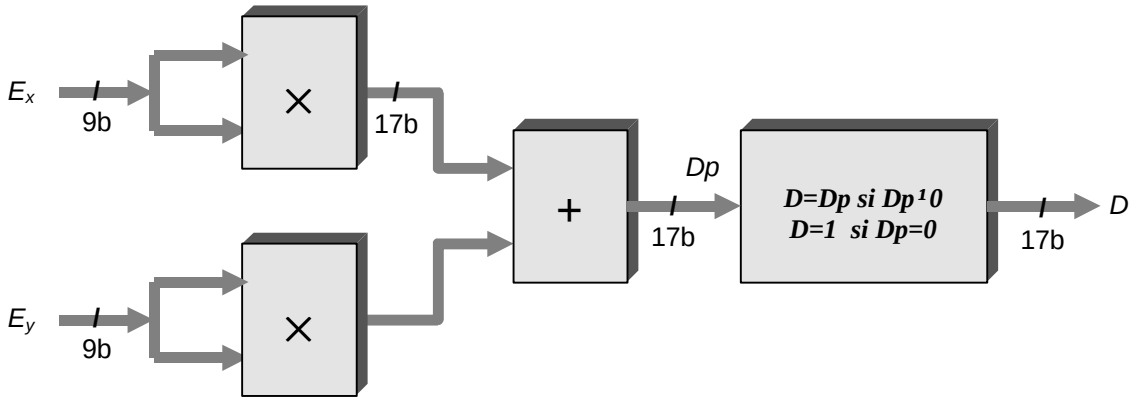


Figura 56. - Módulo de cálculo de término D.

Las descripciones VHDL de los módulos *cdterm* y *multsgn3* que integran el calculador de término D se muestran en los apéndices B1 y B20 respectivamente.

4.2.4. Bloque de cálculo de términos P

Al igual que el término D, en la sección 4.1.3 se definió el término P como parte de la ecuación de flujo óptico por medio de la ecuación (4.7) y que se reproduce aquí por conveniencia:

(4.7) $$P \equiv E_x\bar{u} + E_y\bar{v} + E_t$$

Esta ecuación guarda similitud con la del término D, y su realización física tiene también cierta similitud con la del término D. Los gradientes y los coeficientes de promediado recibidos tienen módulo y signo, es decir son representaciones binarias con signo en módulo 2 y los resultados deben ser también representaciones en módulo 2. El número de bits del resultado está en consonancia con el número de bits de los gradientes y de los coeficientes de promediado, y en este caso son 26 bits más el signo.

Se utilizaran bloques multiplicadores tal y como fueron utilizados para el cálculo del término D, por las mismas circunstancias antes mencionadas.

Por otro lado, una vez obtenido el término P, pueden calcularse los términos Px y Py, los cuales están definidos como:

(4.10) $$P_x \equiv E_x(E_x\bar{u} + E_y\bar{v} + E_t)$$

(4.11) $$P_y \equiv E_y(E_x\bar{u} + E_y\bar{v} + E_t)$$

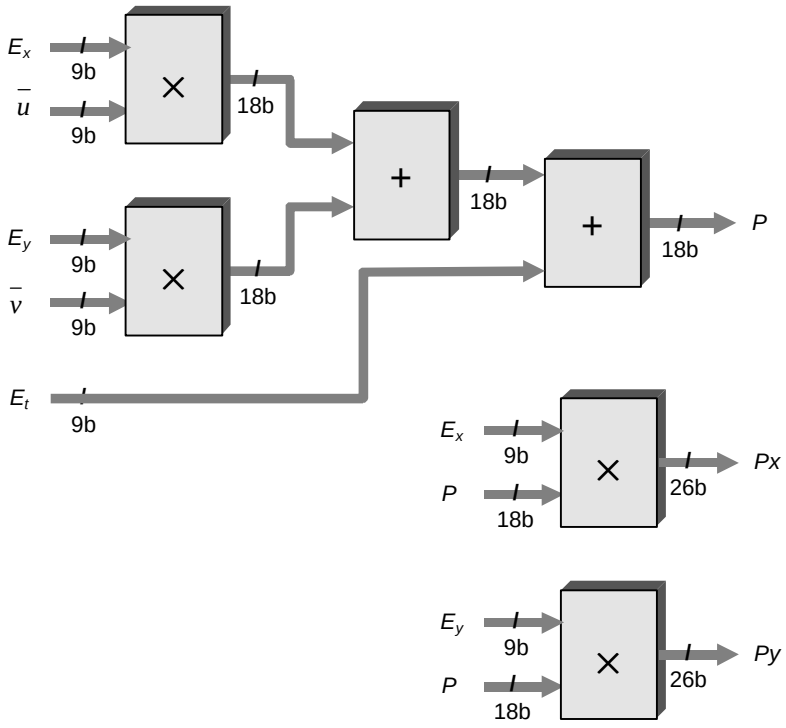


Figura 57. - Módulo de cálculo de términos Px y Py.

Las descripciones VHDL de los módulos *cpterm, multsgn1* y *multsgn2* que integran el calculador de términos P se muestran en los apéndices B4, B18 y B19 respectivamente.

4.2.5. Bloques de cálculo de términos Rx y Ry

Los términos Rx y Ry son los más difíciles de implementar por medio de circuitos lógicos dado que involucran la división matemática entre dos términos variables. En la sección 4.1.3 se definieron los términos Rx y Ry en las ecuaciones (4.13) y (4.14) que se reproducen a continuación:

(4.13) $$R_x \equiv \frac{P_x}{D} = \frac{E_x(E_x\bar{u} + E_y\bar{v} + E_t)}{a^2 + E_x^2 + E_y^2}$$

(4.14) $$R_y \equiv \frac{P_y}{D} = \frac{E_y(E_x\bar{u} + E_y\bar{v} + E_t)}{a^2 + E_x^2 + E_y^2}$$

En cuanto al signo de los operandos, para este caso, el numerador (*P*) puede ser positivo o negativo pero el denominador (*D*) es siempre positivo. Por ello el resultado será siempre del mismo signo del numerador. Esto simplifica algo el cálculo, se separa el bit de signo y se calcula el módulo. En caso de que el signo sea negativo habrá que volver positivo el numerador con una operación de complemento a 2.

El número de bits del numerador (*P)* es de 26. Conociendo que el menor valor que puede tomar el término D es la unidad, el resultado será a lo sumo de 26 bits.

Para lograr una alta velocidad de cálculo aprovechando al máximo los recursos *hardware* se optó por una arquitectura segmentada, con la que se puede obtener un flujo de datos constante y a alta velocidad, adecuado al propósito del sistema planteado. En este caso, al ser el resultado de 26 bits se requieren 26 iteraciones para obtener el resultado completo, por ello se hizo una segmentación en 26 bloques secuenciales (figura 58).

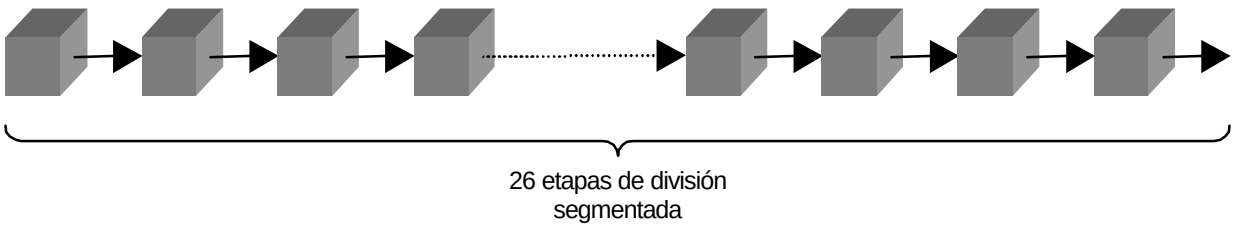


Figura 58. Segmentación de la división en 26 etapas.

La división se efectúa por medio de un algoritmo semejante al utilizado para efectuar divisiones con *papel-y-lápiz* conocido con frecuencia como algoritmo de *restitución* [BROW99] [CAPP99] [HEUR97] [MANO93] [OMON94]. Este algoritmo se representa en forma esquemática en la figura 59.

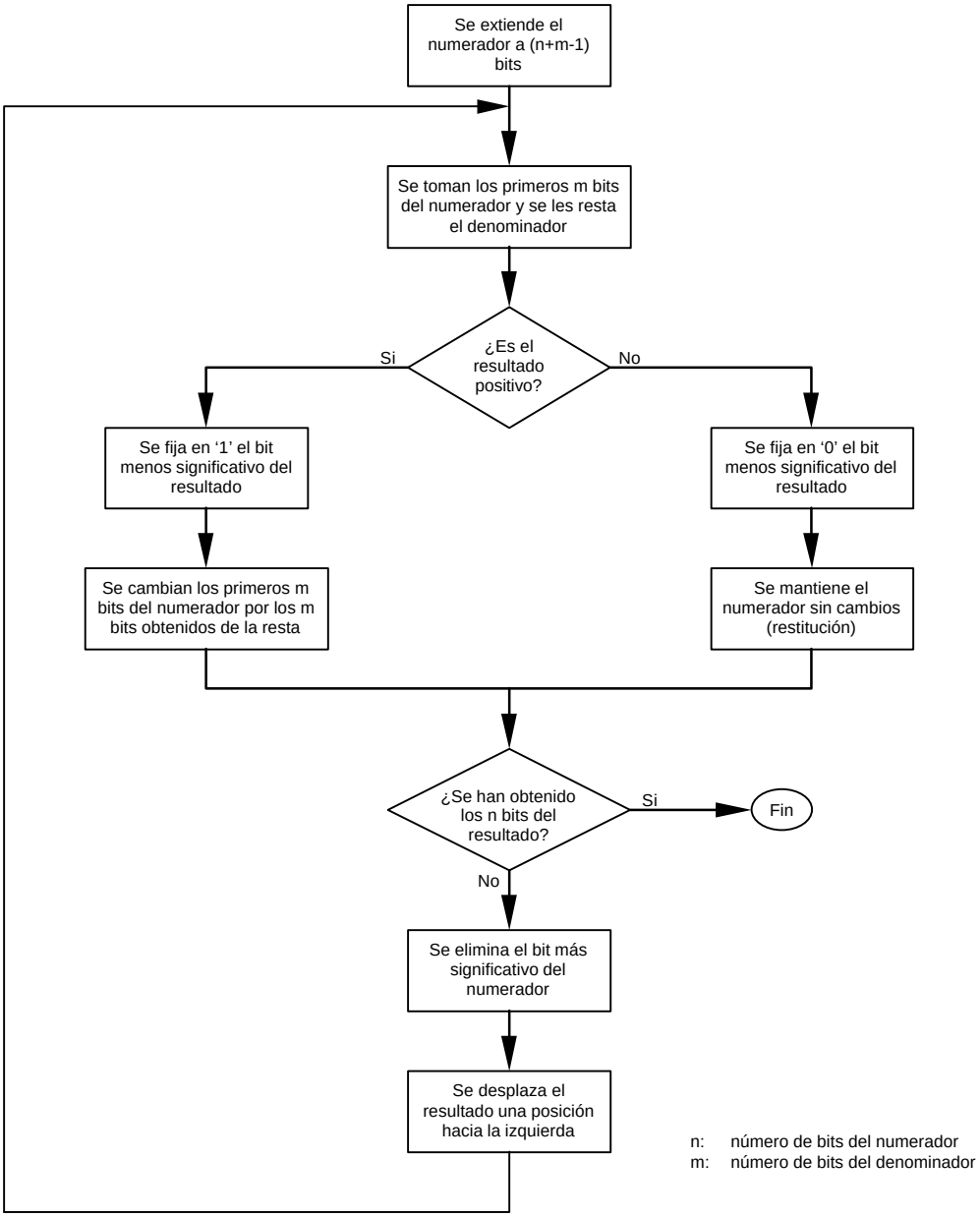


Figura 59. Algoritmo para la realización de la división.

La realización de estos algoritmos en lenguaje VHDL se lleva a cabo utilizando estructuras *GENERATE*, sin embargo éstas no están bien soportadas aún por todos los programas de síntesis de VHDL. Por ello, se optó por desarrollar el módulo de división que fuera igualmente interpretado por los simuladores como por los sintetizadores de VHDL (véase el apéndice B14). Esto origina un código VHDL más largo y difícil de leer, pero adecuado a los objetivos de esta Tesis.

Para facilitar cálculos posteriores, el valor calculado de los términos R no se expresa en complemento a 2, sino como signo y magnitud, esto es, el bit más significativo es el signo y los restantes son la magnitud. Dado que existen dos términos R, este módulo debe ser incluido dos veces en el sistema.

4.2.6. Bloques estimadores de velocidad

Una vez obtenidos los términos R, la obtención de la estimación de la velocidad se reduce a una resta entre los coeficientes de promediado y los términos R, tal y como fue expresado en las ecuaciones 4.14 y 4.15:

(4.15) $$u = \bar{u} - R_x = \bar{u} - \frac{E_x(E_x\bar{u} + E_y\bar{v} + E_t)}{a^2 + E_x^2 + E_y^2}$$

(4.16) $$v = \bar{v} - R_y = \bar{v} - \frac{E_y(E_x\bar{u} + E_y\bar{v} + E_t)}{a^2 + E_x^2 + E_y^2}$$

Sin embargo, hay que hacer una consideración previa antes de su realización por *hardware*. Los valores de los términos R pueden ser muy grandes. De hecho, los valores de R tienen 26 bits más signo. Sin embargo, los valores muy grandes indican que existen zonas de la imagen con *problemas de apertura*. Es necesario recordar que los términos R no están expresados en un formato de complemento a 2 sino en signo y magnitud (véase la sección 4.2.5).

El bloque estimador de velocidad se encarga de efectuar la operación de substracción entre los coeficientes de promediado y los términos R y limitar a su vez la estimación de velocidad entre los rangos de +127 y -127 *pixels*/imagen.

El bloque estimador de velocidad se presenta en forma esquemática en la figura 60. Como se puede observar se hacen dos truncamientos, de manera que no es necesario hacer la conversión a complemento a 2 y la substracción utilizando 26 bits.

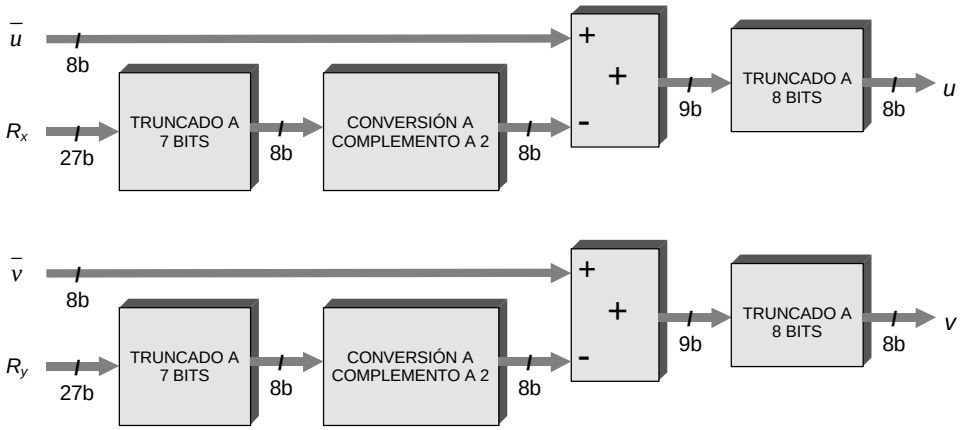


Figura 60. Bloques estimadores de velocidad.

En el apéndice B23 se muestra el módulo VHDL *vestim* correspondiente al bloque estimador de velocidad.

4.2.7. Bloques conversores

La función primordial de los bloques conversores es la de convertir los valores estimados de velocidad, los cuales están en formato complemento a 2, en un formato útil para ser utilizado en el sistema de visión, cuyas señales están diseñadas para ser mostradas en un monitor de vídeo. Para este propósito, las estimaciones "positivas y negativas" de velocidad deben ser convertidas a "valores de gris" comprendidos entre 0 y 255.

El módulo conversor aquí diseñado (figura 61) convierte las estimaciones de velocidad en un rango comprendido entre 1 (máxima velocidad hacia la izquierda o hacia abajo) y 255 (máxima velocidad hacia la derecha o hacia arriba).

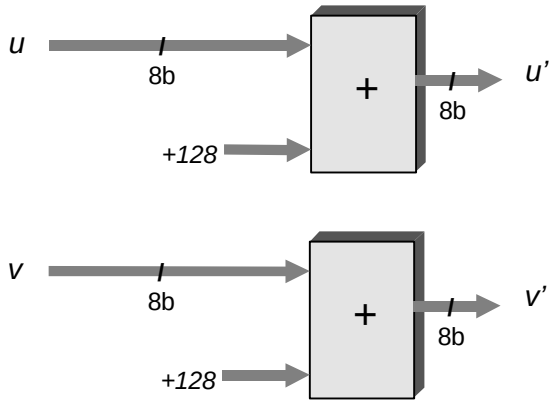


Figura 61. Bloques conversores.

En el apéndice B2 se muestra el módulo VHDL *conver* correspondiente al bloque conversor de formato.

4.2.8. Retardadores para los coeficientes de promediado

Las estimaciones de velocidad obtenidas de los estimadores de velocidad son una especie de campo vectorial, donde cada *pixel* de la imagen posee un par de números que representan la velocidad vertical y horizontal del punto. Estos campos vectoriales pueden ser considerados como dos "imágenes" donde cada "*pixel*" se corresponde con un valor de velocidad.

A partir de las estimaciones de velocidad es necesario obtener los coeficientes de promediado para lo cual se requiere la información de 9 "*pixels*" de estas "imágenes" de estimación de velocidad, tal y como se mostró en la figura 48.

Los bloques de retardo para el cálculo de los coeficientes de promediado no sólo tienen como función entregar los diversos *pixels* necesarios para el cálculo de los mismos, sino que tienen la particularidad de que también sincronizan el lazo de realimentación del circuito.

Los coeficientes de promediado son necesarios en las etapas de cálculo de los términos P, así como también son necesarios los gradientes de la imagen. Sin embargo, los gradientes centrados en un *pixel* en particular deben ir acompañados por los coeficientes de promediado del *pixel* correspondiente en la imagen anterior, lo cual significaría un retardo igual al número de *pixels* en un cuadro de la imagen. No obstante, entre la entrada de datos al bloque de

cálculo de los términos P y la salida desde la cual se toman las señales de realimentación, existe un retardo debido principalmente a los bloques para el cálculo de los términos P, D y R, y por el bloque de cálculo de los coeficientes de promediado. Por ello el retardo necesario en los bloques de retardo para los coeficientes de promediado es algo inferior al número *pixels* de un cuadro de imagen (figura 62).

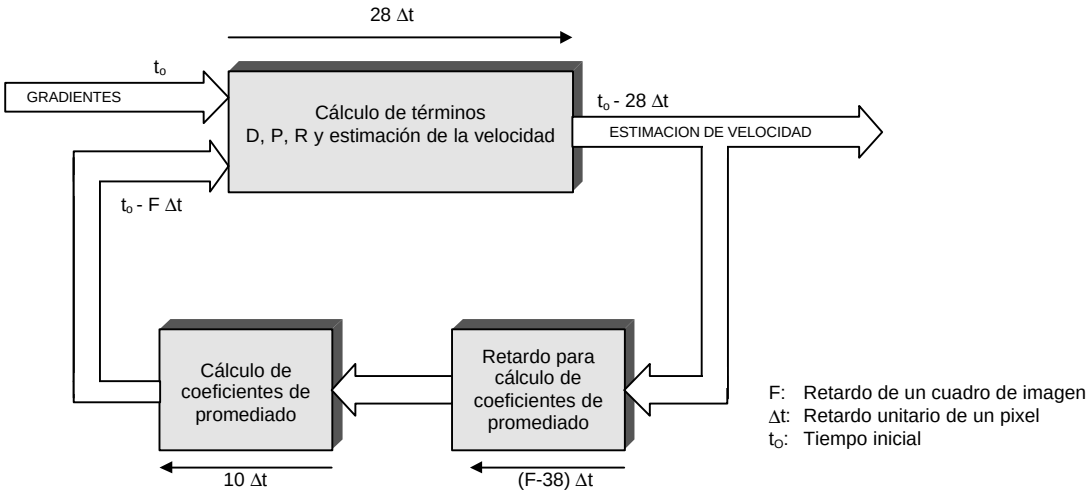


Figura 62. Diagrama simplificado del lazo de realimentación del sistema donde se observan las diferentes relaciones temporales.

Para construir un bloque de retardo que obtenga los *pixels* necesarios para calcular los coeficientes de promediado se requiere una serie de FIFOs y registros conectados en la forma mostrada en la figura 63. Hay que señalar que se requieren dos estructuras iguales, una para el cálculo de los coeficientes de promediado horizontales y otra para el cálculo de los verticales, tal y como se mostró en el diagrama de bloques general de la figura 50.

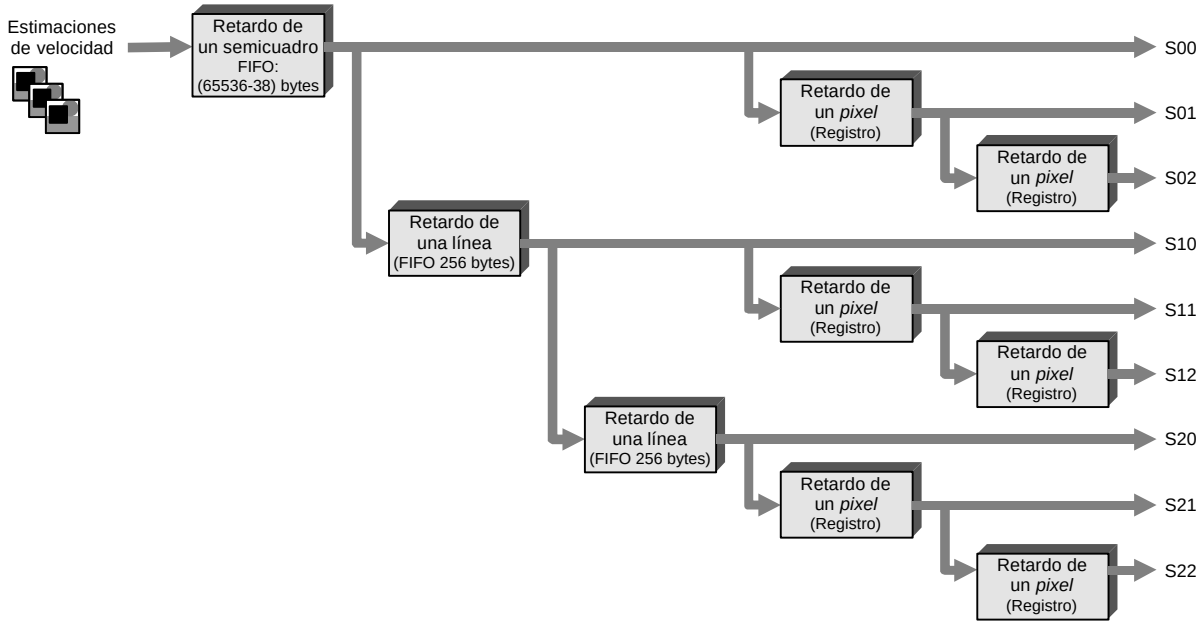


Figura 63. Conexión de FIFOs y registros que componen un bloque de retardo para el cálculo de los coeficientes de promediado.

Por las mismas razones expuestas en la sección 4.2.1 para desarrollar los retardadores para el cálculo de gradientes, es necesario utilizar una combinación de FIFOs externas a la FPGA y algunos registros internos de la FPGA para desarrollar el esquema propuesto. En la figura 64 se muestra el diagrama a bloques indicando las partes a desarrollar dentro de la FPGA y las que se desarrollan fuera.

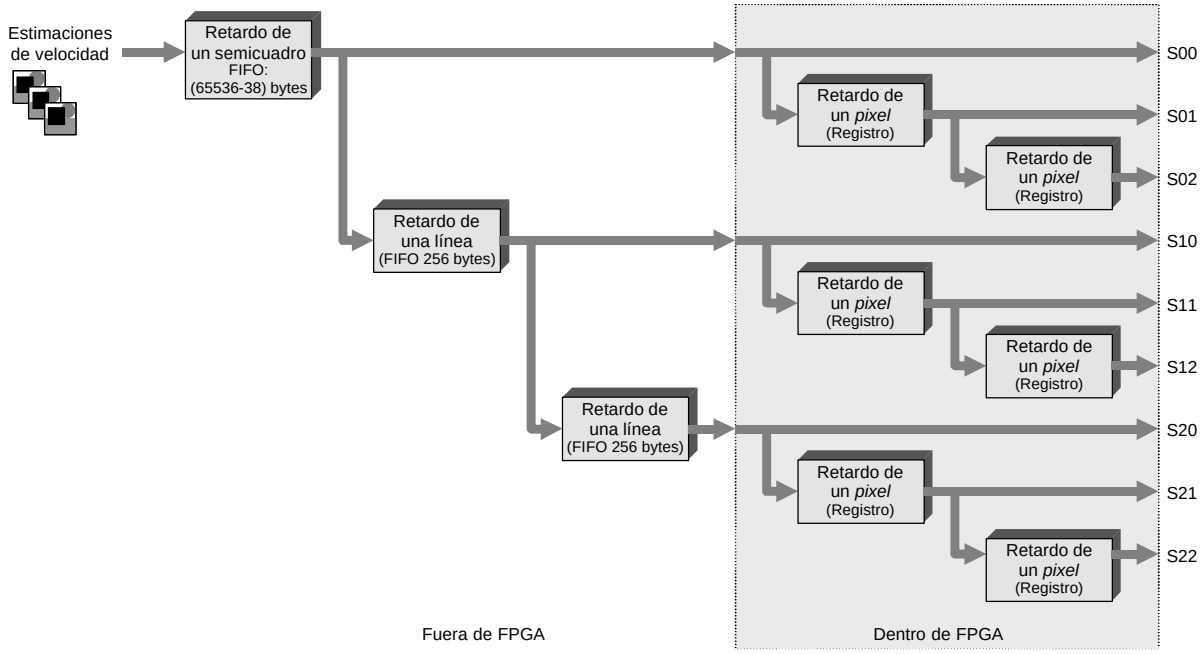
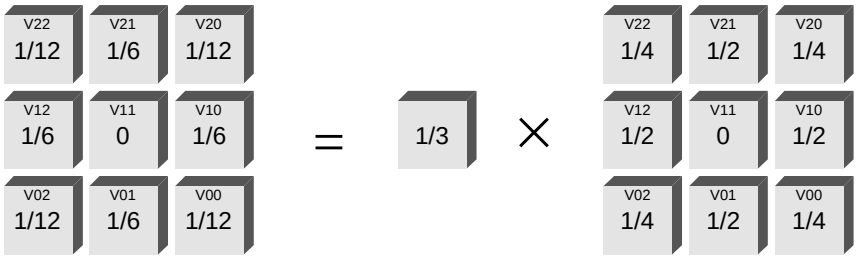


Figura 64. Separación de bloques a desarrollar dentro y fuera de la FPGA.

La descripción VHDL de los bloques de retardo para el cálculo de los coeficientes de promediado se lleva a cabo en varios módulos: *delaycp1*, *delaycp2*, *delaycp3*, y *memfifo* mostrados en los apéndices B6, B7, B8, y B17 respectivamente.

4.2.9. Bloque de cálculo de los coeficientes de promediado

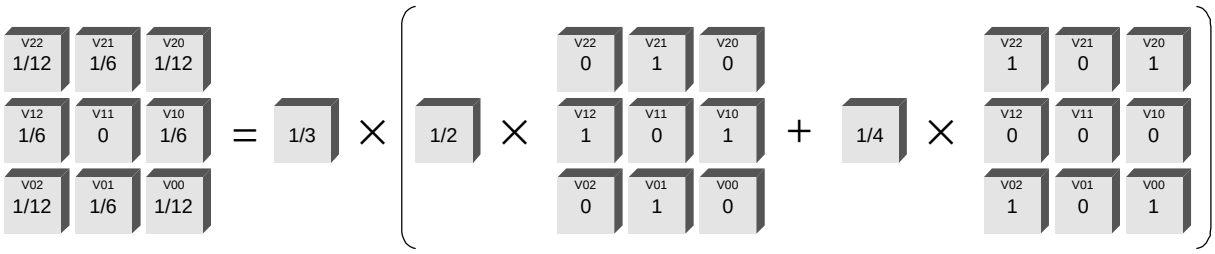
El cálculo de los coeficientes de promediado, tal y como fueron definidos en la sección 2.7.4, involucra una complicación en su realización: los valores involucrados en la máscara para su obtención no son potencias de 2. Por ello es necesario hacer una transformación para facilitar al máximo su realización por medios digitales. En este caso se ha optado por simplificar la máscara a una máscara compuesta por potencias de 2 y una multiplicación del resultado por 1/3, tal y como se ilustra en la figura 65.



$$CP = \frac{1}{3}(\frac{1}{2}(V21 + V12 + V10 + V01) + \frac{1}{4}(V22 + V20 + V02 + V00))$$

Figura 65. Máscara para la obtención de los coeficientes de promediado.

Pero aún es necesario hacer algunas consideraciones adicionales. Dado que el orden en que se efectúan las operaciones altera la precisión del resultado debido a que las operaciones se efectúan en base a aritmética entera, es necesario fijar un orden de operaciones que minimice este error. En la figura 66 se ilustran gráficamente estas operaciones.



$$CP = \frac{1}{3}(\frac{1}{2}(V21 + V12 + V10 + V01) + \frac{1}{4}(V22 + V20 + V02 + V00))$$

Figura 66. Obtención de los coeficientes de promediado tratando de minimizar los errores debidos al uso de aritmética entera.

Una vez más, al igual que para el cálculo de los términos R, en el cálculo de la división por 3 se optó por una arquitectura segmentada. En este caso, se requieren 10 secciones para obtener el resultado completo (figura 67). La realización de esta división en lenguaje VHDL es muy similar a la utilizada para el cálculo de los términos R.

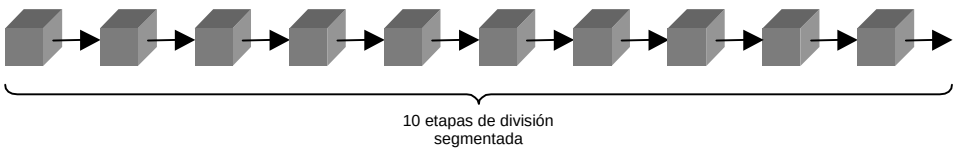


Figura 67. Multiplicación por 1/3: una división segmentada de 10 etapas

El resultado de los coeficientes de promediado evaluado de la forma señalada tiene 11 bits de precisión incluyendo el signo, pero para simplificar el *hardware* se truncará este resultado a 8 bits. Es necesario recordar que los valores altos de los coeficientes de promediado se refieren a zonas de la imagen donde aún no se llega a una convergencia de los resultados de la estimación, por ello una operación de truncamiento no afectará a los resultados.

En la figura 68 se observan los bloques que componen el módulo de cálculo de los coeficientes de promediado. Obsérvese que es necesario calcula un coeficiente para las velocidades horizontales y otro para las verticales, por lo que es necesario repetir los mismos bloques del circuito dos veces.

En los apéndices B3 y B13 se muestran las descripciones VHDL *cpromed* y *divip03* que componen la descripción general del cálculo de los coeficientes de promediado.

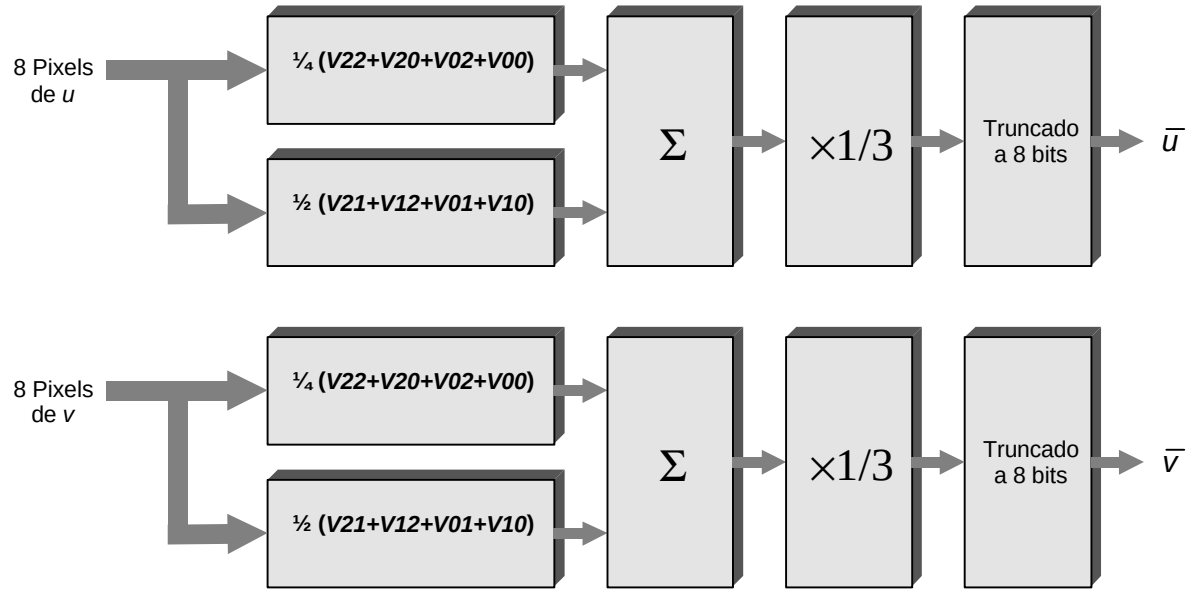


Figura 68. Módulos para el cálculo de los coeficientes de promediado.

4.2.10. Bloque de control de FIFOs

El bloque de control de FIFOs se utiliza para ajustar el tamaño de las diferentes memorias FIFO que se utilizan en el sistema. El tamaño de una memoria FIFO viene dado por la separación temporal entre el reloj de entrada de datos a la FIFO y el reloj de salida de datos. Así, por ejemplo, para una memoria FIFO de 10 bytes de capacidad se requiere que antes de que se presente un pulso en el reloj de salida de la FIFO existan 10 pulsos en el reloj de entrada de la FIFO. El funcionamiento de un módulo de control de FIFO se representa en la figura 69.

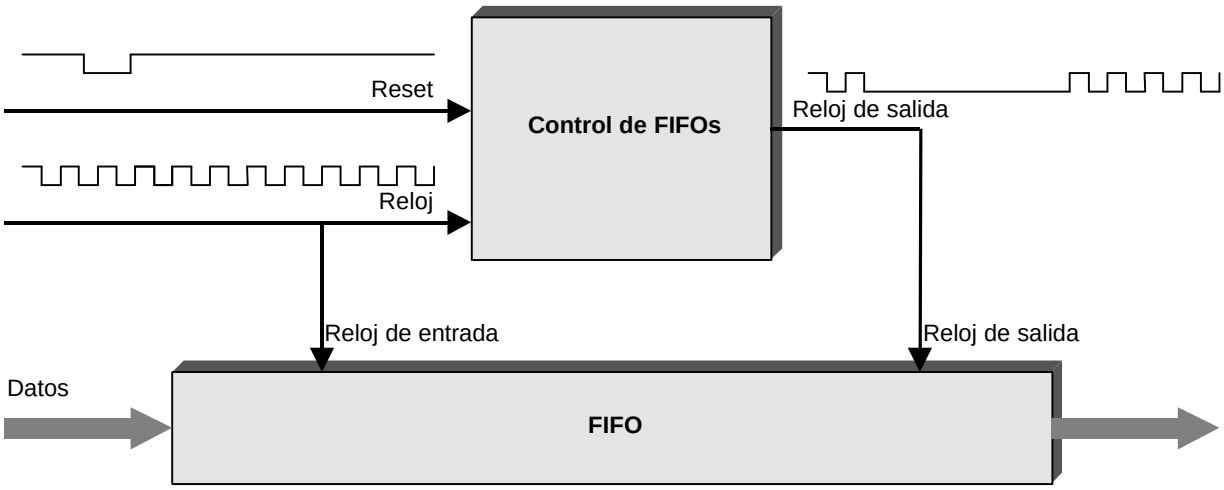


Figura 69. Control de FIFO.

Se utilizan 3 bloques de control de FIFO en el sistema para controlar las FIFOs de los retardadores asociados a los bloques de cálculo de los coeficientes de promediado. Como la longitud de cada una de estas FIFO es distinta, los controladores de FIFO también son distintos. En la descripción VHDL, mostrada en el apéndice B5, se incluye un parámetro genérico para poder configurar la longitud de la FIFO que se controla.

4.2.11. Bloque retardador para compensación

Debido a los retardos que se producen en los cálculos de los términos P, D y R, es necesario introducir una compensación de tiempo entre los valores de los coeficientes de promediado que llegan al calculador de términos P y a los estimadores de velocidad.

El retardo de compensación debe retardar la llegada de los coeficientes de promediado hasta los estimadores de velocidad, en una cantidad de pulsos de reloj igual a al tiempo de latencia en el cálculo de los términos P y R. En este sistema este retardo es de 27 pulsos de reloj.

Básicamente puede considerarse este retardo de compensación como una FIFO muy pequeña que puede ser construida con los recursos disponibles en la FPGA. En la figura 70 se muestra de manera esquemática el propósito del bloque de retardo de compensación. La descripción VHDL de este módulo se muestra en el apéndice B12.

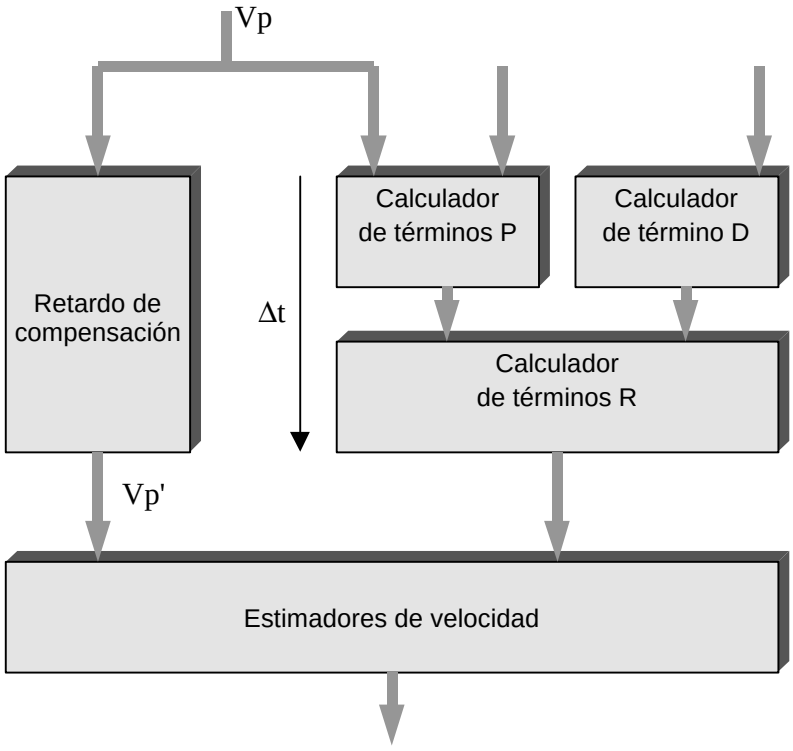


Figura 70. Retardo de compensación.

4.2.12. Bloque de sincronización

El bloque de sincronización lleva a cabo varias tareas referentes a la generación de señales que permiten sincronizar la operación de las diversas partes del sistema (figura 71). Estas pueden clasificarse dentro de los siguientes grupos:

- Generación del reloj de procesamiento principal.
- Sincronización de la toma de datos y generación de datos para el sistema y las FIFO de retardo para el cálculo de gradientes.
- Retardo de las señales de sincronismo de vídeo según latencia del sistema.
- Sincronización del *Reset* general del sistema con un sincronismo vertical.

El reloj de procesamiento principal comprende dos señales, el reloj de procesamiento propiamente dicho y su negado. Estas dos señales se utilizarán en todos los circuitos del sistema. Dado que todo el procesamiento interno se efectúa sólo en los casos en que existen datos válidos, el reloj de procesamiento principal es una señal fruto de la mezcla de la señal de reloj (CLK) y la señal de dato válido (DV) procedentes de la cámara de vídeo.

En cuanto a la sincronización de los datos, este bloque toma los datos en el flanco positivo del reloj procedente de la cámara y genera dos buses de datos: uno para las FIFOs, válido en el frente positivo del reloj de procesamiento y otro para el sistema de procesamiento, válido en el frente negativo de la señal de reloj. De esta manera, los datos que entran al sistema de procesamiento, tanto el que llega directo como el que llega procedente de las FIFO, están sincronizados en el mismo flanco de reloj.

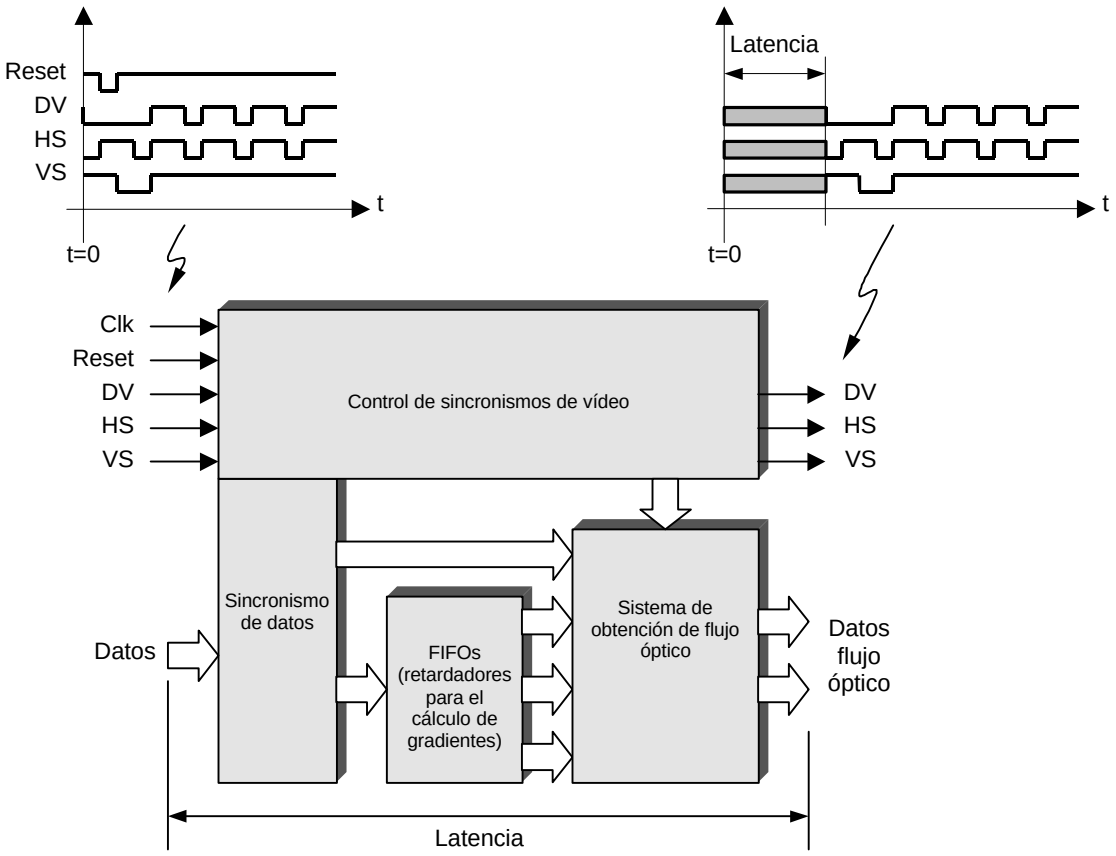


Figura 71. Generación de señales de reloj y sincronismo.

También al bloque de sincronización le corresponde la función de generar las señales de sincronización de vídeo que se requieren a la salida del sistema. Las señales de sincronización de vídeo son básicamente las señales de sincronismo horizontal, vertical y datos válidos. Dado que de estas señales depende el punto de inicio de las imágenes, es necesario que estén debidamente sincronizadas con los datos a la salida. Ahora bien, aunque en un sistema de arquitectura segmentada se supone que los datos fluyen a la misma velocidad tanto a la entrada como a la salida, puede existir cierta *latencia* que pudiera ser considerada el equivalente digital del desfasaje analógico. Por ello, dado que los datos sufren esa *latencia*, es necesario que los pulsos de sincronismo sufran la misma *latencia* (figura 71).

El manejo de la señal de *Reset* del sistema es sincronizado por este bloque con un sincronismo vertical, de esta manera los efectos de una inicialización del sistema sobre las imágenes de vídeo pueden ser minimizados. Adicionalmente, las memorias FIFO requieren una señal de *Reset* de características especiales, en sincronismo con las señales de reloj [CYPR00] [IDT00], la cual también es generada por este bloque.

La descripción VHDL del bloque de sincronización comprende dos módulos VHDL. Un bloque para generar la latencia en cada una de las señales de sincronismo de vídeo, mostrado en el apéndice B11. El otro módulo es el bloque de sincronización propiamente dicho, e incluye el uso del módulo de generación de latencia. Este bloque es mostrado en el apéndice B21.

4.2.13. FPGA para estimación del flujo óptico

Habiendo definido cada uno de los bloques del sistema para la determinación del flujo sólo resta establecer los módulos que componen la FPGA del sistema. Una vez más toda la descripción de conexión de todos los módulos VHDL se realiza utilizando el mismo lenguaje de descripción de *hardware*, el VHDL. Sin embargo, la gran cantidad de módulos que lo componen hacen un poco difícil la visualización del conjunto, por ello en la figura 72 se presenta un esquema de los módulos que componen la FPGA. El módulo VHDL recibe el nombre de *fpgablk* y está descrito en el apéndice B15.

En la FPGA están incluidos todos los módulos a excepción de las FIFOs más grandes. Dado que se pretende sintetizar en una FPGA, este módulo y todos los que él contiene deben ser enteramente sintetizables. Se puede considerar que está constituido con 4 grupos de módulos:

- Los módulos cuya función es la de control y sincronización
- Los módulos de cálculo de la ecuación de flujo óptico.
- Los módulos de cálculo de los coeficientes de promediado.
- Los módulos conversores de señales al formato de salida.

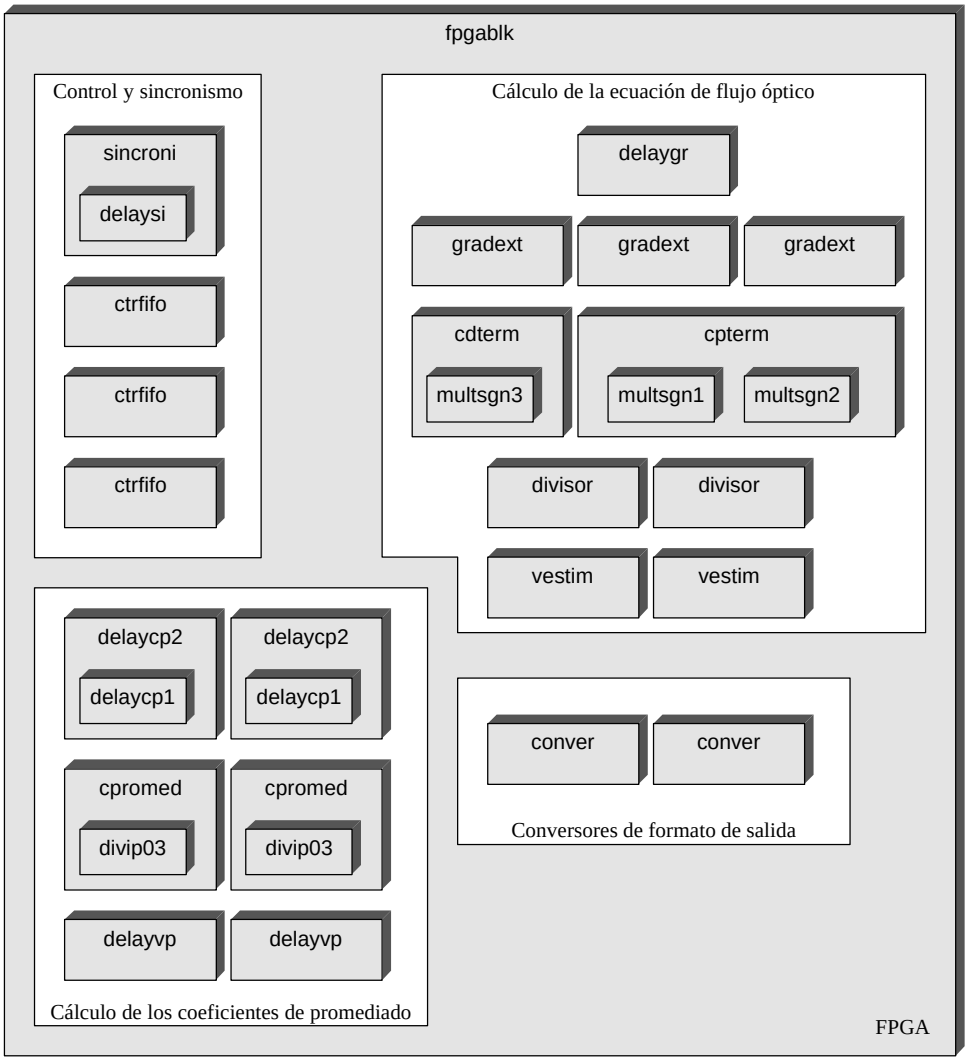


Figura 72. Módulos contenidos en la FPGA.

4.2.14. Sistema completo y banco de pruebas

Por último, es necesario un módulo que concentre tanto la FPGA como las FIFOs externas al banco de trabajo con lo cual se harán las pruebas del sistema. Este módulo se ha denominado *system* y su estructura se presenta en la figura 73 y la descripción en lenguage VHDL se presenta en el apéndice B22.

Dado que este bloque contiene el banco de pruebas, la descripción VHDL del mismo será no sintetizable. Para llevar a cabo las pruebas, será necesario tener imágenes de prueba en el formato *hex* que fue propuesto en el capítulo 3.

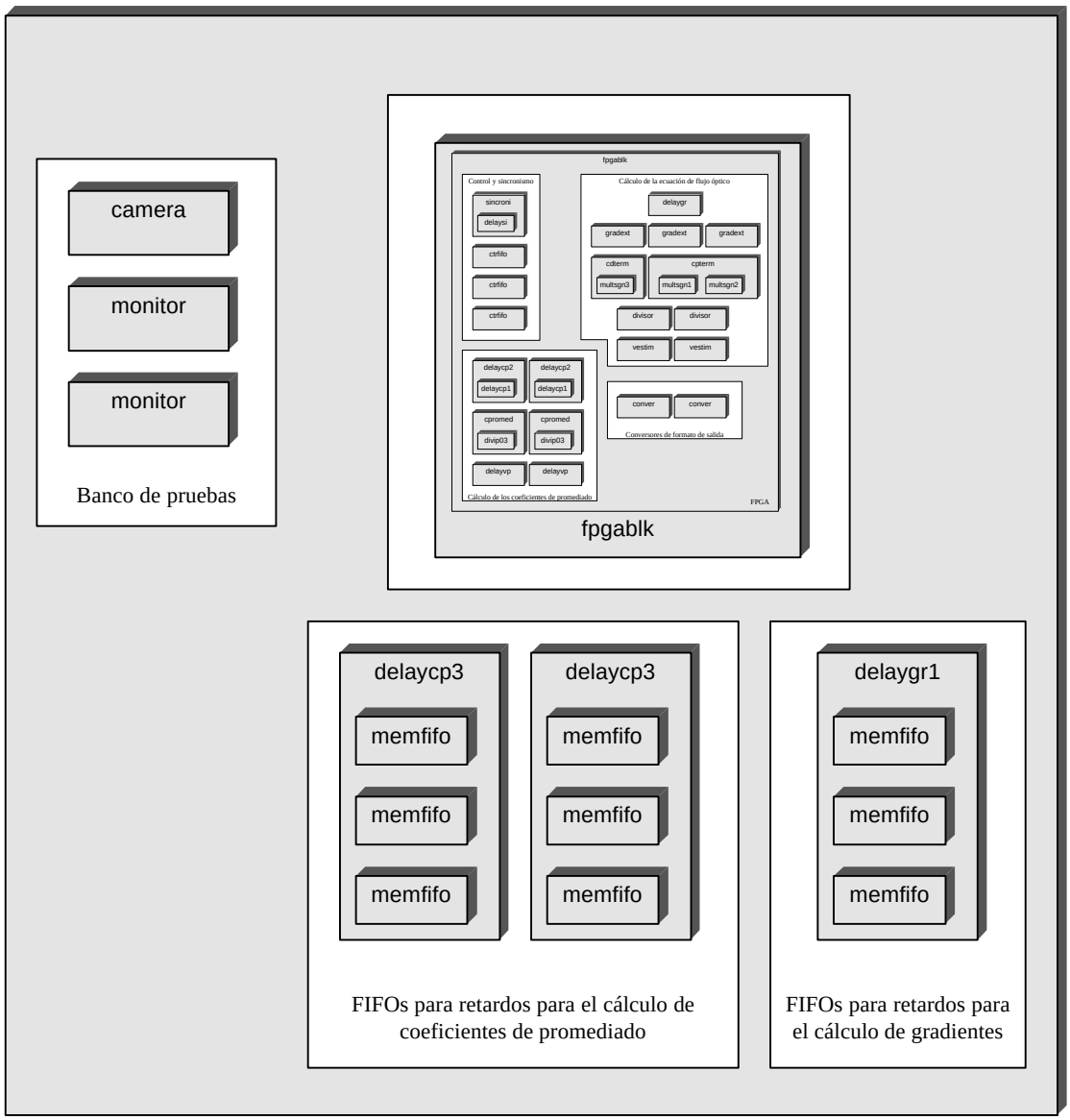


Figura 73. Sistema para la determinación del flujo óptico y banco de pruebas.

5

Capítulo 5

Simulaciones y resultados

... cada vez que un salvaje rastrea la caza
emplea una minuciosidad de observación
y una precisión de razonamiento inductivo y deductivo que,
aplicado a otros asuntos,
le darían una reputación de hombre de ciencia...

Thomas H. Huxley
Mr. Darwin's Critics, 1871

5.1. Organización de las pruebas del sistema

Para llevar a cabo las pruebas del sistema es necesario seguir una serie de etapas, que en este caso se han dividido en tres, las pruebas del algoritmo, las pruebas de los bloques constitutivos y las pruebas generales del sistema.

En las pruebas del algoritmo el objetivo es probar el comportamiento del algoritmo seleccionado para la determinación del flujo óptico.

La finalidad de las pruebas de los bloques constitutivos es la de probar cada uno de los bloques diseñados con lenguaje VHDL, con el propósito de analizar su comportamiento y de efectuar ajustes si se encontrara algún problema.

Las pruebas generales del sistema buscan simular el funcionamiento general del sistema completo, determinando su viabilidad.

5.2. Pruebas del algoritmo

Una vez seleccionado el algoritmo básico sobre el cual se desarrollaría la arquitectura, se llevaron a cabo las primeras pruebas utilizando el Analizador de Algoritmos explicado en la sección 3.4. Para estas pruebas se utilizaron secuencias limitadas de imágenes sintéticas y reales, pero utilizando las ecuaciones iterativas de flujo óptico (2.26) en múltiples iteraciones entre el mismo conjunto de imágenes. Sin embargo no se aprovechó la similitud de imágenes en la secuencia que, tal y como se señaló en la sección 4.1.1, representa una solución mucho más acertada para la realización de un sistema electrónico de procesamiento en tiempo real. A pesar de ello, el análisis del algoritmo básico fue de utilidad para ver el potencial del algoritmo.

Las primeras secuencias de imágenes utilizadas en el Analizador de Algoritmos fueron imágenes sintéticas, como *el punto móvil en cuadriculado fijo*, cuya secuencia de 3 imágenes se muestra en la figura 74.

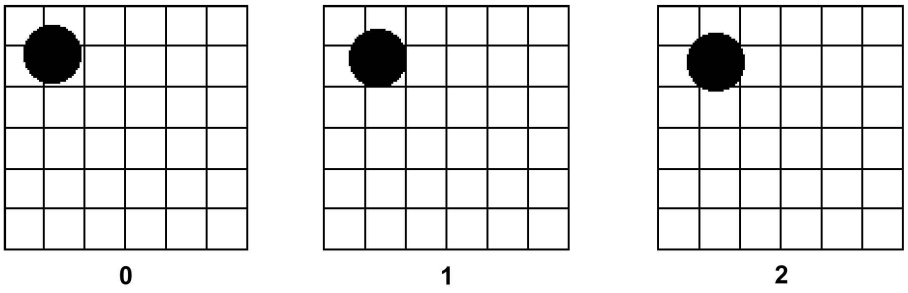


Figura 74. Secuencia de imágenes sintéticas: *punto móvil en cuadriculado fijo.*

La citada secuencia de imágenes genera después de cinco iteraciones el campo vectorial mostrado en la figura 75 y que se representa con pequeñas líneas grises. El Analizador de Algoritmo también genera una ayuda al representar un vector resultante sobre el objeto analizado. Obsérvese que en este caso, el hecho de que el punto tiene un color único, impide el cálculo de vectores dentro de ciertas áreas del punto. Esto es debido al problema de apertura que se presenta en aquellas áreas de los objetos sin gradiente de color.

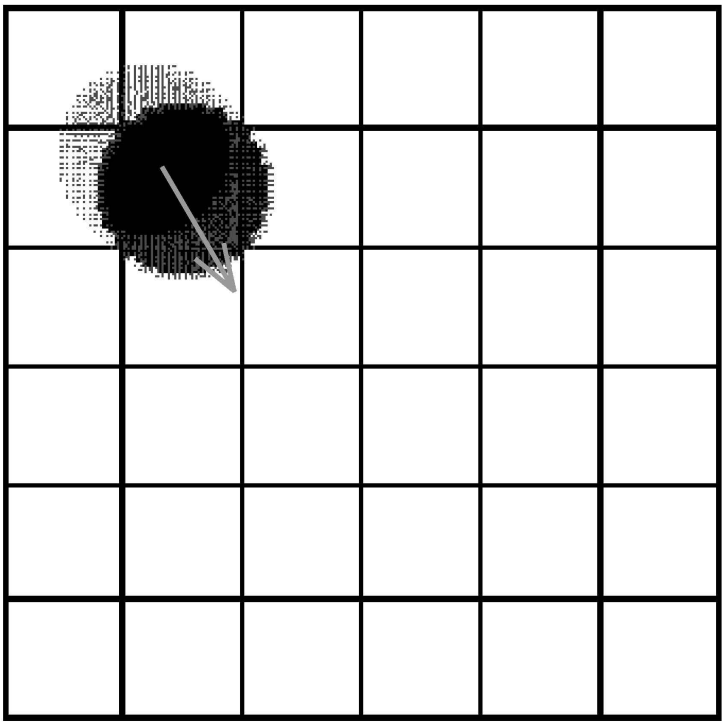


Figura 75. Campo vectorial resultante del análisis de la secuencia sintética *punto móvil en cuadriculado fijo.*

5.3. Pruebas de los bloques

En las pruebas de los bloques se busca afinar el funcionamiento de cada uno de los bloques constitutivos del sistema. Específicamente son dos los puntos en que es necesario hacer hincapié en estas pruebas. En primer lugar, verificar que cada bloque efectúa las operaciones que se tienen previstas. Y en segundo lugar, que su respuesta en tiempo esté dentro de los parámetros fijados, es decir, que pueda funcionar a la velocidad impuesta por el reloj del sistema.

Sin embargo, para llevar a cabo estas pruebas es necesario establecer el tipo de componentes a utilizar y el correspondiente programa de síntesis. Para esta Tesis se decidió utilizar FPGAs de la firma *Altera*, dada la experiencia anterior con estos dispositivos y a la disponibilidad del programa de desarrollo MAX+Plus II, bastante más intuitivo y fácil de utilizar.

El programa MAX+Plus II permite sintetizar circuitos electrónicos para ser programados en FPGAs de *Altera* a partir de la descripción VHDL. Además permite la simulación del funcionamiento de los mismos. La mayor limitación de este programa es que sólo permite el uso de código VHDL que pueda ser sintetizado en FPGAs. Por ello, no es posible simular el funcionamiento de un sistema que incluya dispositivos externos como puede ser el caso de las FIFOs utilizadas en el sistema que trata esta Tesis, o utilizar bancos de prueba que lean información desde ficheros tal y como se propuso en la sección 3.3.

De lo anterior se desprende que para poder simular el funcionamiento de todo el sistema no podemos utilizar el MAX+Plus II. Se requiere el uso de un programa de simulación VHDL genérico tal y como el *ModelSim*. Por ello, se opta por utilizar los dos. El programa MAX+Plus II se utilizará para las simulaciones de cada uno de los elementos que serán sintetizados dentro de una FPGA, y permitirá la estimación de tiempos de retardo en el funcionamiento de cada uno de ellos. Por otro lado el programa *ModelSim* se utilizará para la simulación general del sistema.

La utilización de dos interpretadores distintos de VHDL trae consigo una serie de problemas, debidos a la manera distinta que tiene cada uno de los dos programas de interpretar el código VHDL. En realidad debería ser igualmente válido el código VHDL para los dos interpretadores, pero aún existen diferencias. Por ello fue necesario que cada uno de los bloques desarrollados fuese comprobado con los dos interpretadores y verificar que su funcionamiento fuese igual en ambos.

En las secciones siguientes se muestran los resultados finales de la simulación de cada uno de los bloques con el MAX+Plus II. Obviamente, cada uno de ellos requirió una serie de pasos previos que no se exponen aquí, pero que buscaban ajustar a que la misma descripción VHDL produjera resultados iguales en cada uno de los interpretadores a utilizar, que estos resultados fueran los requeridos, y que los retardos temporales se ajustaran a los requeridos.

5.3.1. Prueba de bloque de cálculo de gradiente

Tal y como se explicó en la sección 4.2.2, el bloque de cálculo de gradiente realiza una operación matemática con 8 variables de entrada y emite un resultado. La prueba de este bloque se realiza introduciendo 7 vectores de prueba de 8 variables sincronizadas con un reloj. Las vectores de prueba y sus respectivos resultados se muestran en la tabla 6.

Tabla 6. Vectores de prueba de bloque de cálculo de gradiente.

Valores de entrada								Valor de salida	
A0 (hex)	**A1 (hex)**	**A2 (hex)**	**A3 (hex)**	**A4 (hex)**	**A5 (hex)**	**A6 (hex)**	**A7 (hex)**	**e (dec)**	**e (hex)**
01	02	03	04	05	06	07	20	-7	1F9
10	20	10	30	10	40	10	50	-40	1D8
00	30	40	50	60	70	80	90	-24	1E8
50	60	70	80	90	A0	B0	C0	-16	1F0
11	12	13	14	13	12	11	10	0	000
21	22	23	22	21	20	19	08	4	004
FF	01	FF	01	FF	01	FF	01	254	0FE

Utilizando los mencionados vectores de prueba, se efectúan las pruebas del bloque de cálculo de gradiente, obteniéndose los resultados mostrados en la figura 76, que corresponden a la realización del bloque en una FPGA de *Altera* de la serie 10K-3. Se observan unos resultados coincidentes con los esperados para los valores de entrada especificados en los vectores de prueba. En cuanto a la respuesta temporal del bloque, ésta se muestra en la figura 77, donde se observa que la respuesta del bloque es estable después de unos 20 ns del flanco de subida de la señal de reloj.

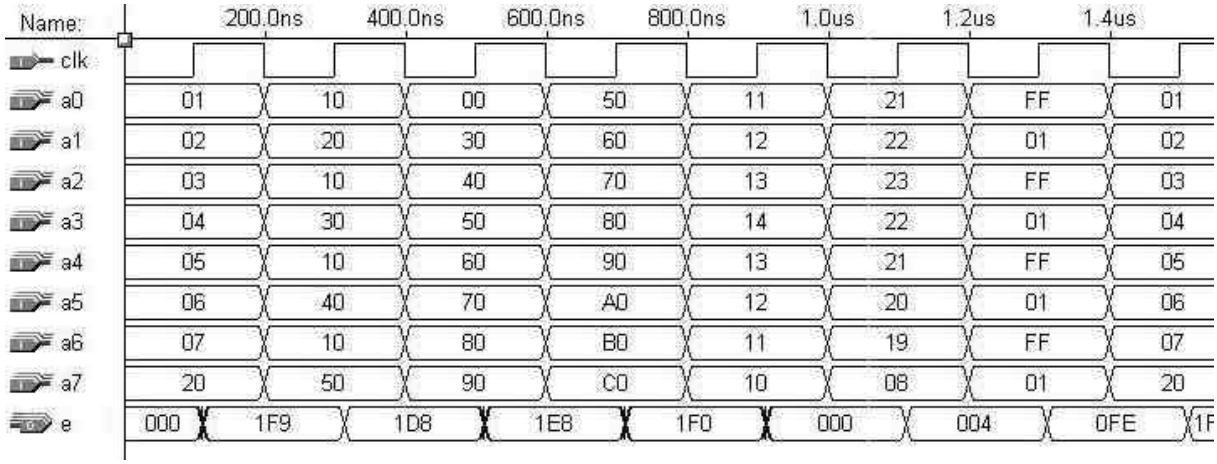


Figura 76. Resultados de la simulación con los vectores de prueba sobre el bloque de cálculo de gradiente.

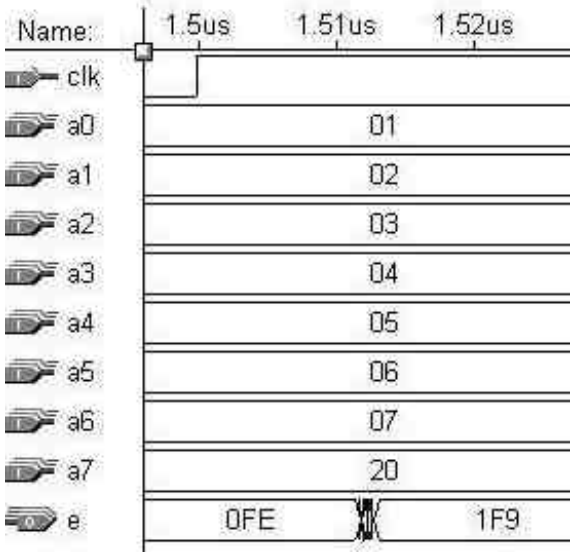


Figura 77. Detalle de la respuesta temporal del bloque de cálculo de gradiente.

5.3.2. Prueba de bloque de cálculo de término D

En la sección 4.2.3 se presentó el bloque de cálculo de término D, que consiste en una operación matemática sobre dos variables. Por ello, la prueba de este bloque se realiza

introduciendo 7 vectores de prueba de 2 variables sincronizadas con un reloj. Los vectores de prueba y sus respectivos resultados se muestran en la tabla 7.

Tabla 7. Vectores de prueba de bloque de cálculo de término D.

Valores de entrada				Valores de salida	
Ex		Ey		D	
(hex)	(dec)	(hex)	(dec)	(hex)	(dec)
101	-255	101	-255	7FC02	130050
3F9	-7	000		00031	49
000	0	1D8	-40	00640	1600
1E8	-24	1F0	-16	00340	832
144	324	080	128	29A10	170512
177	375	122	-222	2E5D5	189909
000	0	000	0	1	1

Realizando la simulación de funcionamiento del bloque de cálculo de término D en MAX+Plus II con los vectores de prueba señalados se obtienen los resultados mostrados en la figura 78. Estos resultados son coincidentes con los valores esperados para dichos vectores. La respuesta temporal más detallada se muestra en la figura 79, donde se observa una respuesta estable a partir de 20 ns después del flanco de subida de la señal de reloj.

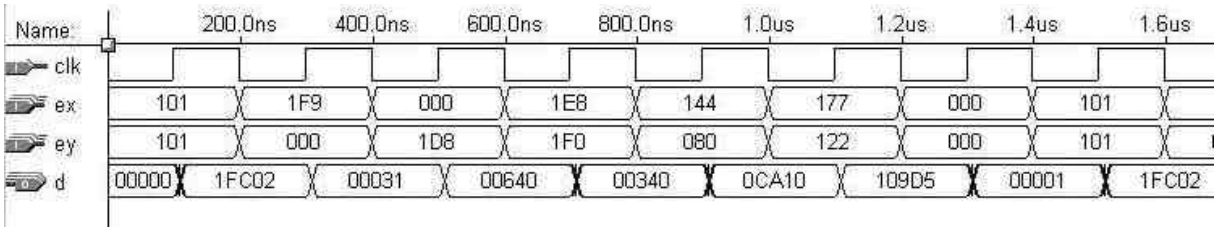


Figura 78. Resultados de la simulación con los vectores de prueba sobre el bloque de cálculo de término D.

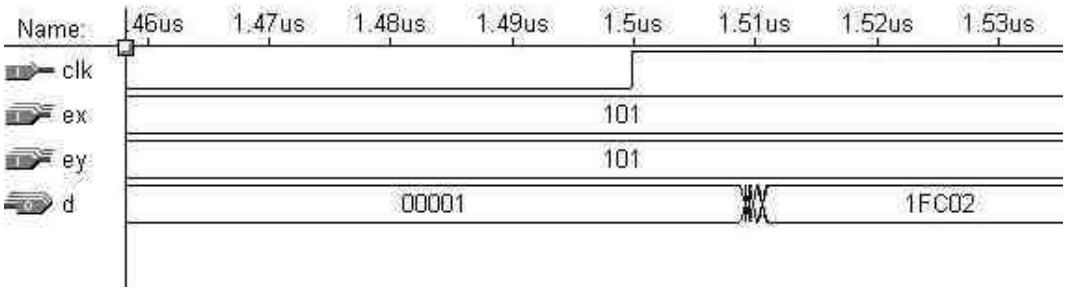


Figura 79. Detalle de la respuesta temporal del bloque de cálculo de término D.

5.3.3. Prueba de bloque de cálculo de términos P

Tal y como se mostró en la sección 4.2.4, el cálculo de término P, involucra 5 variables de entrada (los gradientes y los coeficientes de promediado) y dos variables de salida (los términos Px y Py). Para probar este bloque se prepara un conjunto de 8 vectores de prueba de 5 variables sincronizadas con un reloj. Estos vectores de prueba y sus respectivos resultados se muestran en la tabla 8.

Tabla 8. Vectores de prueba de bloque de cálculo de términos P.

Valores de entrada					Valores de salida	
Ex (hex)	Ey (hex)	Et (hex)	*u* (vpx) (hex)	*v* (vpy) (hex)	Px (hex)	Py (hex)
1F9	000	000	1F9	000	3FFFEA9	0000000
000	1D8	000	000	1D8	0000000	3FF0600
1E8	1F6	000	1F9	1D8	3FFCAC0	3FFE9D0
0E7	0F0	000	1E8	1F6	3E3FFC8	3E2E880
0FF	0FF	085	1E8	1F6	3DEC859	3DEC859
101	101	101	101	101	204FA03	204FA03
00A	014	01E	028	032	00037DC	0006FB8
0FF	0FF	0FF	0FF	0FF	1FB03FF	1FB03FF

Con los vectores de prueba presentados en la tabla 8 se efectúan las pruebas del bloque de cálculo de términos P en el simulador MAX+Plus II con un dispositivo FPGA de la serie 10K-3 de *Altera*. Los resultados, mostrados en la figura 80, son acordes a lo indicado en la misma tabla 8. En cuanto a la respuesta temporal del bloque, ésta se muestra en la figura 81, en la misma se han agregado dos variables internas denominadas *pxb* y *pyb* para poder visualizar los tiempos de cálculo del bloque antes del registro de salida. Se observa cómo las señales son inestables por un tiempo superior a 100 ns. Esto establece una cota superior para la velocidad máxima que puede alcanzar el sistema. Con el registro de salida se obtiene una respuesta estable para la siguiente etapa después de unos 20 ns del flanco de subida de la señal de reloj.

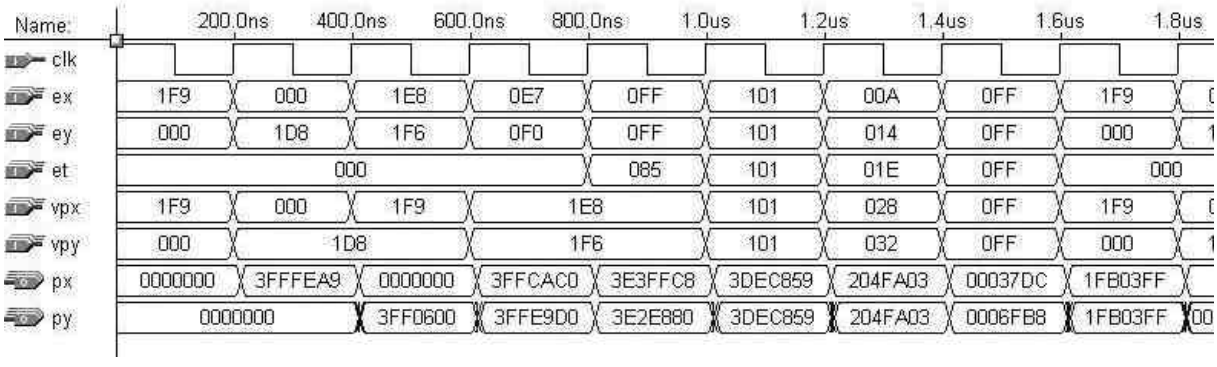


Figura 80. Resultados de la simulación con los vectores de prueba sobre el bloque de cálculo de términos P.

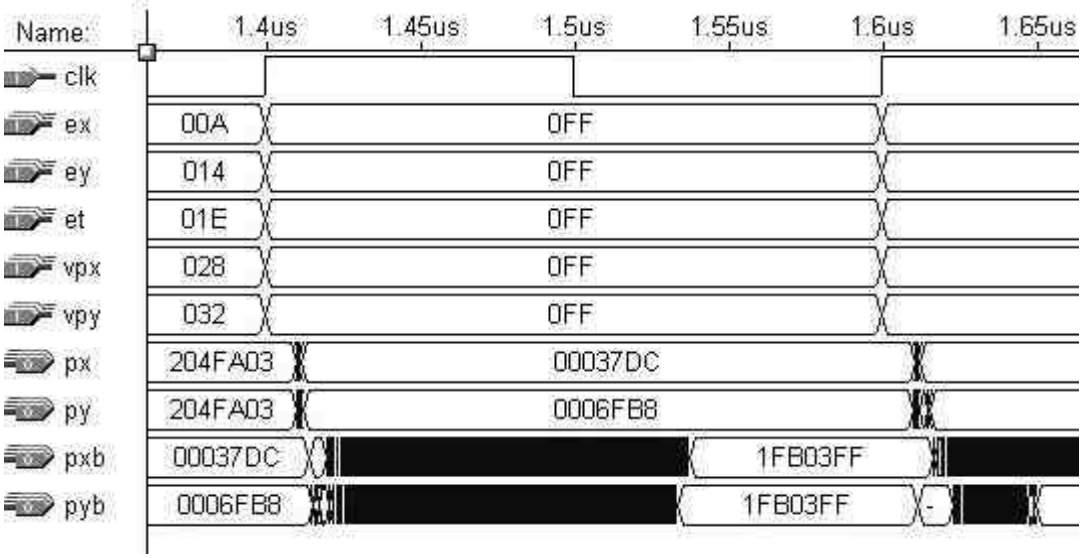


Figura 81. Detalle de la respuesta temporal del bloque de cálculo de términos P.

5.3.4. Prueba de bloque de cálculo de términos R

El cálculo de los términos R involucra una división segmentada, tal y como se refirió en la sección 4.2.5. En esta división hay que tener en cuenta que tanto el numerador como el denominador tienen tamaños limitados. El numerador puede ser positivo o negativo, pero el denominador es siempre positivo. El resultado puede ser por tanto positivo como negativo. Sin embargo, la notación seleccionada para los negativos es diferente. En el caso del numerador, la notación es complemento a 2, pero en el caso del resultado la notación es signo-magnitud por razones explicadas con anterioridad.

Con esto en mente se crearon una serie de vectores de prueba del bloque de cálculo de términos R. El número de vectores que se utilizaron fue mayor a 20, sin embargo aquí, en la tabla 9, se presentan los 7 más significativos y sus respectivos resultados esperados. En la figura 81 se muestran los resultados de la simulación del bloque de cálculo de términos R. Para entender la figura, es necesario recordar que por ser una arquitectura segmentada, existe una latencia entre la entrada y la salida de datos. Por ello, en la parte superior de la figura se representa el período de tiempo en que los vectores se introducen al bloque, y en la parte inferior, el período de tiempo en que aparecen los resultados relativos a esos mismos vectores. En la misma figura se observa que los datos son válidos en el flanco de bajada de la señal y que las señales son estables unos 50 ns después del flanco de subida de la señal.

Tabla 9. Vectores de prueba de bloque de cálculo de términos R.

Valores de entrada				Valores de salida	
P		D		R	
(hex)*	(dec)	(hex)	(dec)	(hex)**	(dec)
0FED260	16700000	0F618	63000	0000109	265
0000000	0	032C8	13000	0	0
3FFFFFF	-1	00001	1	2000001	-1
00056D9	22233	10930	67888	0	0
0FFFFFF	16777215	0FFFF	65535	100	256
3000001	-16777215	0FFFF	65535	2000100	-256
00000C8	200	00000	0	1FFFFFF	33554431***

* En formato complemento a 2
** En formato signo-mantisa (signo el bit más significativo)
*** Sobre-rango (entero más grande posible)

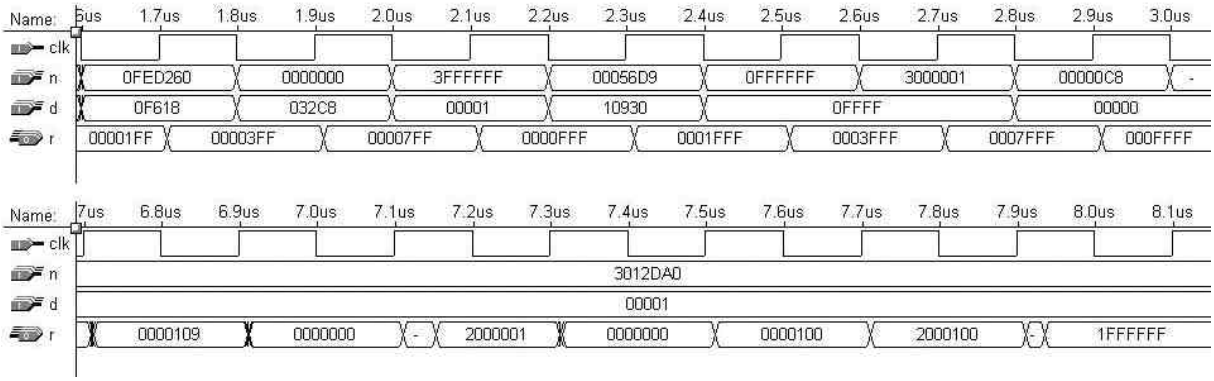


Figura 82. Resultados de la simulación con los vectores de prueba sobre el bloque de cálculo de términos R.

5.3.5. Prueba del bloque estimador de velocidad

Tal y como fue reseñado en la sección 4.2.6, el bloque estimador de velocidad, que en apariencia debe efectuar una operación matemática más o menos sencilla, consta de varias etapas donde se realizan truncamientos y conversiones de formato, lo cual hace que deban incluirse ciertas operaciones que ejerciten las diferentes partes del bloque. Para las pruebas se ha seleccionado un conjunto de 8 vectores mostrados en la tabla 10 junto con los resultados esperados.

Utilizando estos vectores de prueba se obtienen los resultados que se muestran en la figura 83 y que coinciden con los esperados que se mostraron en la tabla 10. Al igual que otros bloques que componen el sistema, el bloque de estimación de la velocidad tiene un registro de salida, lo cual hace que el dato aparezca a la salida unos 20 ns (figura 84) después del flanco de subida de la señal de reloj.

Tabla 10. Vectores de prueba del bloque de estimación de velocidad.

Valores de entrada				Valores de salida	
Vp		R		V	
(hex)*	(dec)	(hex)**	(dec)	(hex)**	(dec)
000	0	0000020	32	E0	-32
020	32	0000000	13000	20	32
1DF	-33	0000021	33	BE	-42
1E0	-32	2000020	-32	00	0
1FF	-1	0000020	32	DF	-33
1FE	-2	1FFFFFF	33554431	81	-127
1FF	-1	3FFFFFF	-33554431	7E	126

* En formato complemento a 2
** En formato signo-magnitud (signo el bit más significativo)

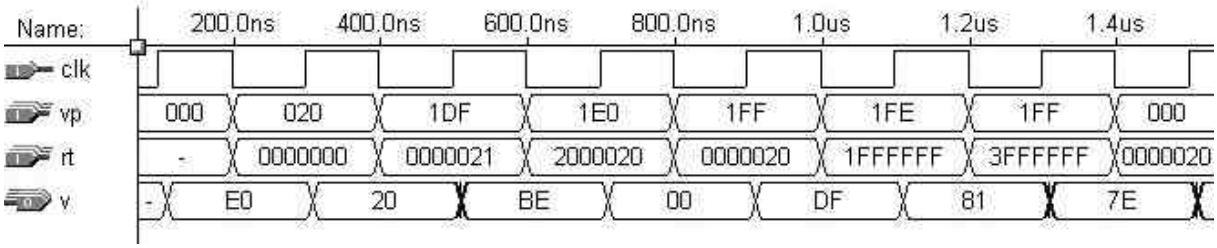


Figura 83. Resultados de la simulación con los vectores de prueba sobre el bloque de cálculo de términos R.

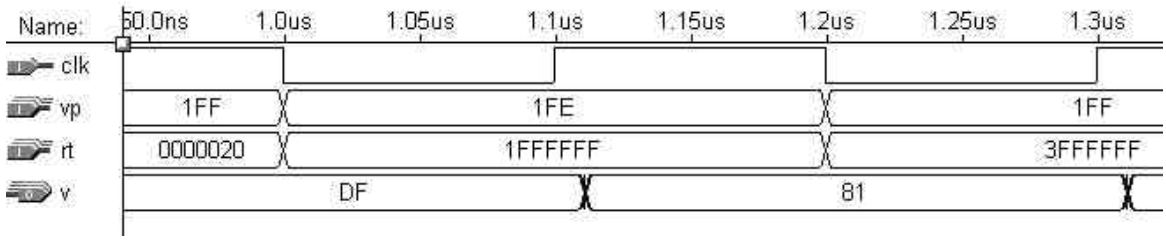


Figura 84. Detalle de la respuesta temporal del bloque de cálculo de términos R.

5.3.6. Prueba del bloque conversor

La función del bloque conversor es relativamente simple, tal y como se analizó en la sección 4.2.7. Su función consiste en desplazar los valores de velocidad de manera que los valores positivos y negativos de velocidad se representen como valores positivos entre 0 y 255 (0 y FFh), correspondiéndole al cero la posición central, es decir, 128 (80h). Para probar este bloque se seleccionó un conjunto de 5 vectores, representados en la tabla 11 con sus respectivos resultados esperados.

Tabla 11. Vectores de prueba del bloque conversor.

Valores de entrada		Valores de salida	
Vi		Vc	
(hex)*	(dec)	(hex)	(dec)
00	0	80	128
20	32	A0	160
7F	127	FF	255
A0	-96	20	32
FF	-1	7F	127

* En formato complemento a 2

Al igual que los demás bloques del sistema, el bloque conversor tiene un registro para sincronizar la salida con la señal de reloj. Los resultados de la simulación de funcionamiento de este bloque se muestran en la figura 85 y concuerdan con los resultados esperados de la tabla 11. En la figura 86 se muestra un detalle de los tiempos de respuesta del bloque. La salida se presenta después de 10 ns del flanco de subida del reloj.

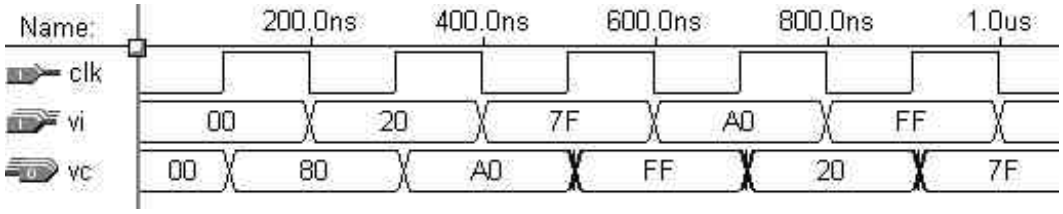


Figura 85. Resultados de la simulación con los vectores de prueba sobre el bloque conversor.

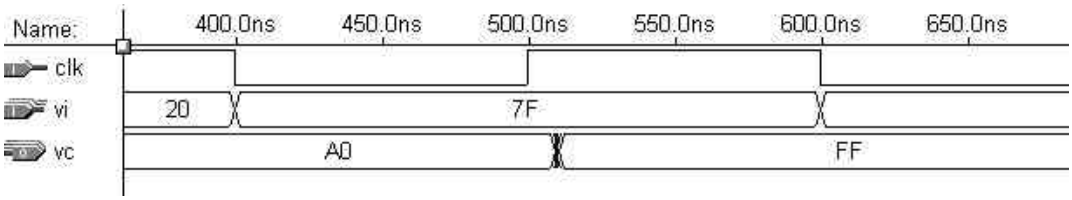


Figura 86. Detalle de la respuesta temporal del bloque conversor.

5.3.7. Prueba del bloque de cálculo de los coeficientes de promediado

En la sección 4.2.9 se estudió el funcionamiento del bloque de cálculo de los coeficientes de promediado y básicamente es la aplicación de un cálculo del tipo suma de productos. La característica más importante de estos productos es la de tener un multiplicador común difícil de llevar a cabo por *hardware* y que requiere una arquitectura segmentada para su cálculo. Para la simulación del bloque se ha seleccionado varios conjuntos de vectores de prueba, de

los cuales aquí se muestra uno de ellos que contiene 5 vectores, tal y como se puede ver en la tabla 12.

Tabla 12. Vectores de prueba de bloque de cálculo de los coeficientes de promediado.

Valores de entrada								Valor de salida	
V00 (hex)	**V01 (hex)**	**V02 (hex)**	**V10 (hex)**	**V12 (hex)**	**V20 (hex)**	**V21 (hex)**	**V22 (hex)**	**CP (dec)**	**CP (hex)**
FF	FF	FF	FF	FF	FF	FF	FF	-1	1FF
00	00	00	00	00	00	00	00	0	000
40	50	60	70	80	90	A0	B0	-8	1F8
81	92	A3	B4	45	56	67	78	-3	1FD
09	11	13	15	01	03	05	07	10	00A

Para este bloque, al ser implementado con una arquitectura segmentada, hay que recordar que el resultado se presentara con una latencia de 10 períodos. En la figura 87 se muestran las señales obtenidas de la simulación del bloque, y en ellas se observa que se cumple el resultado esperado de la aplicación de los vectores de prueba antes señalados.

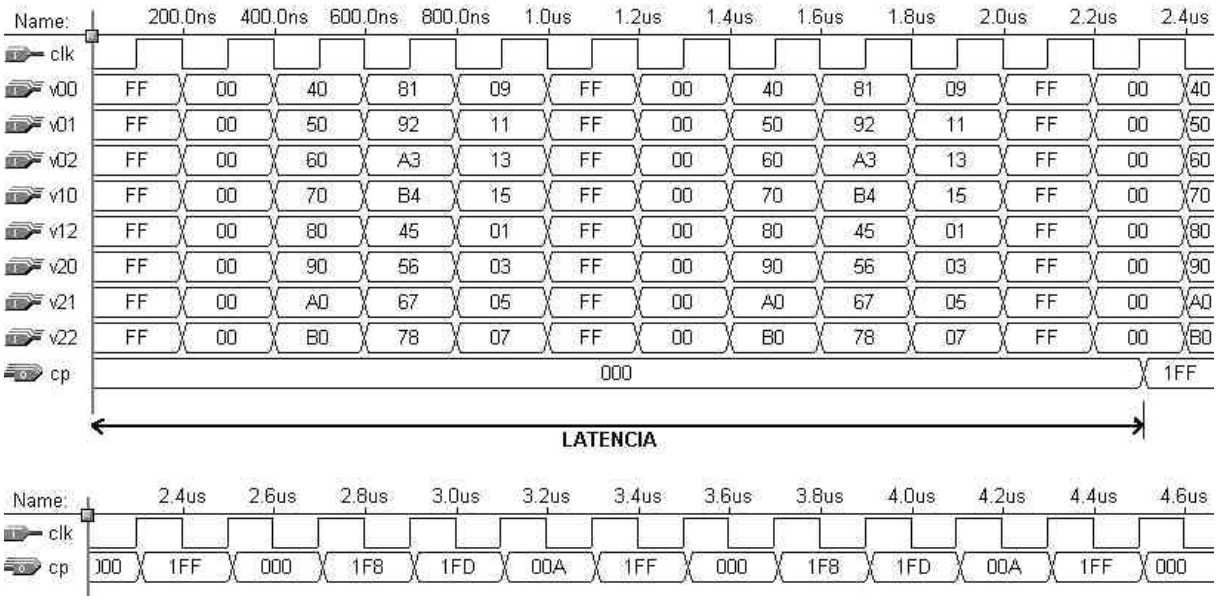


Figura 87. Resultados de la simulación con los vectores de prueba sobre el bloque de cálculo de coeficientes de promediado.

5.3.8. Prueba del bloque de control de FIFOs

El bloque de control de FIFOs tiene como finalidad generar las señales de reloj que controlan la lectura de las memorias FIFO. Tal y como se analizó en la sección 5.3.8, esto se lleva a cabo bloqueando la señal de reloj aplicada al control de lectura de las FIFO por un determinado tiempo, justo después de la aplicación de una señal de *reset*.

Para la simulación de este bloque se utilizó un conjunto muy simple de vectores de prueba que generaban una señal de reloj y una señal de inicialización o *reset*. El tamaño de la

memoria FIFO determina el número de períodos de reloj que debe bloquearse la señal de lectura. Para utilizar este bloque varias veces en el sistema con memorias de distinto tamaño, es necesario que sea configurable, y así fue diseñado. Para las pruebas se considera que el control se realiza sobre una memoria de 256 bits de capacidad. El resultado de esta prueba se observa en la figura 88, que cumple con los requisitos esperados.

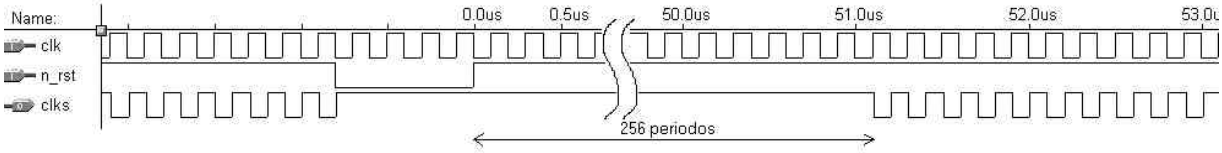


Figura 88. Resultados de la simulación con los vectores de prueba sobre el bloque de control de FIFOs.

5.3.9. Prueba del bloque de sincronización

Tal y como se mencionó en la sección 4.2.12, el bloque de sincronización debe realizar varias funciones:

- Generación del reloj de procesamiento principal.
- Sincronización de la toma de datos y generación de datos para el sistema y las FIFO de retardo para el cálculo de gradientes.
- Retardo de las señales de sincronismo de vídeo según latencia del sistema.
- Sincronización del *Reset* general del sistema con un sincronismo vertical.

Para probar cada una de estas funciones se han creado grupos de vectores que permiten separar el análisis de los resultados del funcionamiento general del bloque de sincronización.

La primera función del bloque de sincronización es la de generar el reloj interno de procesamiento (*pclk*) y, tal y como se mencionó, debe generarse sólo en los casos en que la señal de dato válido (*dv*) se encuentre activa y el *Reset* del sistema (*n_prst*) lo permita. La simulación de ésta función del bloque de sincronización se muestra en el diagrama de tiempo de la figura 89.

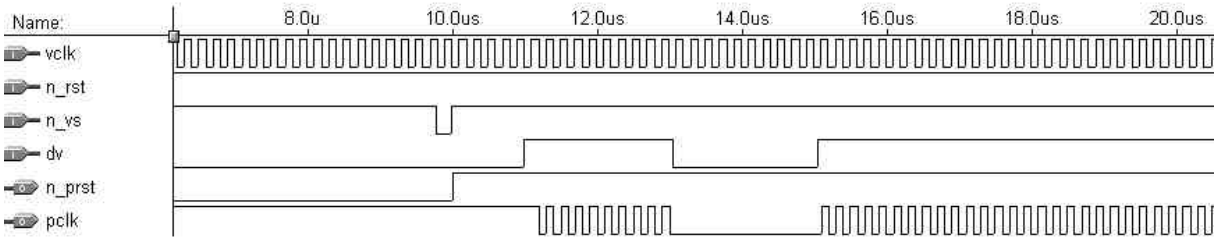


Figura 89. Resultados de la simulación de generación de reloj principal de procesamiento.

En la sincronización de los datos, éstos deben sincronizarse con el reloj de procesamiento del sistema (*pclk*), el cual se genera a partir del reloj procedente de la cámara (*vclk*). Como se señaló, deben generarse dos datos, uno sincronizado con el flanco de subida del reloj (*pdat*) para ser utilizado en la entrada de las FIFO y otro (*x000*) sincronizado con el flanco de bajada para ser utilizado por el bloque de cálculo de gradientes.

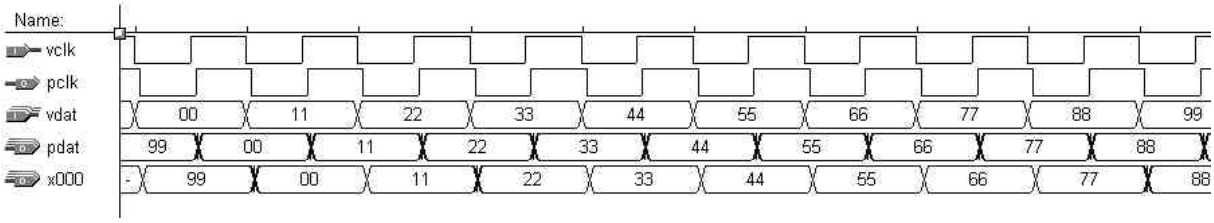


Figura 90. Resultados de la simulación de sincronización de datos.

La función de retardo de las señales de sincronismo de vídeo (*n_vs*, *n_hs*, *dv*) según la latencia del sistema se prueba alimentando el bloque con las señales de sincronismo y observando el retraso que sufren, tal y como se muestra en la figura 91 y con mayor detalle en la figura 92.

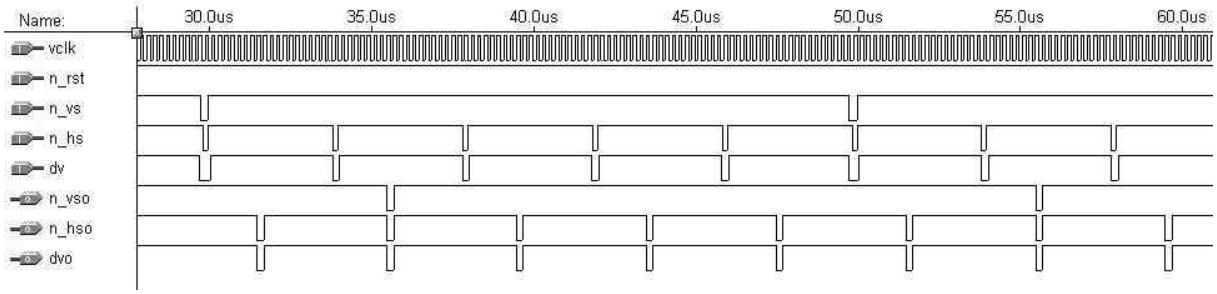


Figura 91. Resultados de la simulación de retardo en los sincronismos de vídeo.

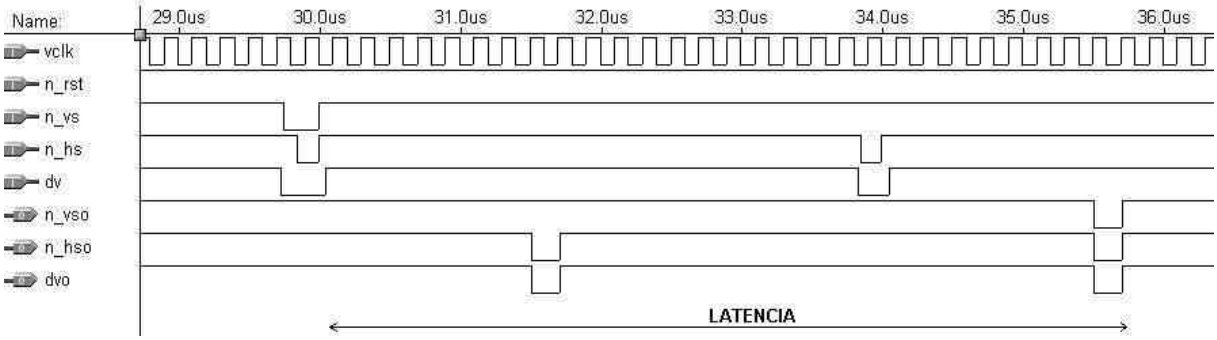


Figura 92. Retardo entre las señales de sincronismo de vídeo de entrada y de salida para compensar la latencia del sistema.

La prueba de la función de Sincronización del *Reset* se observa en el diagrama de tiempos de la figura 93. Básicamente se crea una nueva señal de *Reset* interna (*n_prst*) cuya validez comienza con el *Reset* externo y finaliza en cuanto se presenta un sincronismo vertical.

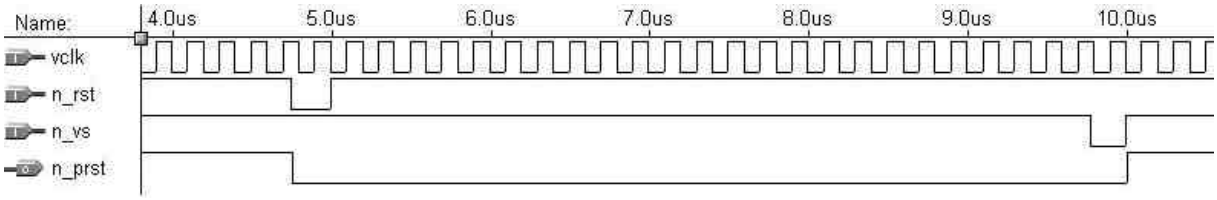


Figura 93. Resultados de la simulación de sincronización del *Reset* en el bloque de sincronización.

5.4. Pruebas generales del sistema

En las pruebas generales del sistema se procede a realizar una simulación del funcionamiento de todo el sistema, utilizando para ello el banco de prueba VHDL descrito en la sección 3.3 y el *software* de simulación *ModelSim*. Este *software* es básicamente un interpretador de VHDL en su sentido más puro, por lo que no nos permite ni sintetizar ni determinar los retardos y otros problemas que pueden surgir por el uso de circuitos físicos. Pero éstos debieron haber sido detectados en las pruebas individuales de cada bloque utilizando el *software* MAX+Plus II en la sección anterior.

La ventaja del *software* de simulación *ModelSim* reside en su capacidad de lectura y escritura de ficheros (siguiendo el estándar VHDL) y en su capacidad de aceptar un modelado de tipo comportamental. Por tanto, nos permite utilizar el banco de pruebas y modelos comportamentales de la memoria FIFO, dos piezas clave para lograr la simulación del funcionamiento del sistema.

Para estas pruebas se utiliza la descripción VHDL del sistema completo junto con el banco de pruebas, tal y como fue descrito en la sección 4.2.14. En la figura 94 muestra en forma esquematizada el sistema de prueba a simular.

Aunque el sistema está desarrollado para procesar imágenes de prácticamente cualquier tamaño, y en las pruebas de respuesta en tiempo de las FPGA se determina que con la tecnología actual el sistema soporta imágenes de 256 x 256 *pixels*, para estas pruebas se prepararon secuencias de imágenes de 64 x 64 *pixels*. Esto se hace con el fin de agilizar las pruebas y reducir el tamaño de los ficheros *hex* de imágenes. El resultado es igualmente válido para imágenes mayores y el análisis de los mismos es mucho más simple.

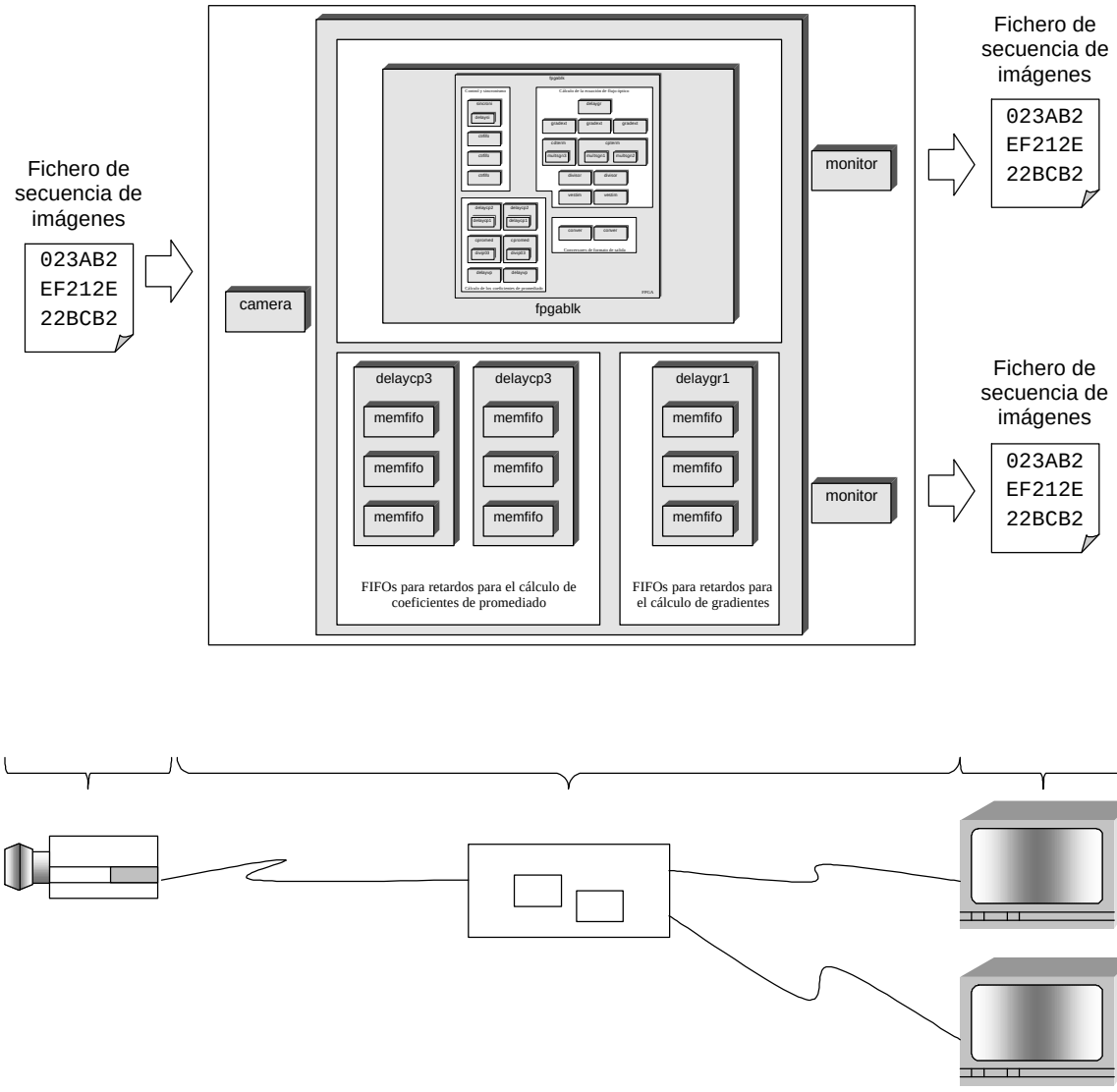


Figura 94. Montaje para las pruebas generales del sistema

Las secuencias de imágenes preparadas fueron las siguientes:

- **Secuencia 1:** Franja vertical desplazándose de izquierda a derecha a una velocidad de un *pixel* por cuadro.
- **Secuencia 2:** Franja vertical desplazándose de izquierda a derecha a una velocidad de dos *pixels* por cuadro.
- **Secuencia 3:** Franja vertical desplazándose de izquierda a derecha a una velocidad de tres *pixels* por cuadro.
- **Secuencia 4:** Franja vertical desplazándose de derecha a izquierda a una velocidad de dos *pixels* por cuadro.
- **Secuencia 5:** Franja horizontal desplazándose de arriba a abajo a una velocidad de dos *pixels* por cuadro.

A continuación se analizan los resultados obtenidos en la simulación del sistema ante estas secuencias de imágenes.

5.4.1. Prueba con secuencia 1

La *secuencia 1* se compone de en una franja vertical de bordes suaves que se desplaza de izquierda a derecha en una secuencia de 17 imágenes, algunas de las cuales se muestran en la figura 95. El desplazamiento es de un *pixel* por cuadro.

Como resultado de la operación del sistema ante esta secuencia de estímulo, se obtienen los dos ficheros de imágenes *hex* que representan las dos componentes del flujo óptico. Utilizando el programa *Visor* (estudiado en la sección 3.3.2) se obtiene la imagen vectorial mostrada en la figura 96. Esta imagen está tomada en el octavo cuadro de la secuencia. En ella se observa los vectores del flujo óptico hacia la derecha, indicando que en la zona existe un patrón de imagen que se mueve en dicho sentido. Se observan también en la zona de movimiento la presencia de franjas donde los vectores se hacen cero, esto puede ser debido a la aproximación en el cálculo cuando se realiza el suavizado, pero para llegar a esta conclusión es necesario ver el resultado con las demás secuencias de imágenes.

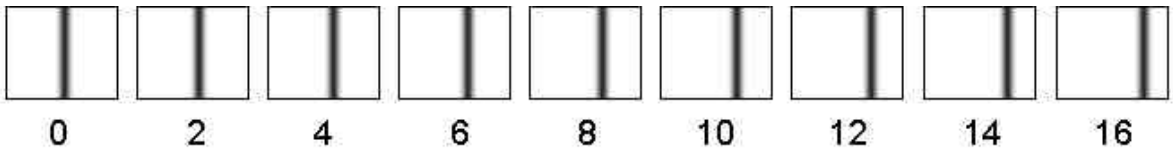


Figura 95. Secuencia 1 de imágenes de prueba.

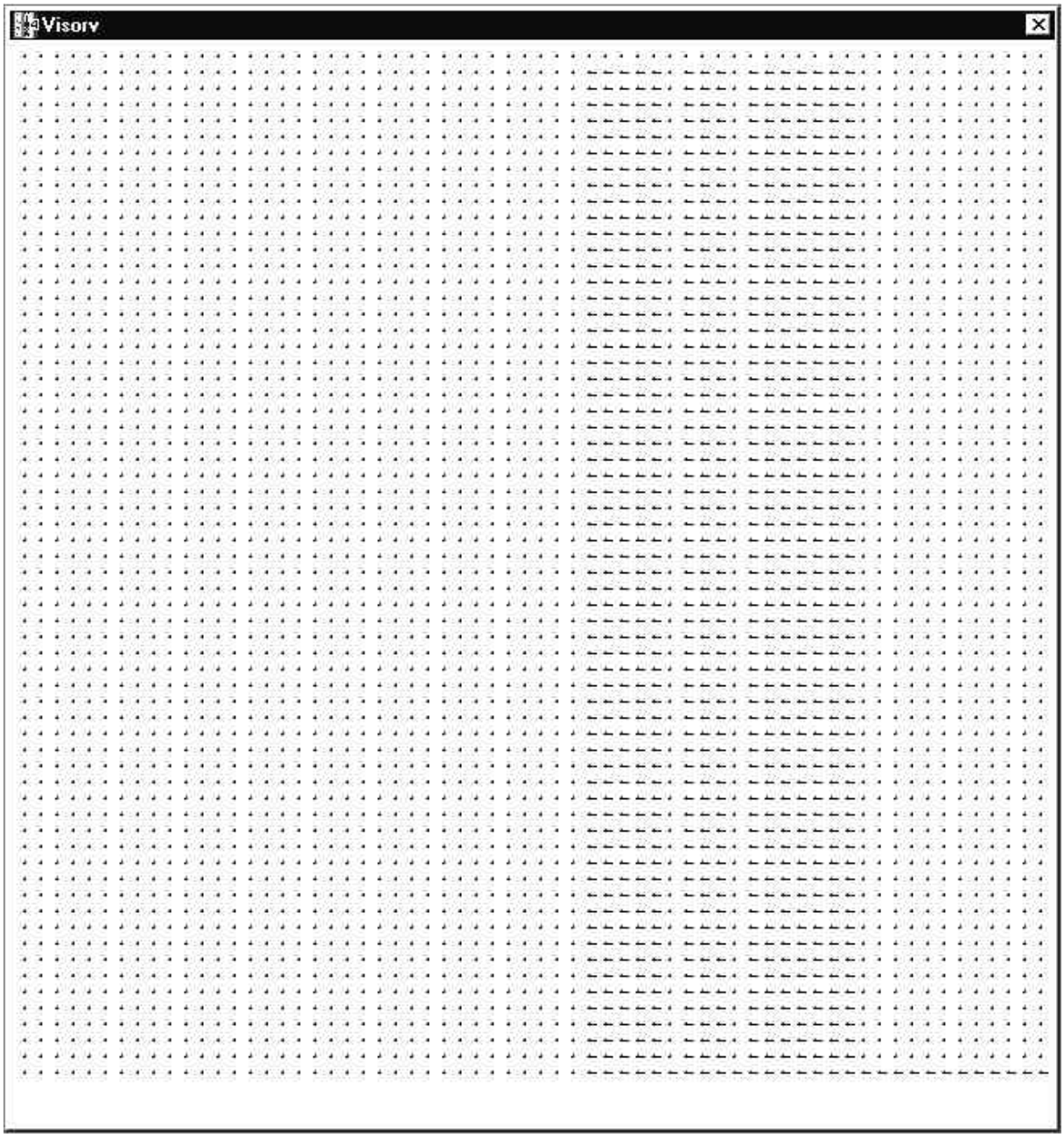


Figura 96. Flujo óptico obtenido con la *secuencia 1*.

A pesar de que la información mostrada en la imagen de la figura 96 es útil desde el punto de vista visual, es más interesante ver y analizar los valores que aparecen en el mismo fichero *hex*. Un fragmento de los resultados en formato *hex* para la componente horizontal del flujo óptico se muestra en la figura 97. En él se pueden ver los sucesivos valores 80h y 81h que toman los vectores. Tal y como se señaló en la sección 3.3.1, los valores se encuentran desplazados en un valor de 80h, por lo que este valor representa una velocidad de 0 y el valor 81h representa una velocidad de 1 *pixel*/cuadro, que corresponde con el valor de velocidad con que fue creada la *secuencia 1* de imágenes.

```
80 80 80 80 81 81 81 81 81 80 81 81 81 80 81 81 81 81 81 81 81 80 80 80
80 80 80 80 81 81 81 81 81 80 81 81 81 80 81 81 81 81 81 81 81 80 80 80
80 80 80 80 81 81 81 81 81 80 81 81 81 80 81 81 81 81 81 81 81 80 80 80
80 80 80 80 81 81 81 81 81 80 81 81 81 80 81 81 81 81 81 81 81 80 80 80
80 80 80 80 81 81 81 81 81 80 81 81 81 80 81 81 81 81 81 81 81 80 80 80
```

Figura 97. Extracto del fichero *hex* de la componente horizontal del flujo óptico obtenido con la *secuencia 1*.

5.4.2. Prueba con secuencia 2

La *secuencia 2* se compone de en una franja vertical de bordes suaves que se desplaza de izquierda a derecha en una secuencia de 17 imágenes, algunas de las cuales se muestran en la figura 98. El desplazamiento es de dos *pixels* por cuadro.

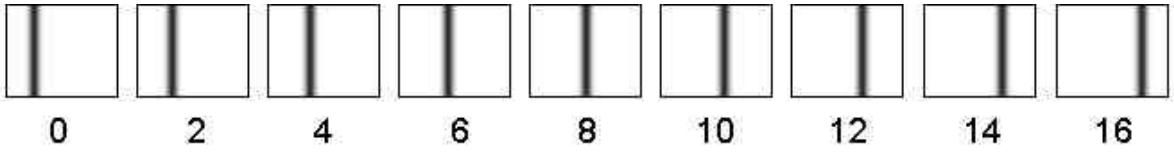


Figura 98. Secuencia 2 de imágenes de prueba.

El resultado de la operación del sistema ante esta secuencia de imágenes se muestra en la forma vectorial suministrada por el programa *Visor* en la figura 99. Esta imagen está tomada en el octavo cuadro de la secuencia. En ella se observan los vectores del flujo óptico hacia la derecha, indicando que en la zona existe un patrón de imagen que se mueve en dicho sentido.

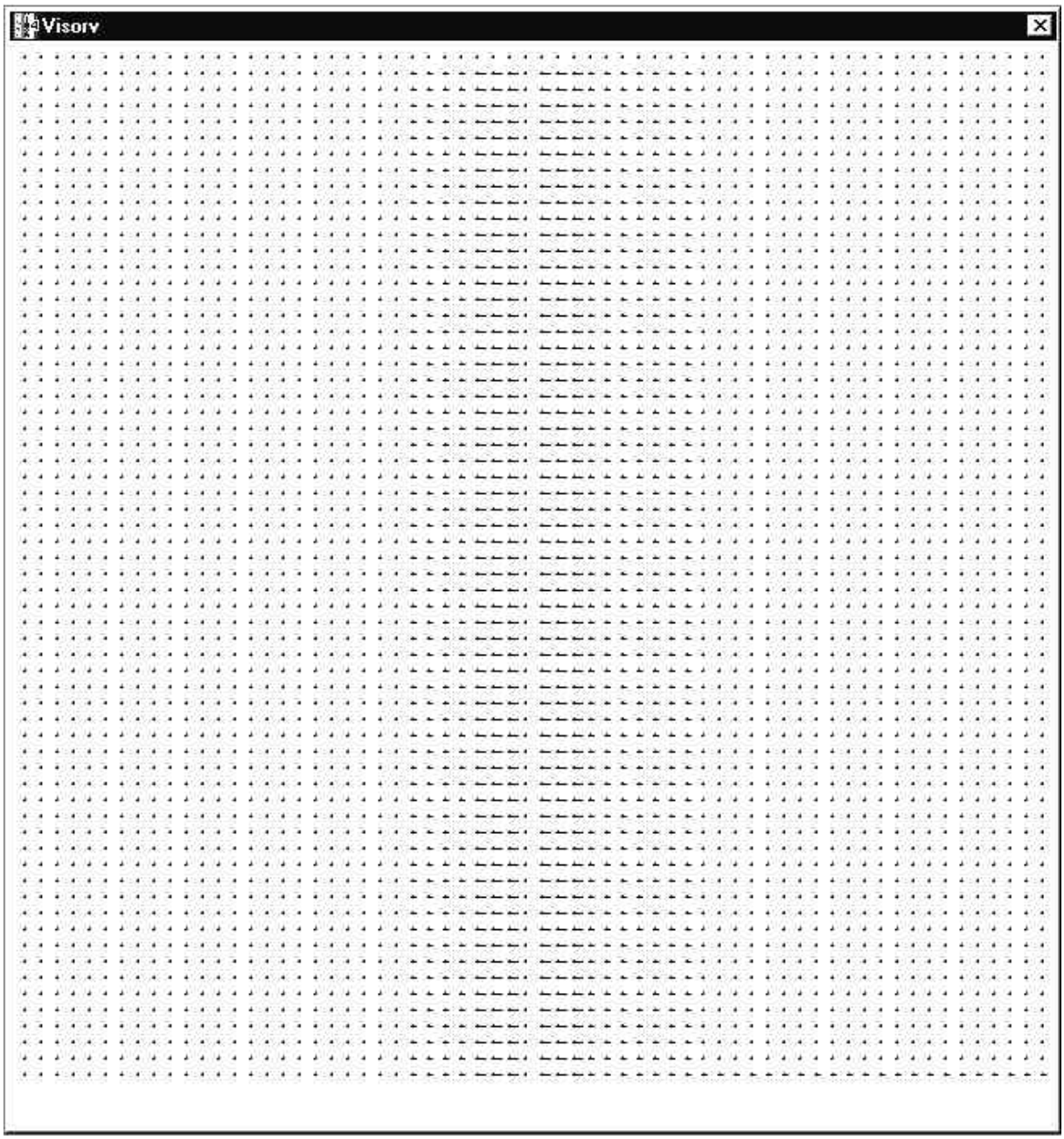


Figura 99. Flujo óptico obtenido con la *secuencia 2*.

Analizando el fichero en formato *hex* de la componente horizontal del flujo óptico, mostrado en la figura 100. Se observa que los valores de los vectores de velocidad indican en varios puntos los 2 *pixels* por cuadro esperados, pero también se presentan franjas donde los valores de velocidad aún no convergen totalmente y la velocidad es cero o un *pixel* por cuadro.

```
80 80 80 80 81 81 81 81 82 82 82 80 82 82 82 81 81 81 81 81 81 81 80 80
80 80 80 80 81 81 81 81 82 82 82 80 82 82 82 81 81 81 81 81 81 81 80 80
80 80 80 80 81 81 81 81 82 82 82 80 82 82 82 81 81 81 81 81 81 81 80 80
80 80 80 80 81 81 81 81 82 82 82 80 82 82 82 81 81 81 81 81 81 81 80 80
80 80 80 80 81 81 81 81 82 82 82 80 82 82 82 81 81 81 81 81 81 81 80 80
```

Figura 100. Extracto del fichero *hex* de la componente horizontal del flujo óptico obtenido con la *secuencia 2*.

5.4.3. Prueba con secuencia 3

La *secuencia 3* se compone de en una franja vertical de bordes suaves que se desplaza de izquierda a derecha en una secuencia de 17 imágenes, algunas de las cuales se muestran en la figura 101. El desplazamiento es de tres *pixels* por cuadro.

El resultado de la operación del sistema ante la *secuencia 3* se muestra en la forma vectorial suministrada por el programa *Visor* en la figura 102. Esta imagen está tomada en el octavo cuadro de la secuencia. Al igual que en las secuencias anteriores, se observan los vectores del flujo óptico hacia la derecha, indicando que en la zona existe un patrón de imagen que se mueve en dicho sentido.

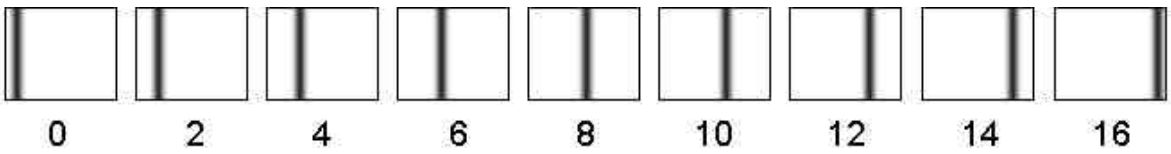


Figura 101. Secuencia 3 de imágenes de prueba.

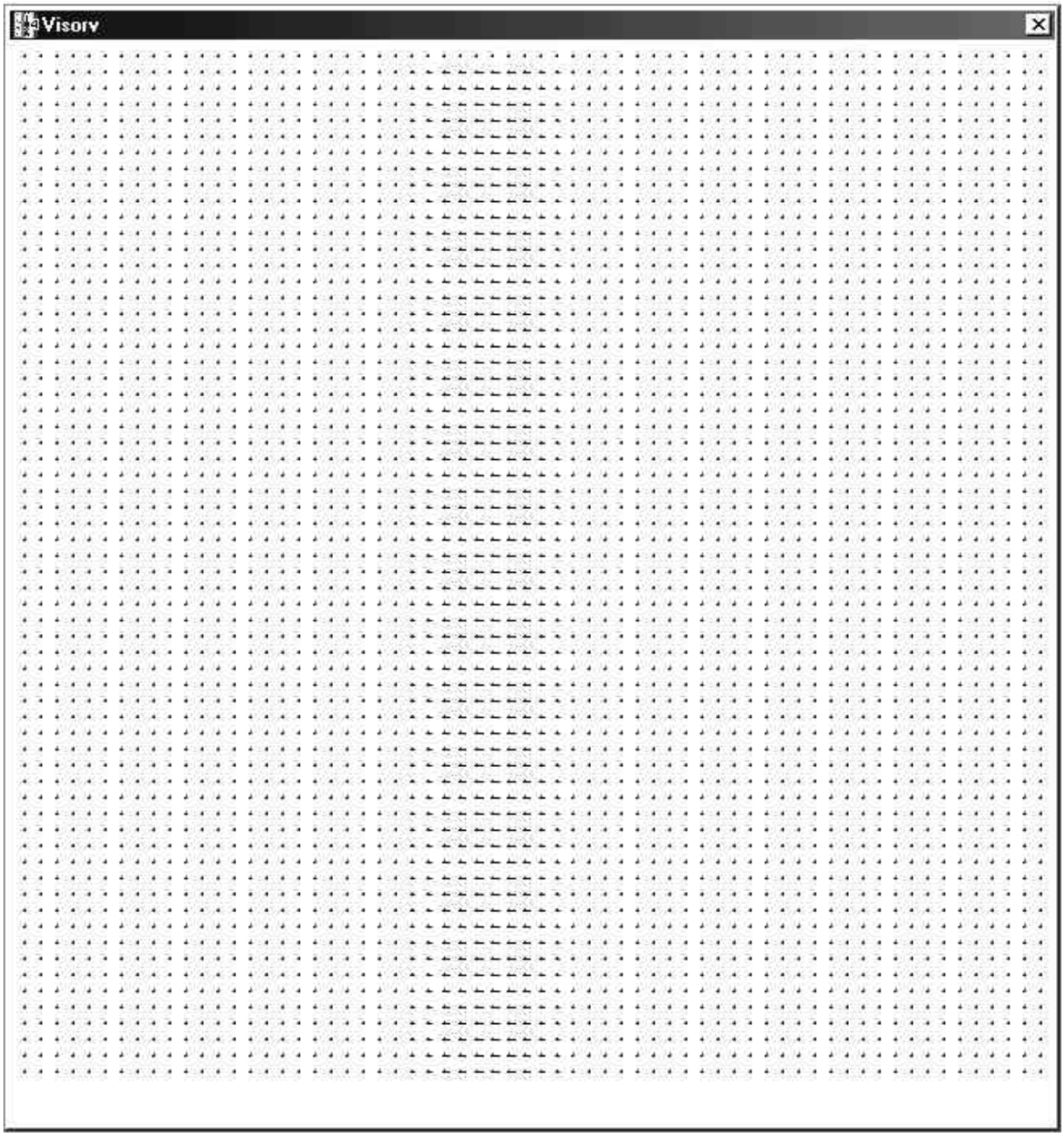


Figura 102. Flujo óptico obtenido con la *secuencia 3*.

Analizando el fichero en formato *hex* de la componente horizontal del flujo óptico, mostrado en la figura 103, se observa que los valores de los vectores de velocidad indican en algunos puntos los 3 *pixels* por cuadro esperados, pero se presentan varias franjas donde los valores de velocidad aún no convergen totalmente y la velocidad indicada es de uno a cinco *pixels* por cuadro.

```
80 80 80 81 81 81 82 83 84 84 85 85 85 84 83 82 81 81 81 81 81 81 80 80
80 80 80 81 81 81 82 83 84 84 85 85 85 84 83 82 81 81 81 81 81 81 80 80
80 80 80 81 81 81 82 83 84 84 85 85 85 84 83 82 81 81 81 81 81 81 80 80
80 80 80 81 81 81 82 83 84 84 85 85 85 84 83 82 81 81 81 81 81 81 80 80
80 80 80 81 81 81 82 83 84 84 85 85 85 84 83 82 81 81 81 81 81 81 80 80
```

Figura 103. Extracto del fichero *hex* de la componente horizontal del flujo óptico obtenido con la *secuencia 3*.

5.4.4. Prueba con secuencia 4

La *secuencia 4* se compone de en una franja vertical de bordes suaves que se desplaza de derecha a izquierda en una secuencia de 17 imágenes, algunas de las cuales se muestran en la figura 104. El desplazamiento es de dos *pixels* por cuadro.

Con esta secuencia se cambia el sentido del movimiento de la franja vertical y como resultado, los vectores de flujo óptico deben cambiar de sentido. Y así ocurre, tal como se observa en la representación obtenida con el programa *Visor* y mostrada en la figura 105. Los vectores indican un movimiento de derecha a izquierda.

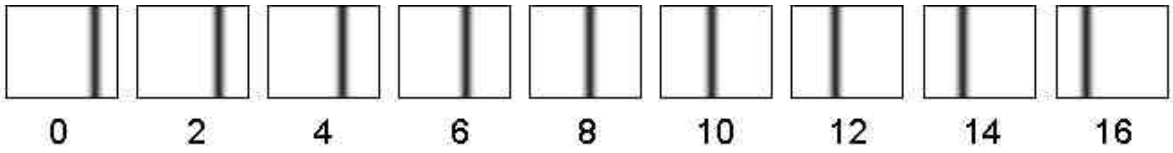


Figura 104. Secuencia 4 de imágenes de prueba.

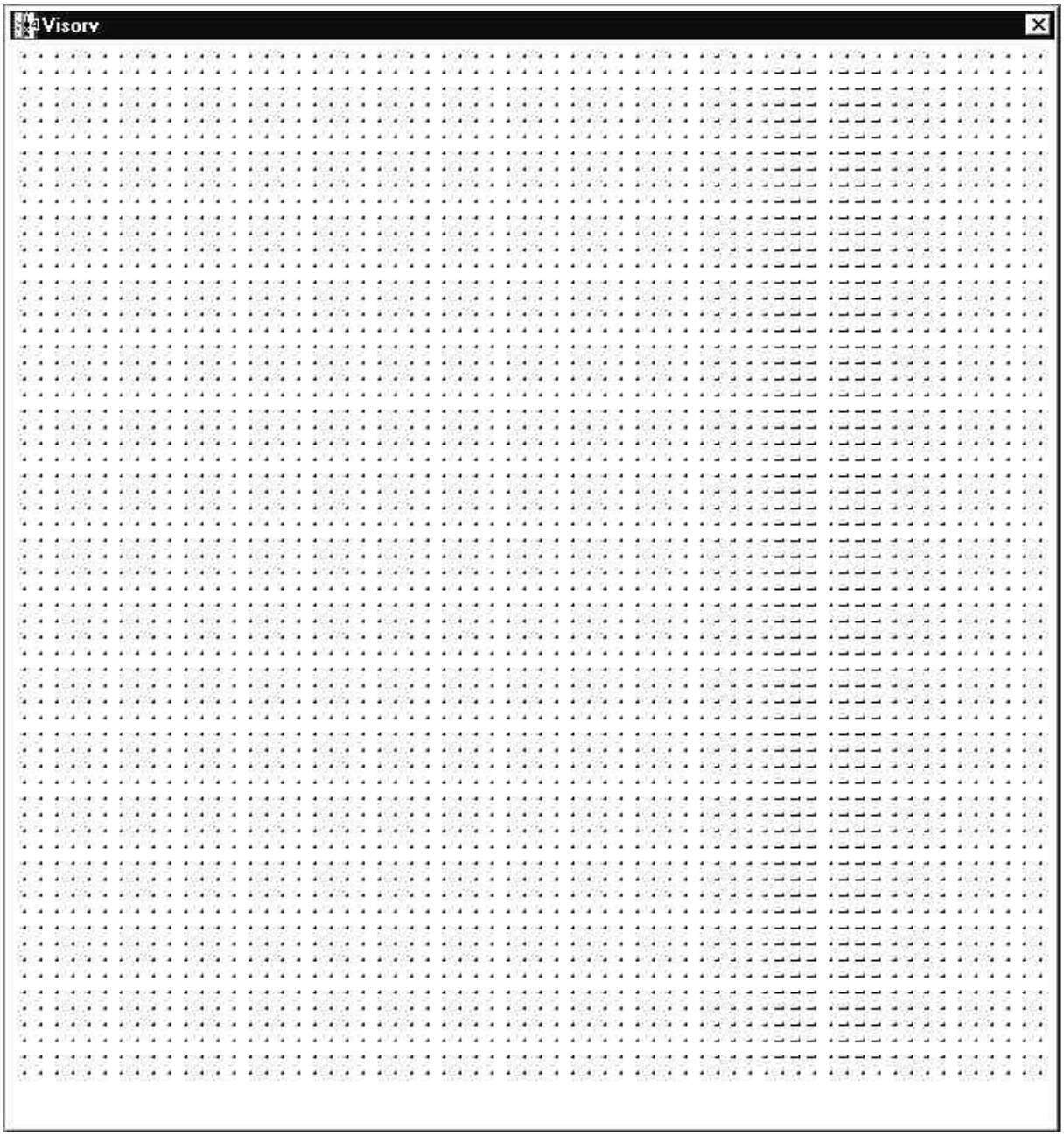


Figura 105. Flujo óptico obtenido con la *secuencia 4*.

En figura 106 se muestra un fragmento del fichero en formato *hex* de la componente horizontal del flujo óptico. En esta ocasión se observan que los valores de velocidad obtenidos tiene valores inferiores a 80h, lo cual indica que el movimiento detectado es hacia la izquierda. Una vez más, en varios puntos de la imagen se observa el valor de 2 *pixels* por cuadro esperados, pero también se presentan franjas donde los valores de velocidad aún no convergen totalmente y la velocidad es cero o un *pixel* por cuadro.

```
80 80 80 80 7F 7F 7F 7F 7E 7E 7E 80 7E 7E 7E 7F 7F 7F 7F 80 80 80 80 80
80 80 80 80 7F 7F 7F 7F 7E 7E 7E 80 7E 7E 7E 7F 7F 7F 7F 80 80 80 80 80
80 80 80 80 7F 7F 7F 7F 7E 7E 7E 80 7E 7E 7E 7F 7F 7F 7F 80 80 80 80 80
80 80 80 80 7F 7F 7F 7F 7E 7E 7E 80 7E 7E 7E 7F 7F 7F 7F 80 80 80 80 80
80 80 80 80 7F 7F 7F 7F 7E 7E 7E 80 7E 7E 7E 7F 7F 7F 7F 80 80 80 80 80
```

Figura 106. Extracto del fichero *hex* de la componente horizontal del flujo óptico obtenido con la *secuencia 4*.

5.4.5. Prueba con secuencia 5

La *secuencia 5* se compone de en una franja horizontal de bordes suaves que se desplaza de arriba a abajo en una secuencia de 17 imágenes, algunas de las cuales se muestran en la figura 107. El desplazamiento es de dos *pixels* por cuadro.

Utilizando el programa *Visor* se observan (figura 108) los resultados de la operación del sistema ante esta secuencia: un flujo óptico que indica un movimiento vertical de la franja de la imagen, de arriba hacia abajo.

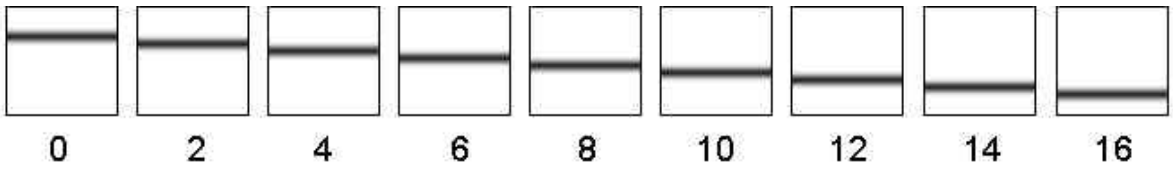


Figura 107. Secuencia 5 de imágenes de prueba.

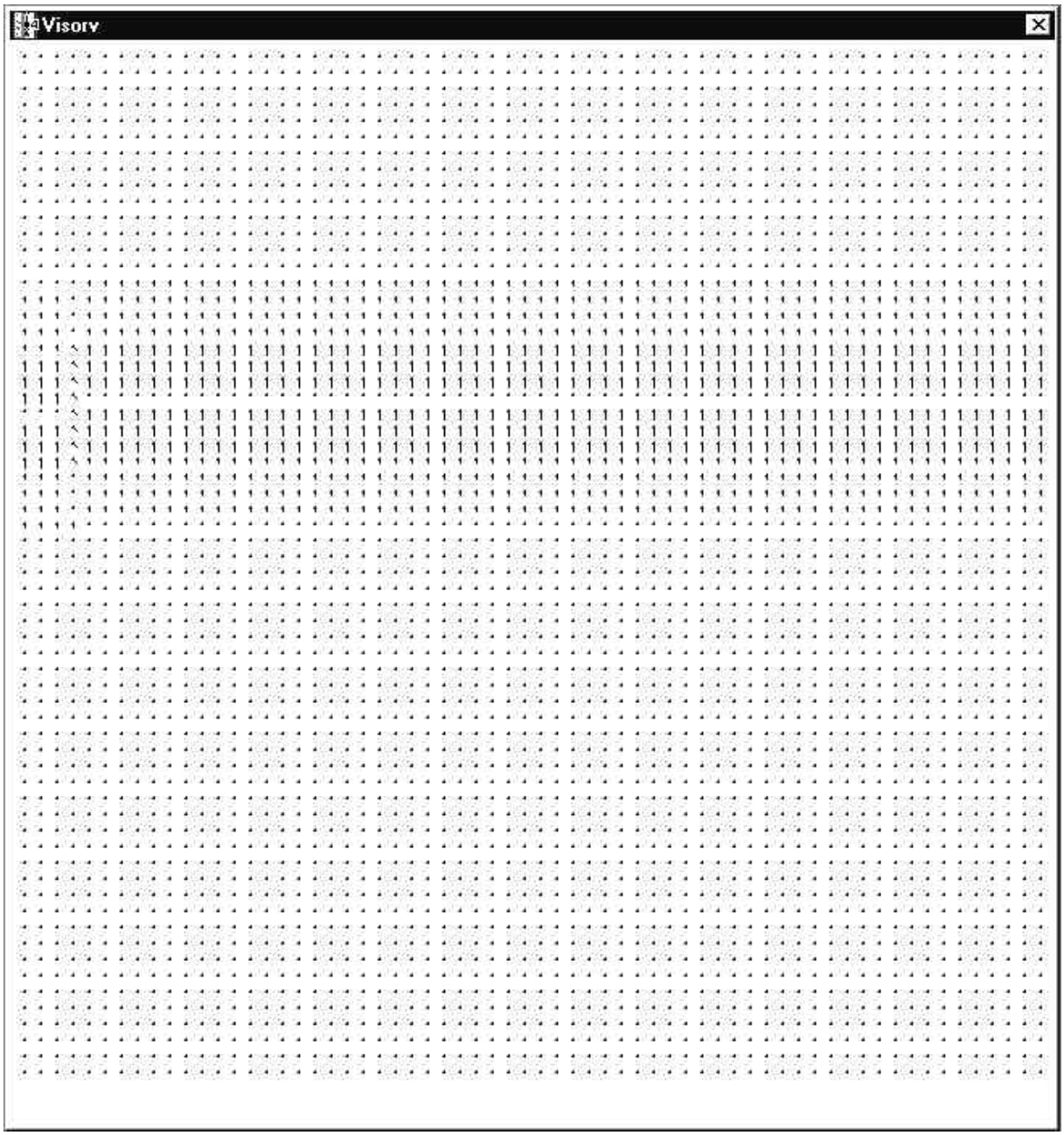


Figura 108. Flujo óptico obtenido con la *secuencia 5*.

El análisis de los ficheros *hex*, en este caso, se lleva a cabo en el que contiene la información vertical. Un fragmento del mismo se presenta en la figura 109 y en él observan valores de velocidad inferiores a 80h, indicando con ello que el sentido del flujo óptico es de arriba abajo. El valor del vector de velocidad es, en múltiples puntos de 2 *pixels* por cuadro en otros puntos el valor de velocidad es de 1 *pixel* por cuadro, no habiendo convergido aún al valor final.

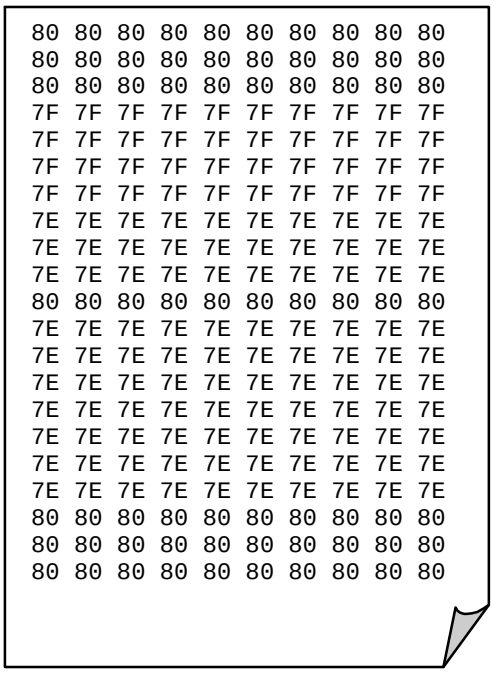


Figura 109. Extracto del fichero *hex* de la componente vertical del flujo óptico obtenido con la *secuencia 5*.

5.5. Síntesis de la FPGA

Hasta este momento cada bloque del sistema se había sintetizado independientemente en una FPGA para observar su comportamiento ahora es necesario integrar todos los bloques del sistema.

Cuando se dio comienzo a esta Tesis, las limitaciones de los dispositivos comerciales del tipo FPGA no permitían que todo el sistema pudiera ser sintetizado en un solo integrado. Pero el desarrollo de éstos ha sido vertiginoso en los últimos años y hoy en día es posible sintetizar el dispositivo de extracción de flujo óptico, a excepción de las memorias FIFO, en un solo dispositivo.

Desde los inicios, los dispositivos elegidos fueron los de la serie 10K de *Altera* que por estos tiempos eran los dispositivos de mayor capacidad que producía el mencionado fabricante. En un principio durante las pruebas y ajustes el sistema se sintetizaba en dos o tres FPGA 10K50 ó 10K70, pero ya para las últimas síntesis realizadas se pudo sintetizar el sistema en una sola FPGA, la 10K130. En la tabla 13 se resumen algunas de las características de la realización del sistema en un dispositivo FPGA, a no ser que se modifiquen algunas de estas etapas aumentando el grado de segmentación de los mismos.

Tabla 13. Características de la FPGA sintetizada.

Dispositivo	EPF10K130EQC240-1
Celdas lógicas utilizadas	6602
Número de celdas lógicas disponibles	6656
Porcentaje de utilización de celdas lógicas	99%
Total de Flip-Flops requeridos	3592
Pines de entrada requeridos	86
Pines de salida requeridos	32
Total de pines requeridos	112
Total de pines disponibles	180
Porcentaje de utilización de pines	62%

5.6. Análisis de resultados

Varias conclusiones pueden extraerse de las pruebas realizadas sobre la arquitectura planteada. En primer lugar, observando las pruebas de los bloques individuales en las simulaciones funcionales, podemos observar las posibilidades físicas de realizar la arquitectura utilizando la tecnología basada en FPGA. En segundo lugar, con las pruebas generales del sistema evaluamos el funcionamiento del mismo para la obtención del flujo óptico. En tercer y último lugar, la síntesis de la FPGA permite establecer la posibilidad de construir el sistema en un o varios dispositivos FPGA disponibles en la actualidad.

De las pruebas de bloques determinamos que la arquitectura segmentada utilizada para desarrollar el sistema para la determinación del flujo óptico en tiempo real con imágenes digitales de 256 x 256 *pixels* de resolución es factible utilizando las FPGA de mayor velocidad existentes en la actualidad. Sin embargo, de la figura 81 se desprende una limitación para una futura aplicación con imágenes de doble resolución. El hecho de duplicar la resolución trae consigo una cuadruplicación del número de *pixels* a procesar y por tanto si se mantiene la misma cantidad de imágenes consecutivas por unidad de tiempo, es necesario cuadruplicar también el reloj de procesamiento, por tanto, varias etapas del sistema no podrían procesar a tiempo sus resultados antes de pasarlos a la etapa siguiente.

Con las pruebas generales del sistema se observa que el sistema parece capaz de determinar con cierta precisión el flujo óptico a partir de ciertas imágenes sintéticas. Sin embargo, se observan varias imprecisiones cuyos efectos utilizando imágenes reales no son posibles de cuantificar. Las simulaciones se han realizado con imágenes sintéticas simplificadas para posibilitar una simulación utilizando las herramientas disponibles. Es muy posible que en la realización física del sistema se logren mejores resultados dada la mayor cantidad de iteraciones en el cálculo del flujo óptico, así como también la existencia de texturas más densas en las imágenes reales que ayudan en la convergencia de las ecuaciones de flujo óptico.

La síntesis del sistema en dispositivos FPGA ha ido evolucionando a través del tiempo en el cual se ha llevado a cabo esta Tesis. En los primeros años, la tecnología de las FPGA no permitía la construcción del sistema en menos de tres dispositivos, luego podía ser construido

en dos dispositivos, y ahora, tal y como se señaló, el sistema puede ser construido utilizando sólo un dispositivo FPGA y una serie de memorias FIFO externas. Es muy posible que en poco tiempo también parte de las memorias FIFO puedan ser incluidas dentro de la FPGA.

La síntesis se realizó de manera automática sobre un dispositivo FPGA de la serie 10K de *Altera*, por tanto el programa seleccionó el dispositivo más simple de la familia que pudiera contener el sistema, el EPF10K130EQC240-1. Utilizando este dispositivo en la realización física del sistema se tendrían dos desventajas, la primera es que si el desempeño del sistema con señales e imágenes reales requiriera algún circuito adicional, de sincronización o filtrado, la FPGA propuesta no contaría con recursos para implementarlo. Por ello, se deberá optar por una FPGA con más recursos dentro de la misma familia. La segunda desventaja del dispositivo propuesto se encuentra en el número excesivo de pines que no se utilizan. En la tabla 13 se observa que sólo el 62% de los pines se utilizan. Si bien es útil para futuras ampliaciones el disponer de pines adicionales, es necesario considerar que ellos complican innecesariamente el circuito impreso y aumentan la posibilidad de fallos del sistema.

Recientemente se han realizado intentos encaminados a la utilización de la nueva familia de FPGAs de *Altera*: las *APEX 20KE*. Estas FPGAs tienen densidades de integración superiores a la familia *FLEX 10K* además de disponer de mayor cantidad de memoria RAM con lo cual es posible incluir parte de las memorias FIFO dentro de la FPGA. Sin embargo, el uso de esta nueva familia de FPGAs requiere de la utilización de nuevos programas para la síntesis con los cuales es necesario familiarizarse además de configurarlos correctamente para que permitan la utilización de todos los módulos VHDL desarrollados en esta Tesis..

6

Capítulo 6

Conclusiones y trabajo futuro

...unos granos cayeron a lo largo del camino:
vinieron las aves y se los comieron
Otros cayeron en terreno pedregoso, con muy poca tierra,
y brotaron enseguida, pues no había profundidad.
Pero apenas salió el sol, los quemó y, por falta de raíces, se secaron.
Otros cayeron en medio de cardos: estos crecieron y los ahogaron.
Otros granos, finalmente, cayeron en buena tierra
y produjeron cosecha,...

San Mateo 13,4-8

6.1. Conclusiones

El objetivo primordial de esta Tesis era el *modelado por medio de un lenguaje de descripción de hardware de un sistema para la determinación del flujo óptico a partir de secuencias de imágenes en tiempo real*. Para cumplir con este objetivo fue preciso hacer un estudio de las diferentes posibilidades para la determinación del flujo óptico que pudieran existir y seleccionar alguna de ellas que fuera posible desarrollar por medio de dispositivos electrónicos manteniendo siempre la meta de la determinación en tiempo real.

Una vez seleccionado el método de determinación del flujo óptico fue necesario desarrollar una arquitectura adecuada al mismo. En este caso fue seleccionado el método de determinación del flujo óptico basado en gradientes y formulado por Horn y Schunck [HORN81]. Pudiera haber sido desarrollada una arquitectura diferente haciendo menos exigentes las condiciones que se impusieron desde un principio sobre la resolución y la velocidad de las imágenes, pero el hecho de considerar adecuadamente el término *tiempo real* estableció un reto que obligó a desarrollar una arquitectura más genérica y con mayores posibilidades de cara al futuro.

Tal y como fue el propósito inicial, el desarrollo de esta arquitectura fue realizado totalmente utilizando el lenguaje de descripción *hardware* VHDL. Para ello fue necesario desarrollar una metodología de trabajo que permitió describir las ecuaciones originales por medio del lenguaje VHDL teniendo siempre en cuenta que la arquitectura debería ser sintetizable para su posterior realización física. El hecho de que aún pueda considerarse el lenguaje VHDL como de reciente introducción trajo consigo una serie de problemas en el uso de las herramientas de síntesis y simulación, debido a las diferencias de interpretación por parte de los distintos fabricantes de las mismas. Sin embargo, ya hoy día comienzan a vislumbrarse herramientas más estandarizadas que facilitarán considerablemente el diseño de sistemas similares al realizado en esta Tesis.

Para la realización de la Tesis también fue necesario desarrollar una serie de herramientas que permitieron analizar los algoritmos y probar el sistema. Algunas de estas herramientas tienen por sí solas un valor muy relevante incluso para aplicaciones en otras líneas de investigación.

6.2. Resumen de las aportaciones originales

Varias han sido las aportaciones originales de esta Tesis. La primera de ellas es la transformación de un algoritmo para la determinación del flujo óptico desarrollado hasta ahora por medios *software* a un medio *hardware*. Con ello se logra algo que hasta el momento no se había podido: una determinación del flujo óptico en tiempo real, entendiéndose por tiempo real el procesamiento de secuencias de imágenes a resoluciones y tiempos similares a los encontrados en la televisión estándar.

La metodología aquí presentada es uno de las grandes aportaciones de esta Tesis. Es cierto que el objetivo a alcanzar, el desarrollo de una arquitectura para la obtención del flujo óptico, es una aplicación muy particular, pero la metodología puede ser aplicada en los casos más variados.

Otra aportación de esta Tesis es la metodología utilizada para llevar a cabo el desarrollo de una arquitectura a partir de una ecuación y que aquí se han dejado plasmadas de manera que sirvan de base a investigaciones similares que pudieran emprenderse en el futuro.

Por otro lado, se ha desarrollado una arquitectura segmentada que permite el desarrollo de nuevas aplicaciones modulares para el procesamiento de imágenes e tiempo real. En este ámbito de aplicaciones, también se desarrolló el banco de pruebas VHDL con el cual es posible que estos módulos creados en VHDL sean probados y ajustados antes de ser implementados por *hardware.* El hecho de crear este banco de pruebas trajo además como resultado un nuevo formato para intercambio de imágenes entre diferentes programas de prueba o simulación. Los formatos normalmente utilizados en la actualidad no pueden ser utilizados por el lenguaje estándar VHDL.

6.3. Trabajo futuro

En la línea de investigación comenzada en esta Tesis queda mucho trabajo por hacer. En primer lugar está la realización física del sistema aquí estudiado. Para ello se requiere preparar una serie de tarjetas electrónicas. La primera serie de tarjetas electrónicas que en este momento se construyen son las que conforman el banco de trabajo. Al igual que se creó un banco de trabajo *software* apropiado para las pruebas del sistema descrito en lenguaje VHDL, se requiere crear un banco de trabajo *hardware* con las mismas especificaciones para probar la realización física del sistema.

El banco de trabajo *hardware* que se está construyendo consta de tres bloques independientes. El primer bloque tiene la capacidad de tomar las señales de vídeo de una cámara convencional y generar las señales digitales de vídeo. El segundo bloque toma señales de vídeo digital y las convierte a señales de vídeo compuesto para poder ser visualizadas en un monitor de televisión estándar. El tercer bloque permitirá almacenar imágenes individuales de la secuencia para que puedan ser analizadas en un ordenador.

Después de la construcción del banco de trabajo, se procederá a la construcción del sistema propiamente dicho. Ya en este momento se están dando los primeros pasos en el desarrollo de la tarjeta que contendrá el sistema de extracción del flujo óptico. Sin embargo, es posible que la realización final del sistema contenga ciertas modificaciones a lo que se ha presentado en este documento, dado que cada día aparecen nuevos dispositivos que permitirán lograr optimizaciones del sistema así como también mayores posibilidades para ampliaciones futuras.

En vista del análisis de algunos de los resultados obtenidos en esta Tesis, es posible que sea necesario agregar algunos bloques adicionales al sistema, como pudiera ser un filtro para las señales de entrada de vídeo, con el fin de mejorar la precisión del sistema en el cálculo del

flujo óptico. No obstante la necesidad de ello se verá a medida que se vayan realizando los experimentos con señales reales.

También quedan abiertas las puertas para la realización del sistema en nuevos dispositivos que ya comienzan a aparecer, así como también la utilización de nuevos programas de síntesis que parecen ser más eficientes en la cantidad de recursos que utilizan. Esto lo posibilita el hecho de que el sistema fue desarrollado en su totalidad utilizando VHDL. Aún así, el hecho de que los nuevos programas de síntesis parecen soportar mejor el estándar de lenguaje VHDL, abre la posibilidad de mejorar la descripción y por tanto la operación de los diferentes módulos que contempla el sistema.

Las optimizaciones del sistema estarían encaminadas a aumentar la velocidad de funcionamiento para lograr el uso de imágenes de mayor resolución y a mejorar determinados bloques del sistema para obtener resultados más precisos en la estimación del flujo óptico. También cabe la posibilidad de integrar parte de las memorias FIFO dentro del mismo dispositivo FPGA, con lo cual el sistema se acercaría más a la definición de *system on a chip* que se busca hoy día.

En este momento, con la ayuda de fondos proporcionados por Departamento de Educación, Universidades e Investigación del Gobierno Vasco, se está procediendo a la realización física del sistema para la determinación del flujo óptico y para ello se contemplan todas estas optimizaciones y mejoras.

También es necesario destacar las líneas futuras que se presentan por el uso de otras aportaciones indirectas de esta Tesis, como ha sido con el caso del banco de pruebas desarrollado en VHDL que ha permitido el estudio de otros sistemas de procesamiento de imágenes entre los que se destacan el de mejoramiento dinámico de secuencias de imágenes. Ya se pretende una extensión de este banco de pruebas para aplicaciones relacionadas con secuencias de imágenes a color.

A más largo plazo, en el camino abierto por esta tesis se vislumbran las posibilidades de utilizar el flujo óptico para la caracterización tridimensional del entorno en vistas de aplicaciones tales como la conducción autónoma de vehículos. También existen planteamientos para su utilización en aplicaciones relacionadas con el seguimiento de objetos, que pueden ser de utilidad en el campo de la vigilancia electrónica.

6.4. Publicaciones generadas a partir de esta Tesis

El desarrollo de esta Tesis ha dado lugar a una gran cantidad de ideas directa o indirectamente relacionadas con el tema central de la misma, algunas de las cuales pudieron ser utilizadas también en el desarrollo de proyectos relacionados con el mundo empresarial. De todo esto ha surgido una gran cantidad de publicaciones en revistas y congresos nacionales e internacionales entre las que se pueden citar las siguientes:

- A. Zuloaga, J. Martín, L. Nozal, "Determinación del movimiento a partir de secuencias de imágenes - Procesamiento primario," Mundo Electrónico, n. 278, pp. 55-59, Madrid, España, Junio, 1997.

- A. Zuloaga, J. Martín, L. Nozal, "Determinación del movimiento a partir de secuencias de imágenes - Técnicas de determinación del movimiento," Mundo Electrónico, n. 279, pp. 26-30, Madrid, España, Julio, 1997.

- A. Zuloaga, J. Martín, L. Nozal, "Determinación del movimiento a partir de secuencias de imágenes," URSI'97 Actas del XII Simposium Nacional de la Unión Científica Internacional de Radio, vol. 1, pp. 211-214, Bilbao, España, Septiembre 17-20, 1997.

- J. García, A. Zuloaga, J. Ezquerra, J. Martín, "Banco de pruebas VHDL para la simulación de modelos para el procesamiento de imágenes," URSI'98 Actas del XIII Simposium Nacional de la Unión Científica Internacional de Radio, pp. 747-748, Pamplona, España, Septiembre 16-18, 1998.

- A. Ramos, A. Zuloaga, J. Ezquerra, J. Martín, "Modelo VHDL para la ecualización del histograma de imágenes de vídeo en tiempo real," URSI'98 Actas del XIII Simposium Nacional de la Unión Científica Internacional de Radio, pp. 759-760, Pamplona, España, Septiembre 16-18, 1998.

- A. Zuloaga, J. Martín, J. Ezquerra, "Hardware architecture for optical flow estimation in real time," ICIP'98 Proceedings of 1998 IEEE International Conference on Image Processing, vol. 3, pp. 972-976, Chicago, USA, Octubre 4-7, 1998.

- A. Zuloaga, U. Bidarte, J. Martín, J. Ezquerra, "Optical flow estimator using VHDL for implementation in FPGA," DCIS'98 Proceedings of XIII Desing of Circuits and Systems Conference, pp. 36-41, Madrid, España, Noviembre 17-20, 1998.

- U. Bidarte, J. Ezquerra, J. Martín, A. Zuloaga, "Adaptative and parametrizable histogram equalizer for real time video signals using VHDL," SNRFAI'99 Proceedings of VIII National Symposium on Pattern Recognition and Image Analysis, vol. 2, pp. 59-60, Bilbao, España, Mayo 12-14, 1999.

- L. Nozal, U. Bidarte, J. Ezquerra, J. Martín, A. Zuloaga, "Digital phase locked loop in image acquisition circuits," SNRFAI'99 Proceedings of VIII National Symposium on Pattern Recognition and Image Analysis, vol. 2, pp. 61-62, Bilbao, España, Mayo 12-14, 1999.

- A. Zuloaga, J. Martín, U. Bidarte, J. Ezquerra, "VHDL test bench for digital image processing systems using a new image format," FDL'99 Proceedings of Second International Forum on Design Languages, pp. 187-196, Lyon, Francia, Septiembre 1-3, 1999.

- A. Zuloaga, J. Martin, U. Bidarte, J. Ezquerra, "High speed architecture for image sequence processing described with VHDL," FDL'99 Proceedings of Second International Forum on Design Languages, pp. 177-186, Lyon, Francia, Septiembre 1-3, 1999.

- U. Bidarte, J. Ezquerra, A. Zuloaga, J. Martín, "VHDL modeling of an adaptive and reusable architecture for real-time image enhancement," VIUF'99 Proceedings of VHDL International Users Forum, pp. 94-100, Orlando, USA, Octubre 4-6, 1999.

- A. Zuloaga, J. Ezquerra, U. Bidarte, J. Martín, "VHDL simulation tool for designing image procesing systems," DCIS'99 Proceedings of XIV Desing of Circuits and Systems Conference, pp. 275-280, Palma de Mallorca, España, Noviembre 16-19, 1999.

- U. Bidarte, J. Martin, A. Zuloaga, J. Ezquerra, "FPGA-based adaptive histogram equalizer for real-time video signals," DCIS'99 Proceedings of XIV Desing of Circuits and Systems Conference, pp. 819-824, Palma de Mallorca, España, Noviembre 16-19, 1999.

- U. Bidarte, J. Martin, A. Zuloaga, J. Ezquerra, "Adaptative image brightness and contrast enhancement circuit for real-time vision systems," ICIT'00 Proceedings of 2000 IEEE International conference on industrial technology, Goa, India, Enero 19-22, 2000.

- U. Bidarte, J. Ezquerra, A. Zuloaga, J. García, A. Ramos, "Convolucionador de señales de vídeo en tiempo real desarrollado sobre FPGAs," URSI'99 Actas del XIV Simposium Nacional de la Unión Científica Internacional de Radio, pp. 282-283, Santiago de Compostela, España, Septiembre 8-10, 1999.

- I. Milikua, U. Bidarte, A. Zuloaga, "Aritmética distribuida aplicada a la optimización de recursos en la implementación hardware de un convolucionador de vídeo," URSI'00 Actas del XV Simposium Nacional de la Unión Científica Internacional de Radio, Zaragoza, España, Septiembre 13-15, 2000.

- I. Milikua, A. Zuloaga, U. Bidarte, A. Chavarría, "Convolution of image sequences using distributed arithmetic in real time," DCIS'00 Proceedings of XV Desing of Circuits and Systems Conference, Montpellier, Francia, Noviembre 21-24, 2000.

- J. L. Martín, A. Zuloaga, J. Cruzado, "Software workbench for movement estimation based on optical flow computation," VIIP'01 Proceedings of IASTED international conference on Visualization, Imaging, and Image Processing, Marbella, España, Septiembre 3-5, 2001.

- J. L. Martín, A. Zuloaga, J. Cruzado, "Banco de pruebas para la estimación del movimiento basado en el cálculo del flujo óptico," URSI'01 Actas del XVI Simposium Nacional de la Unión Científica Internacional de Radio, pp. 307-308 Madrid, España, Septiembre 19-21, 2001.

- J. L. Martín, J. Lázaro, U. Bidarte, A. Zuloaga, K. Espinosa, I. Goirizelaia, "Image acquisition and digitalizing system for an electronic voting system," DCIS'01 Proceedings of XVI Desing of Circuits and Systems Conference, Oporto, Portugal, Noviembre 20-23, 2001.

Bibliografía

Bibliografía

[ALPA99] L. Alparone, M. Barni, F. Bartolini, R. Caldelli, "Regularization of optic flow estimates by means of weighted vector median filtering," IEEE Transactions on Image Processing, vol. 8, n. 10, pp. 1462-1467, Octubre, 1999.

[ALTE01] "Altera web site," http://www.altera.com, Altera Corporation, 2001.

[ALTE98] Altera, *1998 Data book*, San Jose, CA: Altera Corporation, 1998.

[ANDR91] A. Andreou, K. Strohbehn, R. Jenkins, "Silicon retina for motion computation," Procedures 1991 IEEE International Symposium Circuits and Systems, pp. 1373-1376, Junio, 1991.

[ARAN89] G. Aranguren, Sistemas de tratamiento de imágenes por medios hardware con aplicación a robótica, Tesis Doctoral, Bilbao: Universidad del País Vasco, 1989.

[ARAP97] C. Arapis, S. Gibbs, C. Breiteneder, "Real-time segmentation of video on a multiprocessor platform," Paralel Computing, vol. 23, pp. 1777-1792, 1997.

[ATHA95] P. Athanas, A. Abbott, "Real-time image processing on a custom computing platform," Computer, vol. 28, n. 2, pp. 16-24, Febrero, 1995.

[ATME00] "Cameras," http://www.tcs.thomson-csf.com/ccd/camera.htm, Atmel Grenoble, 2000.

[AVID00] S. Avidan, A. Shashua, "Trayectory triangulation: 3D reconstruction of moving points from a monocular image sequence," IEEE Transaccions on Pattern Analysis and Machine Inteligence, vol. 22, n. 4, pp. 348-357, Abril, 2000.

[BART97] F. Bartolini, A. Piva, "Median based relaxation of smoothness constraints in optical flow computation," Pattern Recognition Letters, vol. 7, n. 18, pp. 649-655, Julio 30, 1997.

[BATC86] B. Batchelor, C. Bowman et al, "Developments in image processing for industrial inspection," Proceedings SPIE, vol. 730, pp. 34-46, 1986.

[BEAU95] S. Beauchemin, J. Barron, "The computation of optical flow," ACM Computation Surveys, vol. 27, n. 3, pp. 433-467, Septiembre, 1995.

[BENI97] D. Benítez-Diaz, "Modular Architecture for custom-built Systems Oriented to Real-Time Computer Vision: Application to Color Recognition," Journal of Systems Architecture, vol. 42, n. 9-10, pp. 709-723, Febrero 20, 1997.

[BENS85a] K. Benson, Television engineering handbook, "Chapter 14: Brodacast production equipment, systems, and services," K. Benson, B. Rayner, F. Remley and J. Roizen, pp. 14.1-14.152, New York: McGraw-Hill, 1985.

[BENS85b] K. Benson, Television engineering handbook, "Chapter 18: Digital television," E. Tarnai, pp. 18.1-14.67, New York: McGraw-Hill, 1985.

[BERT97] M. Bertozzi, A. Broggi, S. Castelluccio, "A real-time oriented system for vehicle detection," Journal of Systems Architecture, n. 43, pp. 317-325, 1997.

[BHAS94] J. Bhasker, *A VHDL primer*, Englewood Cliffs, NJ: Prentice Hall, 1994.

[BIDA99] U. Bidarte, J. Ezquerra, A. Zuloaga, J. Martin, "VHDL modeling of an adaptive and reusable architecture for real-time image enhancement," VIUF'99 VHDL International Users Forum, pp. 94-100, Orlando, USA, October 4-6, 1999.

[BLAC96] M. Black, A. Jepson, "Estimating Optical Flow in Segmented Images Using Variable-Order Parametric Models with Local Deformations," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 18, n. 10, pp. 972-984, Octubre, 1996.

[BOLC97] C. Bolchini, L. Baresi, "Software metodologies in VHDL code analysis," Journal of Systems Architecture, vol. 44, n. 1, pp. 3-21, Octubre, 1997.

[BOWM87] C. Bowman, B. Batchelor “Kiwivision a high speed architecture for machine vision,” Proceedings SPIE, vol. 849, pp. 42-51, 1987.

[BROW99] B. Brown, "Binary arithmetic," Computer science department Southern Polytechnic State University, http://www.spsu.edu/cs/faculty/bbrown/papers/arithmetic.pdf, pp. 1-9, 1999.

[CAPP99] G. Cappuchino, G. Cocorullo, P. Corsonello, S. Perri, "Educational design of high-performance arithmetic circuits on FPGA," IEEE, http://coeweb.engr.unr.edu/Nov99/12/BEGIN.HTM, Noviembre, 1999.

[CHOI96] J. Choi, S. Kim, "Multi-stage segmentation of optical flow field," Signal processing, vol. 54, n. 2, pp. 109-118, Octubre, 1996.

[CINK98] K. Cinkler. "Very low bit-rate wavelet video coding"". IEEE Journal on Selected Areas in Communications, vol. 16, no.1, pp. 4-11, Enero, 1998.

[COBO98] P. Cobos, F. Monasterio-Huelin. "FPGA implementation of th Horn & Shunk optical flow algorithm for motion detection in real time images," Proceedings XIII Design of Circuits and Integrated Systems Conference, pp. 616-621, Noviembre, 1998.

[CROO99] D. Crookes, "Architectures for high performance image processing: The future," Journal of Systems Architecture, vol. 45, n. 10, pp. 739-748, Abril 1, 1999.

[CYPR00] Cypress, CD-ROM Databook spring/summer 2000, San Jose, CA: Cypress Semiconductor, 2000.

[DATA99] “Pipeline processing illustrated,” http://www.mcbinst.com/yoko3.htm, Datacube, 1999.

[EGGL00] P. Eggleston, "Quality photos from digital video: Salient Stills' algorithms provide a missing link," Advanced Imaging, pp. 39-41, Mayo, 2000.

[ESSA97] I. Essa, A. Pentland, "Coding analysis,interpretation, and recognition of facial expressions," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, n. 7, pp. 757-763, Julio, 1997.

[FATH95] M. Fathy, M. Siyal, “A window-based edge detection technique for measuring road traffic parameters in real-time” Real Time Imaging, vol. 1, pp. 297-305, 1995.

[GARC86] E. Garcia-Calderón, Televisión, 3ra Ed., Madrid: E.T.S. de Ingenieros de Telecomunicación, 1986.

[GHOS96] S. Ghosal, P. Vanek, "A fast scalable algorithm for discontinuous optical flow estimation," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 18, n. 2, pp. 181-194, Febrero, 1996.

[GHOS97] S. Ghosal, M. Mehrotra, "Robust Optical Flow Estimation Using semi-invariant Local Features," Pattern Recognition, vol. 30, n. 2, pp. 229-237, Febrero, 1997.

[GONZ77] R. Gonzalez, P. Wintz, Digital image processing, Reading, MA: Addison-Wesley, 1977.

[GONZ96] R. González, R. Woods, *Tratamiento digital de imágenes*, Wilmington, DE: Addison-Wesley Iberoamericana, 1996.

[GONZ00] J. González, *Visión por computador*, Madrid: Paraninfo, 2000.

[GU96] H. Gu, Y. Shirai, M. Asada, "MDL-Based segmentation and motion modeling in a long image sequence of scene with multiple independiently moving objects," IEEE Transaccions on Pattern Analysis and Machine Inteligence, vol. 18, n. 1, pp. 58-65, Enero, 1996.

[GUPT95] N. Gupta, L. Kanal, "3-D motion estimation from motion field," Artificial Inteligence, vol. 78, n. 1-2, pp. 45-86, Octubre, 1995.

[HAN98] S. Han y J. Woods. "Adaptive Coding of Moving objects for very low bit rates". IEEE Journal on Selected Areas in Communications, vol. 16, no.1, pp. 56-70, Enero, 1998.

[HEUR97] V. Heuring, H. Jordan, *Computer systems design and architecture*, "Chapter 6:Computer arithmetic and the arithmetic computer unit," V. Heuring, H. Jordan, pp. 6.1-6.69, 1997.

[HO00] P. Ho, R. Chung, "Stereo-motion with stereo and motion in complement," IEEE Transaccions on Pattern Analysis and Machine Inteligence, vol. 22, n. 2, pp. 215-220, Febrero, 2000.

[HORN81] B. Horn, B. Schunck, "Determining optical flow," Artificial Intelligence, vol. 17, pp. 185-203, 1981.

[HORN86b] B. Horn, *Robot Vision*, New York: McGraw-Hill, 1986.

[HUAN95] Y. Huang, X. Zhuang, J. Cavanaugh, "Optic flow field segmentation and motion estimation using robust genetic partitioning algorithm," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 17, n. 12, pp. 1177-1190, Diciembre, 1995.

[HWAN87] K. Hwang, F. Briggs, Arquitectura de computadoras y procesamiento paralelo, Mexico: Mc Graw-Hill, 1987.

[IEEE94] IEEE *The IEEE standard VHDL language reference manual, ANSI/IEEE-Std-1076-1993,* 1994.

[IEEE95] IEEE *IEEE standard Verilog hardware description language reference manual, ANSI/IEEE-Std-1364-1995,* 1995.

[INFO97] "Television," http://www.arrakis.com.au/ais_users/tudor/history.html, Infoplease.com, 1997.

[IDT00] "IDT Home page," http://www.idt.com, Integrated Device Technology, Inc. , 2001.

[JAIN97] A. Jain, C. Dorai, "Practicing vision: integration, evaluation and applications," Pattern Recognition, vol. 30, n. 2, pp. 183-196, Febrero, 1997.

[JENS99] E. D. Jensen, "Real-time for the real word," http://www.real-time.org/no_frames/rt_concepts.htm, 1999.

[KEAR87] J. Kearney, W. Thompson, D. Boley, "Optical flow estimation: an error analysis of gradient-based methods with local optimization," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 9, n. 2, pp. 229-234, Marzo, 1987.

[KOLL93] D. Koller, K. Daniilidis, H. Nagel, "Model based object tracking in monocularimage sequences of road traffic scenes," International Journal of Computer Vision, vol. 3, pp. 625-638, 1994.

[KOTT94] D. Kotte, Y. Sung, "Motion estimation via cluster matching," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 16, n. 11, pp. 1128-1132, Noviembre, 1994.

[KRAM96] J. Kramer, "Compact Integrated Motion Sensor With Three-Pixel Interaction," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 18, n. 4, pp. 455-460, Abril, 1996.

[LACA00] M. La Cascia, S. Sclaroff, V. Athitsos, "Fast, reliable head tracking under varying illumination: an approach based on registration of texture-mapped 3D models," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, n. 4, pp. 322-347, Abril, 2000.

[LAPL96] P. Laplante, A. Stoyenko, Real-time imaging, "Chapter 2: Real time motion estimation," P. Nesi, pp. 27-57, Piscataway, NJ: IEEE Press, 1996.

[LIEB97] S. Liebman, "A no-DSP image processing board design choice: the FPGA alternative," Advanced imaging, pp. 10-12, Septiembre, 1997.

[LIMJ90] J. Lim. Two-dimensional signal and image processing. Englewood Cliffs, NJ: 1990.

[LOUG85] M. Lougheed, "A high speed recirculating neighbourhood processing architecture," Proceedings SPIE, vol 534, pp. 22-33, 1985.

[LUCK86] R. Luck, "Using PIPE for inspection applications," Proceedings SPIE, vol. 730, pp. 12-19, 1986.

[MANO93] M. Mano, *Arquitectura de computadoras*, 3ra Ed., México: Prentice Hall, 1993.

[MAZO00] B. Mazor. "Roundtable discussion: The CMOS sensor: what lies ahead," http://www.advancedimagingmag.com/jun2k/cmos.htm, Advanced Imaging magazine on internet, June, 2000.

[MCBA00] "Yokogawa CSU10. How does it work?," http://www.mcbinst.com/yoko3.htm, McBain Instruments, 2000.

[MCCU85] D. McCubbrey, M. Lougheed, "Morphological image analysis using a raster pipeline processor," IEEE workshop on computer architectures for pattern analysis and image database management, 1985.

[MEMI98] E. Mémin, P. Pérez, "Dense estimation and object-based segmentation of the optical flow with robust techniques," Pattern Analysis on image processing, vol. 7, n. 5, pp. 703-719, Mayo, 1998.

[MITI88] A. Mitiche, R. Grisell, J. Aggarwal, "On Smoothness of a vector field: applications to optical flow," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 10, n. 6, pp. 943-949,

Noviembre, 1988.

[MODE97] Model Technology, ModelSim PE/PLUS User's Manual, Oregon: Model Technology Incorporated, 1997.

[MUEL97] S. Mueller, "Validate image-processing device models with a VHDL test environment," Electronic Design, vol. 45, n. 22, pp. 59-64, Octubre, 1997.

[NAGE92] H. Nagel, G. Orban, Artificial an biological vision systems, "Direct estimation of optical flow and its derivates," H. Nagel, pp. 193-224, Berlin: Springer Verlag, 1992.

[NAM99] J. Nam, "What is the CCD Image Sensor?," http://smdl.snu.ac.kr/~jhnam/ccd/imgsen.htm, Semiconductors materials and devices laboratory, Seoul National University, 1997.

[NCS96] National Communication Systems: Technology & Standards Division. *Telecommunications: Glossary of telecommunication terms (Federal Standard 1037C)*, http://www.its.bldrdoc.gov/fs-1037, General Services Administration Information Technology Service, 1996.

[NEGA98] S. Negahdaripour, "Revised definition of optical flow: integration of ratiometric and geometric cues for dynamic scene analysis.," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, n. 9, pp. 961-979, Septiembre, 1998.

[NESI95] P. Nesi, A. Del Bimbo, D. Ben-Tzvi, "A Robust algorithm for optical flow estimation," Computer vision and image understanding, vol. 26, n. 1, pp. 59-68, Julio, 1995.

[NG98] L. Ng, V. Solo, "Choosing the optimal neighbourhood size in optical flow problems with errors-in-variables modelling," Proceedings of 1998 IEEE International Conference on Image Processing, Octubre, 1998.

[NOZA91] L. Nozal, S. Lorenzo, R. Boucho, M Shaban, ""Real time and low cost image processing architecture based on programable logic devices,"" IEEE/RSJ International Workshop on Intelligent Robots and Systems, pp. 279-284, Noviembre, 1991.

[NOZA93] L. Nozal, S. Lorenzo, "Arquitectura para procesamiento de imágenes - Criterios estructurales de diseño," Mundo Electrónico, n. 236, pp. 42-48, Marzo, 1993.

[NOZA97] L. Nozal, G. Aranguren, J. Martín, J. Ezquerra, "Moving images time gradient implementation using RAM-based FPGA," SPIE procceedings of real-time imaging II, vol. 3028, pp. 108-116, Febrero 10, 1997.

[OFS00] "Optical flow systems," http://www.ofs.co.uk/main/index.html, Optical Flow Systems Ltd., 2000.

[OMON94] A. Omondi, *Computer arithmetic systems: algorithms, architecture and implementation*, Cambridge: Prentice Hall, 1994.

[ONG99] E. Ong, M. Spann, "Robust optical flow estimation based on least-median-of-squares regression," International Journal of Computer Vision, vol. 31, n. 1, pp. 51-82, Enero, 1999.

[OTTE95] M. Otte, H. Nagel, "Estimation of optical flow based on higher-order spatiotemporal derivatives in interlaced and non-interlaced image sequences," Artificial Inteligence, vol. 78, n. 1-2, pp. 5-43, Octubre, 1995.

[PANJ95] D. Panjwani, G. Healey. "Markov random field models for unsupervised segmentation of textured color images," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 17, n. 10, pp. 939-954, Octubre, 1995.

[PARA00] N. Paragios, R. Deriche, "Geodesic active contours and level sets for the detection and tracking of moving objects," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, n. 3, pp. 266-280, Marzo, 2000.

[PARD94] M. Pardàs, P. Salembier, "3D morphological segmentation and motion estimation from image secuences," Signal Processing, vol. 38, no. 1, pp. 31-43, Julio, 1994.

[PARD98] F. Pardo, B. Dierickhx, D. Scheffer, "Space-variant non-orthohonal structure CMOS image sensor design," IEEE Journal of Solid State Circuits, vol. 33, pp. 843-849, 1998.

[PETE98] J. Peters, "A history of television," http://www.ebu.ch/dvb/dvb_articles/dvb_tv-history.htm, 1998.

[QUAN98] "Digital fact book: 8 bits or 10 bits?," http://www.quantel.com/dfb/eightbitsorten.htm, Quantel,

1998.

[SAHA00] S. Saha, "Image compression - from DCT to wavelets: a review," ACM Crossroads, (http://www.acm.org/crossroads/xrds6-3/sahaimgcoding.html), Marzo, 2000.

[SANZ98] C. Sanz, "Arquitecturas VLSI para la estimación de movimiento en codificación de imágenes," Tesis Doctoral, Universidad Politécnica de Madrid, 1998.

[SCHU89] B. Schunck, "Image flow segmentation and estimation by constraint line clustering," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 11, n. 10, pp. 1010-1027, Octubre, 1989.

[SCHW95] H. Schweitzer, "Occan Algorithms for Computing Visual Motion," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 17, n. 11, pp. 1033-1042, Noviembre, 1995.

[SHAR90] H. Shariat, K. Price, "Motion estimation with more than two frames," IEEE Transaccions on Pattern Analysis and Machine Inteligence, vol. 12, n. 5, pp. 417-434, Mayo, 1990.

[SILB99] M. Silbergleid, M. J. Pescatore, *The Guide To Digital Television*, http://digitaltelevision.com/dtvbook/toc.shtml, New York: Miller Freeman PSN Inc., 1999

[SKAH95] K. Skahill, *VHDL for programmable logic*, San Jose, CA: Cypress Semiconductor Corporation, 1995.

[SMIT96] D. Smith, "HDL basic training: top-down chip design using Verilog and VHDL," EDN Magazine, 26 Octubre, 1996.

[SOAT98a] S. Soatto, P. Perona, "Reducing "structure from motion" a general framework for dynamic vision. Part I: modeling," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, n. 9, pp. 933-942, Septiembre, 1998.

[SOAT98b] S. Soatto, P. Perona, "Reducing "structure from motion" a general framework for dynamic vision. Part II: implementation and experimental assessment," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, n. 9, pp. 943-960, Septiembre, 1998.

[STOR89] N. Storey, R. Staunton, "An adaptive pipeline processor for real time image processing," Proceedings SPIE, vol. 1197, pp. 238-246, 1989.

[TAYL95] C. Taylor, D. Kriegman, "Structure and motion from line segments in multiple images," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 17, n. 11, pp. 1021-1032, Noviembre, 1995.

[THOM98] W. Thompson, "Exploiting discontinuities in optical flow," International Journal of Computer Vision, vol. 30, n. 3, pp. 163-173, 1998.

[THIN99] “Mechanics: producing a motion picture,” http://library.thinkquest.org/10015/dat/info/reference/mechanics/, ThinkQuest, 1999.

[TIST94] M. Tistarelly, G. Sandini, "On the advantages of polar and log-polar mapping for direct estimation of time-to-impact from optical flow," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 15, n. 4, pp. 401-410, Abril, 1994.

[TIST96] M. Tistarelli, "Multiple constraints to compute optical flow," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 18, n. 12, pp. 1243-1250, Diciembre, 1996.

[UCHI94] T. Uchiyama, M. Arbib. “Color image segmentation using competitive learning,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 16, no. 12, pp. 1197-1206, Diciembre, 1994.

[VERR89] A. Verri, T. Poggio, "Motion field and optical flow: qualitative properties," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 11, n. 5, pp. 490-498, Mayo, 1989.

[VICE90] M. Vicens, J. Albert, V. Arnau, V. Cerverón, G. Fabregat, "Tratamiento digital de imágenes: técnicas básicas," Mundo Electrónico, n. 208, Julio, 1990.

[VIDE98] “Mixing video sources: digital video,” http://www.videonics.com/articles/mixing.html, Videonics Inc., 1998.

[VISE99] G. Visentin, F. Didot, "Testing space robotics on the japanese ETS-VII satellite," ESA bulletin, n. 99, pp. 61-65, Septiembre, 1999.

[WERK90] P. Werkhoven, J. Koenderink, "Extraction of motion parallax structure in the visual system," Biological cybernetics, vol. 63, pp. 185-199, 1990.

[XIE96] K. Xie, L. Van Eycken, A. Oosterlinck, "Hierarchical motion estimation with smoothness constrains and postprocessing," Optical engineering, vol. 35, n. 1, pp. 145-155, Enero, 1996.

[XILI99] Xilinx, The programmable Logic Data Book 1999, San Jose, CA: Xilinx, Inc., 1998.

[ZHAN95] Z. Zhang. “Estimating motion and structure from correspondence of line segments between two perspective images.” IEEE Transactions on pattern analysis and machine inteligence, vol. 17, n. 12, pp. 1129-1139, Diciembre, 1995.

[ZULO99a] A. Zuloaga, J. Ezquerra, U. Bidarte, J. Martin, "VHDL simulation tool for designing image processing systems," DCIS'99 Proceedings of XIV Desing of Circuits and Systems Conference, pp. 275-280, Palma de Mallorca, Noviembre 16-19, 1999.

[ZULO99b] A. Zuloaga, J. Martin, U. Bidarte, J. Ezquerra, "VHDL test bench for digital image processing systems using a new image format," FDL'99 Second international forum on design languages proceedings, pp. 187-196, Lyon, Septiembre 1-3, 1999.

Apéndices

Apéndice A. Especificaciones técnicas de memorias FIFO

CMOS ASYNCHRONOUS FIFO
2,048 x 9, 4,096 x 9
8,192 x 9, 16,384 x 9
32,768 x 9, 65,536 x 9

IDT7203
IDT7204
IDT7205
IDT7206
IDT7207
IDT7208

FEATURES:

- First-In/First-Out Dual-Port memory
- 2,048 x 9 organization (IDT7203)
- 4,096 x 9 organization (IDT7204)
- 8,192 x 9 organization (IDT7205)
- 16,384 x 9 organization (IDT7206)
- 32,768 x 9 organization (IDT7207)
- 65,636 x 9 organization (IDT7208)
- High-speed: 12ns access time
- Low power consumption
  - *Active: 660mW (max.)*
  - *Power-down: 44mW (max.)*
- Asynchronous and simultaneous read and write
- Fully expandable in both word depth and width
- Pin and functionally compatible with IDT720X family
- Status Flags: Empty, Half-Full, Full
- Retransmit capability
- High-performance CMOS technology
- Military product compliant to MIL-STD-883, Class B
- Standard Military Drawing for #5962-88669 (IDT7203), 5962-89567 (IDT7203), and 5962-89568 (IDT7204) are listed on this function
- Industrial temperature range (–40°C to +85°C) is available (plastic packages only)

DESCRIPTION:

The IDT7203/7204/7205/7206/7207/7208 are dual-port memory buffers with internal pointers that load and empty data on a first-in/first-out basis. The device uses Full and Empty flags to prevent data overflow and underflow and expansion logic to allow for unlimited expansion capability in both word size and depth.

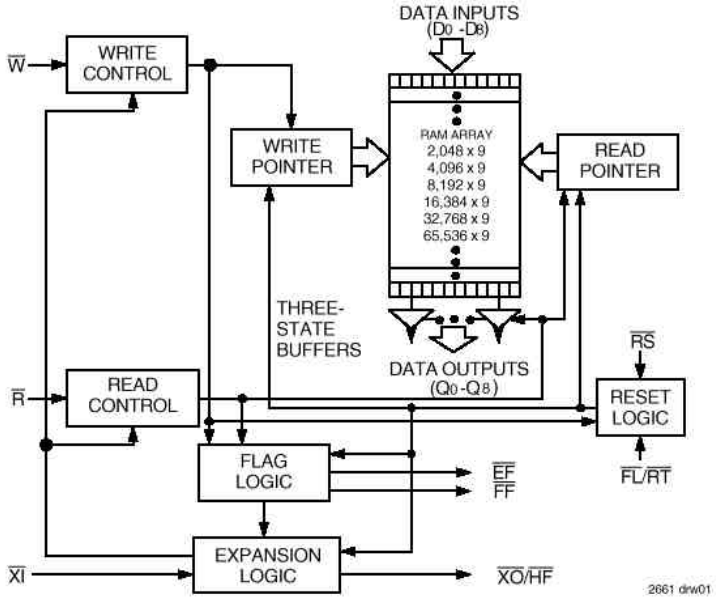
Data is toggled in and out of the device through the use of the Write ($\overline{W}$) and Read ($\overline{R}$) pins.

The device's 9-bit width provides a bit for a control or parity at the user's option. It also features a Retransmit ($\overline{RT}$) capability that allows the read pointer to be reset to its initial position when $\overline{RT}$ is pulsed LOW. A Half-Full Flag is available in the single device and width expansion modes.

These FIFOs are fabricated using IDT's high-speed CMOS technology. They are designed for applications requiring asynchronous and simultaneous read/writes in multiprocessing, rate buffering and other applications.

Military grade product is manufactured in compliance with the latest revision of MIL-STD-883, Class B.

FUNCTIONAL BLOCK DIAGRAM



The IDT logo is a registered trademark of Integrated Device Technology, Inc.

© 2001 Integrated Device Technology, Inc.

SIGNAL DESCRIPTIONS

INPUTS:

DATA IN (D_0–D_8) — Data inputs for 9-bit wide data.

CONTROLS:

RESET ($\overline{RS}$) — Reset is accomplished whenever the Reset ($\overline{RS}$) input is taken to a LOW state. During reset, both internal read and write pointers are set to the first location. A reset is required after power-up before a write operation can take place. **Both the Read Enable ($\overline{R}$) and Write Enable ($\overline{W}$) inputs must be in the HIGH state during the window shown in Figure 2 (i.e. t_{RSS} before the rising edge of $\overline{RS}$) and should not change until t_{RSR} after the rising edge of $\overline{RS}$.**

WRITE ENABLE ($\overline{W}$) — A write cycle is initiated on the falling edge of this input if the Full Flag ($\overline{FF}$) is not set. Data set-up and hold times must be adhered-to, with respect to the rising edge of the Write Enable ($\overline{W}$). Data is stored in the RAM array sequentially and independently of any on-going read operation.

After half of the memory is filled, and at the falling edge of the next write operation, the Half-Full Flag ($\overline{HF}$) will be set to LOW, and will remain set until the difference between the write pointer and read pointer is less-than or equal to one-half of the total memory of the device. The Half-Full Flag ($\overline{HF}$) is then reset by the rising edge of the read operation.

To prevent data overflow, the Full Flag ($\overline{FF}$) will go LOW on the falling edge of the last write signal, which inhibits further write operations. Upon the completion of a valid read operation, the Full Flag ($\overline{FF}$) will go HIGH after t_{RFF}, allowing a new valid write to begin. When the FIFO is full, the internal write pointer is blocked from $\overline{W}$, so external changes in $\overline{W}$ will not affect the FIFO when it is full.

READ ENABLE ($\overline{R}$) — A read cycle is initiated on the falling edge of the Read Enable ($\overline{R}$), provided the Empty Flag ($\overline{EF}$) is not set. The data is accessed on a First-In/First-Out basis, independent of any ongoing write operations. After Read Enable ($\overline{R}$) goes HIGH, the Data Outputs (Q_0 through Q_8) will return to a high-impedance condition until the next Read operation. When all the data has been read from the FIFO, the Empty Flag ($\overline{EF}$) will go LOW, allowing the "final" read cycle but inhibiting further read operations, with the data outputs remaining in a high-impedance state. Once a valid write operation has been accomplished, the Empty Flag ($\overline{EF}$) will go HIGH after t_{WEF} and a valid Read can then begin. When the FIFO is empty, the internal read pointer is blocked from $\overline{R}$ so external changes will not affect the FIFO when it is empty.

FIRST LOAD/RETRANSMIT ($\overline{FL}/\overline{RT}$) — This is a dual-purpose input. In the Depth Expansion Mode, this pin is grounded to indicate that it is the first device loaded (see Operating Modes). The Single Device Mode is initiated by grounding the Expansion In ($\overline{XI}$).

The IDT7203/7204/7205/7206/7207/7208 can be made to retransmit data when the Retransmit Enable Control ($\overline{RT}$) input is pulsed LOW. A retransmit operation will set the internal read pointer to the first location and will not affect the write pointer. The status of the Flags will change depending on the relative locations of the read and write pointers. Read Enable ($\overline{R}$) and Write Enable ($\overline{W}$) must be in the HIGH state during retransmit. This feature is useful when less than 2,048/4,096/8,192/16,384/32,768/65,536 writes are performed between resets. The retransmit feature is not compatible with the Depth Expansion Mode.

EXPANSION IN ($\overline{XI}$) — This input is a dual-purpose pin. Expansion In ($\overline{XI}$) is grounded to indicate an operation in the single device mode. Expansion In ($\overline{XI}$) is connected to Expansion Out ($\overline{XO}$) of the previous device in the Depth Expansion or Daisy-Chain Mode.

OUTPUTS:

FULL FLAG ($\overline{FF}$) — The Full Flag ($\overline{FF}$) will go LOW, inhibiting further write operations, when the device is full. If the read pointer is not moved after Reset ($\overline{RS}$), the Full Flag ($\overline{FF}$) will go LOW after 2,048/4,096/8,192/16,384/32,768/65,536 writes.

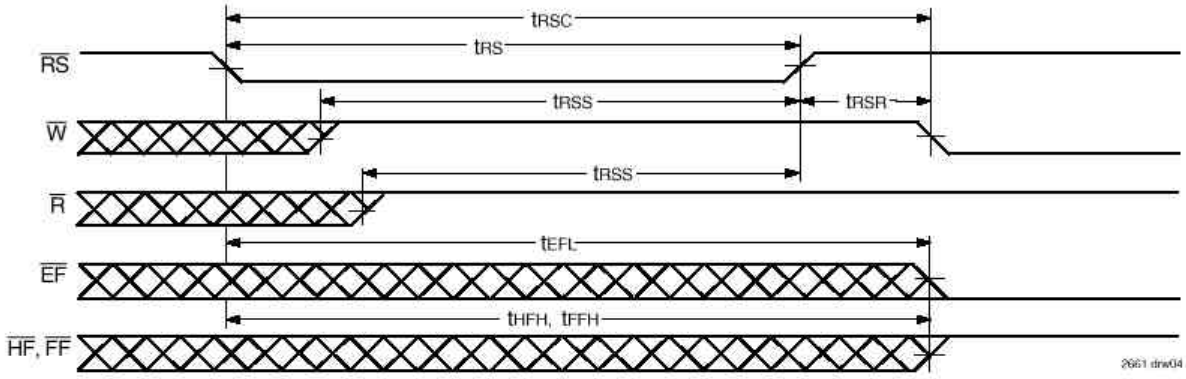
EMPTY FLAG ($\overline{EF}$) — The Empty Flag ($\overline{EF}$) will go LOW, inhibiting further read operations, when the read pointer is equal to the write pointer, indicating that the device is empty.

EXPANSION OUT/HALF-FULL FLAG ($\overline{XO}/\overline{HF}$) — This is a dual-purpose output. In the single device mode, when Expansion In ($\overline{XI}$) is grounded, this output acts as an indication of a half-full memory.

After half of the memory is filled, and at the falling edge of the next write operation, the Half-Full Flag ($\overline{HF}$) will be set to LOW and will remain set until the difference between the write pointer and read pointer is less than or equal to one half of the total memory of the device. The Half-Full Flag ($\overline{HF}$) is then reset by the rising edge of the read operation.

In the Depth Expansion Mode, Expansion In ($\overline{XI}$) is connected to Expansion Out ($\overline{XO}$) of the previous device. This output acts as a signal to the next device in the Daisy Chain by providing a pulse to the next device when the previous device reaches the last location of memory. There will be an $\overline{XO}$ pulse when the Write pointer reaches the last location of memory, and an additional $\overline{XO}$ pulse when the Read pointer reaches the last location of memory.

DATA OUTPUTS (Q_0-Q_8) — Q_0-Q_8 are data outputs for 9-bit wide data. These outputs are in a high-impedance condition whenever Read ($\overline{R}$) is in a HIGH state.



NOTE:
1. $\overline{W}$ and $\overline{R}$ = V_{IH} around the rising edge of $\overline{RS}$.

Figure 2. Reset

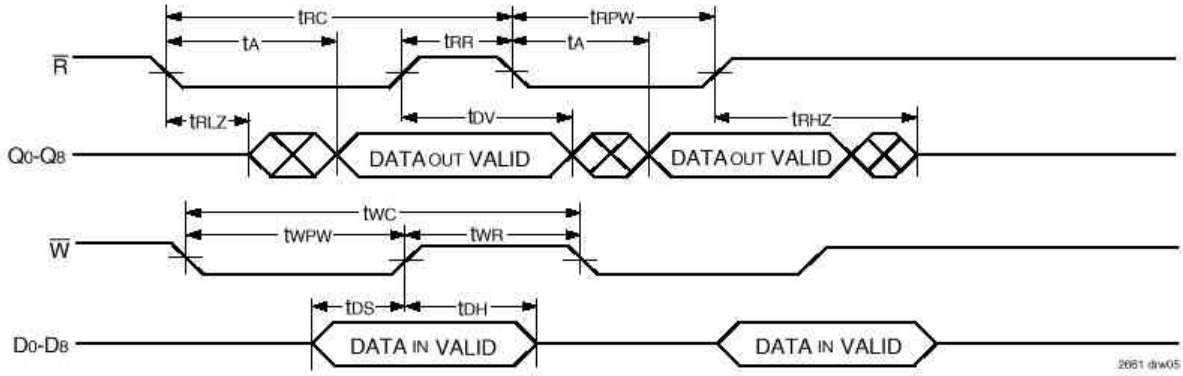


Figure 3. Asynchronous Write and Read Operation

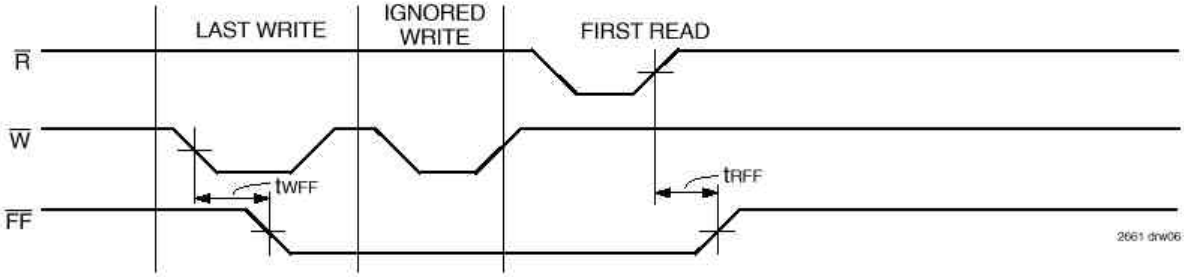


Figure 4. Full Flag Timing From Last Write to First Read

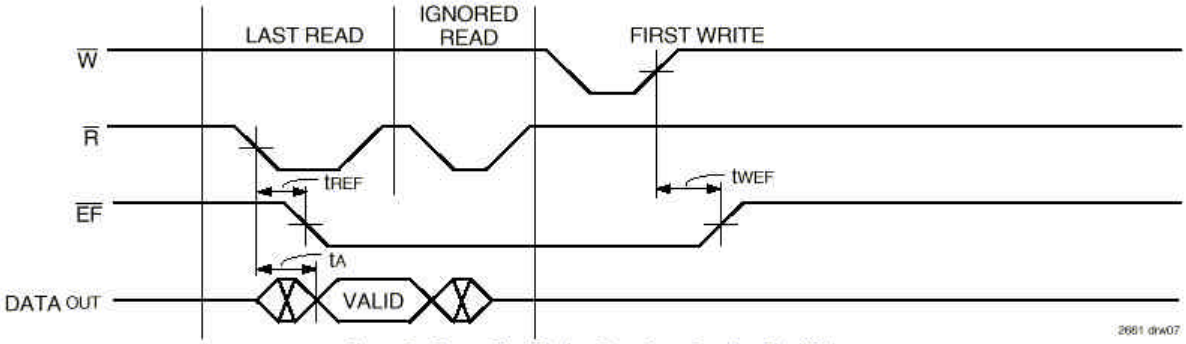
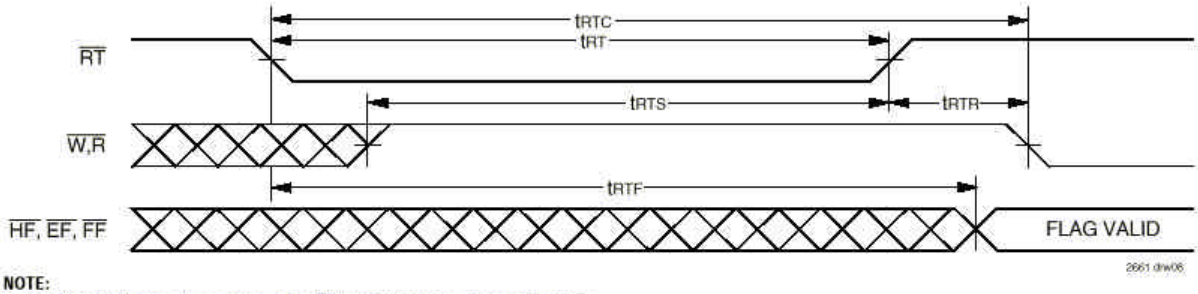


Figure 5. Empty Flag Timing From Last Read to First Write



NOTE:
1. EF, FF and HF may change status during Retransmit, but flags will be valid at tRTC.

Figure 6. Retransmit

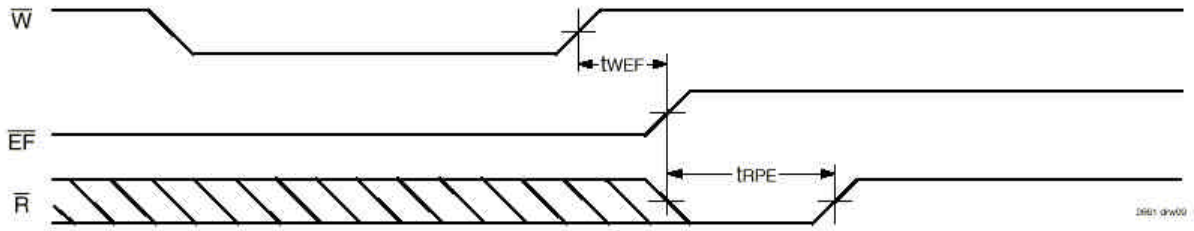


Figure 7. Minimum Timing for an Empty Flag Coincident Read Pulse.

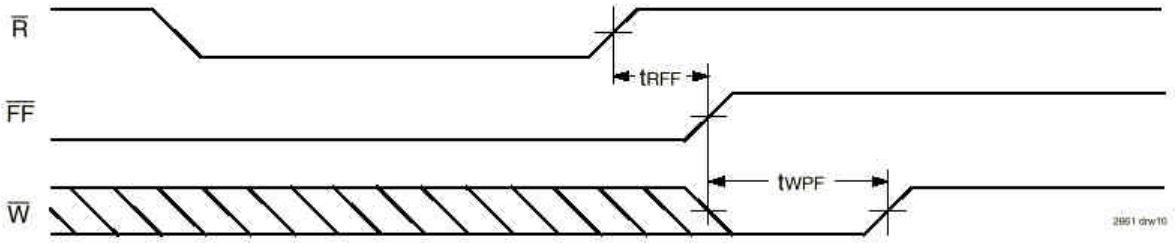


Figure 8. Minimum Timing for a Full Flag Coincident Write Pulse.

Apéndice B. Descripciones VHDL de los módulos

B1. CDTERM: Calculador de término D

```
--------------------------------------------------------------------------
-- CDTERM                                                               --
-- CALCULADOR DE TERMINO D  (Ex^2 + Ey^2 + L^2)                         --
--                                                                      --
-- Algoritmo:                                                           --
--   *Este modulo calcula el termino D, que es básicamente el           --
--    denominador de la ecuacion de flujo optico.                       --
--   *El valor de lambda (L) será cero siempre que Ex^2 + Ey^2 sea      --
--    diferente de cero, en caso contrario L se hará 1 para salvar      --
--    la indeterminación que produciría. Básicamente esto da un peso    --
--    a la parte de estimación cuando la ecuación de flujo óptico       --
--    se hace inoperante.                                               --
--   *Se considera el signo de los gradientes para todos los cálculos   --
--                                                                      --
-- Autor: Aitzol Zuloaga I.                                             --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea    --
--        Departamento de electronica y telecomunicaciones              --
--                                                                      --
-- Fecha: 31/05/97                                                      --
--------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;

------------------------------------------------------------------------

ENTITY cdterm IS

  PORT(
    SIGNAL ex: IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0)                   ; -- Gradiente x
    SIGNAL ey: IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0)                   ; -- Gradiente y
    SIGNAL clk: IN  STD_LOGIC                                       ; -- reloj de operacion
    SIGNAL d:  OUT STD_LOGIC_VECTOR (16 DOWNTO 0) := (OTHERS=>'0')) ; -- Término d: (Ex^2 + Ey^2 + L^2)

END cdterm;

------------------------------------------------------------------------

ARCHITECTURE struct OF cdterm IS

  COMPONENT multsgn3
    PORT(
      SIGNAL m:  IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0) ; -- Multiplicando
      SIGNAL rm: OUT STD_LOGIC_VECTOR (16 DOWNTO 0)); -- Resultado de elevar al cuadrado
  END COMPONENT;

  CONSTANT ule:  INTEGER := ex'HIGH    ; -- Límite superior de bits de ex
  CONSTANT uld:  INTEGER := d'HIGH     ; -- Límite superior de bits de d
  CONSTANT ule2: INTEGER := 2 * ule    ; -- Límite superior de bits de ex^2

  CONSTANT gnd:  STD_LOGIC_VECTOR (uld DOWNTO 0):= (OTHERS=>'0'); --"00...0"

  SIGNAL   ex2:  STD_LOGIC_VECTOR (ule2 DOWNTO 0) ; -- ex^2
  SIGNAL   ey2:  STD_LOGIC_VECTOR (ule2 DOWNTO 0) ; -- ey^2
  SIGNAL   dp:   STD_LOGIC_VECTOR ( uld DOWNTO 0) ; -- resultado: Ex^2 + Ey^2
  SIGNAL   d_a:  STD_LOGIC_VECTOR (  16 DOWNTO 0) ; -- Término d auxiliar  (Ex^2 + Ey^2 + L^2)

BEGIN

  -- Se calculan los cuadrados de los gradientes
  uex2: multsgn3 PORT MAP (ex,ex2); --  ex2 <= exe * exe;
  uey2: multsgn3 PORT MAP (ey,ey2); --  ey2 <= eye * eye;

  -- Se calcula la suma de los cuadrados
  dp  <=  ex2 + ey2;

  -- El resultado nunca debe ser cero en caso de que ex^2+ey^2=0 entonces
  -- se le agrega el factor de correccion L^2 de valor 1
  d_a  <= dp(uld DOWNTO 0) WHEN (dp /= gnd) ELSE gnd(uld-1 DOWNTO 0) & "1";

  -- Se sincroniza el resultado
  PROCESS (clk)
    BEGIN
      IF clk= '1' THEN
        d <= d_a;
      END IF;
  END PROCESS;

END;
```

B2. CONVER: Conversor

```
--------------------------------------------------------------------------
-- CONVER                                                               --
-- CONVERSOR DE NIVEL DE VELOCIDAD PARA COMPATIBILIDAD CON MONITOR      --
--                                                                      --
-- Algoritmo:                                                           --
--   *Este módulo convierte la velocidad de su formato en modulo 2      --
--    a una versión sin signo comprendida entr 0 y 255, donde 0         --
--    significaría una velocidad máxima hacia la izquierda y            --
--    255 una vdad resultante se calcula con 7 bits más signo           --
--                                                                      --
-- Autor: Aitzol Zuloaga I.                                             --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea    --
--        Departamento de electrónica y telecomunicaciones              --
--                                                                      --
-- Fecha: 11/07/98                                                      --
--------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;

--------------------------------------------------------------------------
ENTITY conver IS
  PORT(
    SIGNAL vi:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ; -- Velocidad en módulo 2
    SIGNAL clk: IN  STD_LOGIC                      ; -- reloj de operacion
    SIGNAL vc:  OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)); -- Velocidad con offset
END conver;

--------------------------------------------------------------------------

ARCHITECTURE struct OF conver IS
BEGIN

  -- Se sincroniza el resultado
  PROCESS (clk)
    BEGIN
      IF clk= '1' THEN
        vc <= vi+"10000000";
      END IF;
  END PROCESS;


END;
```

B3. CPROMED: Calculador de coeficientes de promediado

```
--------------------------------------------------------------------------
-- CPROMED                                                              --
-- Este modulo calcula los coeficientes de promediado de una imagen     --
-- que llega pixel a pixel.                                             --
--                                                                      --
-- Algoritmo:                                                           --
--   *Este módulo calcula los coeficientes segun la siguiente mascara   --
--    1/12    1/6     1/12                                              --
--    1/6      -1     1/6                                               --
--    1/12    1/6     1/12                                              --
--   *Las entradas estan ordenadas como sigue                           --
--     v22    v21     v20                                               --
--     v12    v11     v01                                               --
--     v02    v01     v00                                               --
--                                                                      --
--                                                                      --
-- Autor: Aitzol Zuloaga I.                                             --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea    --
--        Departamento de electronica y telecomunicaciones              --
--                                                                      --
-- Fecha: 30/12/98                                                      --
--------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
--USE ieee.std_logic_signed.ALL;

LIBRARY arithmetic;
USE arithmetic.std_logic_arith.ALL;
--------------------------------------------------------------------------

ENTITY cpromed IS

  PORT(
    SIGNAL v00:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ;
    SIGNAL v01:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ;
    SIGNAL v02:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ;
    SIGNAL v10:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ;
    SIGNAL v12:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ;
    SIGNAL v20:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ;
    SIGNAL v21:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ;
    SIGNAL v22:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ;
    SIGNAL clk:  IN  STD_LOGIC                           ; -- reloj de operacion
    SIGNAL lap:  OUT STD_LOGIC_VECTOR ( 8 DOWNTO 0)     ); -- salida laplaciano

END cpromed;
```

```
-------------------------------------------------------------------

ARCHITECTURE struc OF cpromed IS

  COMPONENT divip03
    PORT(
      n:   IN  STD_LOGIC_VECTOR (10 DOWNTO 0)               ; -- numerador
      clk: IN  STD_LOGIC                                    ; -- reloj de operacion
      r:   OUT STD_LOGIC_VECTOR (9 DOWNTO 0):=(OTHERS=>'0')) ; -- resultado  r=n/d
  END COMPONENT;

  SIGNAL ssa: STD_LOGIC_VECTOR ( 8 DOWNTO 0) ;
  SIGNAL ssb: STD_LOGIC_VECTOR ( 8 DOWNTO 0) ;
  SIGNAL ssc: STD_LOGIC_VECTOR ( 8 DOWNTO 0) ;
  SIGNAL ssd: STD_LOGIC_VECTOR ( 8 DOWNTO 0) ;
  SIGNAL sse: STD_LOGIC_VECTOR ( 9 DOWNTO 0) ;
  SIGNAL ssf: STD_LOGIC_VECTOR ( 9 DOWNTO 0) ;
  SIGNAL ssg: STD_LOGIC_VECTOR (10 DOWNTO 0) ;
  SIGNAL ssh: STD_LOGIC_VECTOR (10 DOWNTO 0) ;
  SIGNAL ssi: STD_LOGIC_VECTOR ( 9 DOWNTO 0) ;
  SIGNAL ssj: STD_LOGIC_VECTOR ( 8 DOWNTO 0) ;
  SIGNAL ssk: STD_LOGIC_VECTOR (10 DOWNTO 0) ;
  SIGNAL ssl: STD_LOGIC_VECTOR ( 9 DOWNTO 0) ;
  SIGNAL ssm: STD_LOGIC_VECTOR ( 8 DOWNTO 0) ;
BEGIN


  ssa <= (v00(7) & v00) + (v02(7) & v02);          -- v00+v02
  ssb <= (v20(7) & v20) + (v22(7) & v22);          -- v20+v22

  ssc <= (v12(7) & v12) + (v10(7) & v10);          -- v12+v10
  ssd <= (v01(7) & v01) + (v21(7) & v21);          -- v01+v21

  sse <= (ssa(8) & ssa) + (ssb(8) & ssb);          -- v00+v02+v20+v22
  ssf <= (ssc(8) & ssc) + (ssd(8) & ssd);          -- v12+v10+v01+v21

  ssg <= (sse(9) & sse) + "00000000001";           -- se suma 1 para redondeo decimal
  ssh <= (ssf(9) & ssf) + "00000000010";           -- se suma 2 para redondeo decimal

  ssi <= ssg(10 DOWNTO 1);                         -- (v00+v02+v20+v22)/2 division entre 2
  ssj <= ssh(10 DOWNTO 2);                         -- (v12+v10+v01+v21)/4 division entre 4

  ssk <= (ssi(9) & ssi) + (ssj(8) & ssj(8) & ssj); -- (v00+v02+v20+v22)/2+(v12+v10+v01+v21)/4 suma total antes
de la división por 3

  udivip03:
  divip03 PORT MAP (ssk,clk,ssl);                  -- (v00+v02+v20+v22)/6+(v12+v10+v01+v21)/12

--  ssg <= (sse(9) & sse( 9 DOWNTO 1)) + ("00" & ssf(9 DOWNTO 2));
--  ssh <= ssg / "0000000011";

  -- Se trunca el resultado a 7 bits más signo
  ssm <= "011111111" WHEN ssl(9 DOWNTO 8)="01" ELSE
         "100000001" WHEN ssl(9 DOWNTO 8)="10" ELSE
         ssl(8 DOWNTO 0);

  -- Se sincroniza el resultado
  PROCESS (clk)
    BEGIN
      IF clk= '1' THEN
        lap <= ssm;
      END IF;
  END PROCESS;

END;
```

B4. CPTERM: Calculador de términos P

```
-------------------------------------------------------------------------
-- CPTERM                                                              --
-- CALCULADOR DE TERMINOS P  Ex*(ExU + EyV + Et) y Ey*(ExU + EyV + Et)--
--                                                                     --
-- Algoritmo:                                                          --
--   *Este módulo calcula los terminos Px y Py, que son básicamente    --
--    los numeradores de la ecuacion de flujo optico.                  --
--   *Se considera el signo de los gradientes para todos los cálculos  --
--    por ello se efectúa una operación de extensión del tamaño        --
--    de los operandos que incluye el signo                            --
--                                                                     --
-- Autor: Aitzol Zuloaga I.                                            --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea   --
--        Departamento de electrónica y telecomunicaciones             --
--                                                                     --
-- Fecha: 20/06/98                                                     --
-------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;

-------------------------------------------------------------------------

ENTITY cpterm IS

  PORT(
    SIGNAL ex:  IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0)       ; -- Gradiente x
    SIGNAL ey:  IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0)       ; -- Gradiente y
```

```
    SIGNAL et:  IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0)       ; -- Gradiente t
    SIGNAL vpx: IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0)       ; -- Velocidad promedio u (horizontal)
    SIGNAL vpy: IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0)       ; -- Velocidad promedio v (vertical)
    SIGNAL clk: IN  STD_LOGIC                            ; -- reloj de operacion
    SIGNAL px:  OUT STD_LOGIC_VECTOR (25 DOWNTO 0) :=(OTHERS=>'0') ; -- Término px Ex*(ExU + EyV + Et)
    SIGNAL py:  OUT STD_LOGIC_VECTOR (25 DOWNTO 0) :=(OTHERS=>'0')); -- Término py Ey*(ExU + EyV + Et)

END cpterm;

-----------------------------------------------------------------------

ARCHITECTURE struct OF cpterm IS

  COMPONENT multsgn1
    PORT(
      SIGNAL m1: IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0) ; -- Primer  multiplicando
      SIGNAL m2: IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0) ; -- Segundo multiplicando
      SIGNAL rm: OUT STD_LOGIC_VECTOR (17 DOWNTO 0)); -- Multiplicacion
  END COMPONENT;

  COMPONENT multsgn2
    PORT(
      SIGNAL m1: IN  STD_LOGIC_VECTOR (17 DOWNTO 0) ; -- Primer  multiplicando
      SIGNAL m2: IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0) ; -- Segundo multiplicando
      SIGNAL rm: OUT STD_LOGIC_VECTOR (25 DOWNTO 0)); -- Multiplicacion
  END COMPONENT;

  CONSTANT ulp: INTEGER := px'HIGH;

  SIGNAL p:    STD_LOGIC_VECTOR (17 DOWNTO 0)       ; -- Término p (ExU + EyV + Et)
  SIGNAL spa:  STD_LOGIC_VECTOR (17 DOWNTO 0)       ; -- producto a: ExU
  SIGNAL spb:  STD_LOGIC_VECTOR (17 DOWNTO 0)       ; -- producto b: EyV
  SIGNAL spc:  STD_LOGIC_VECTOR (17 DOWNTO 0)       ; -- sumatorio: ExU + EyV
  SIGNAL px_a: STD_LOGIC_VECTOR (25 DOWNTO 0)       ; -- Término px auxiliar Ex*(ExU + EyV + Et)
  SIGNAL py_a: STD_LOGIC_VECTOR (25 DOWNTO 0)       ; -- Término py auxiliar Ey*(ExU + EyV + Et)

BEGIN

  -- Se calcula la ecuación del termino P
  uspa: multsgn1 PORT MAP (ex,vpx,spa); --spa <= exe * vxpe ;
  uspb: multsgn1 PORT MAP (ey,vpy,spb); --spb <= eye * vype ;

  spc <= spa + spb ;
  p   <= spc + et ;

  -- Se multimplica por Ex y Ey
  cpxt: multsgn2 PORT MAP (p,ex,px_a);
  cpyt: multsgn2 PORT MAP (p,ey,py_a);

  -- Se sincroniza el resultado
  PROCESS (clk)
    BEGIN
      IF clk= '1' THEN
        px <= px_a;
        py <= py_a;
      END IF;
  END PROCESS;

END;
```

B5. CTRFIFO: Control de FIFOs

```
-----------------------------------------------------------------------
-- CTRFIFO                                                           --
-- GENERACION DE SEÑALES DE RELOJ PARA EL CONTROL DE FIFOS           --
--                                                                   --
-- Algoritmo:                                                        --
--   El reloj de salida para las FIFO se retrasa tantas unidades     --
--   de tiempo como bytes se desea almacenar en la FIFO              --
--                                                                   --
-- Autor: Aitzol Zuloaga I.                                          --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea --
--        Departamento de electrónica y telecomunicaciones           --
--                                                                   --
-- Fecha: 09/12/98                                                   --
-----------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;

-----------------------------------------------------------------------

ENTITY ctrfifo IS

  GENERIC(
    CONSTANT capf: IN  INTEGER                      :=  256); -- Capacidad de la FIFO

  PORT(
    SIGNAL clk:   IN  STD_LOGIC                           ; -- Señal auxiliar de reloj
    SIGNAL n_rst: IN  STD_LOGIC                     :='1' ; -- Señal auxiliar de reset
    SIGNAL clks:  OUT STD_LOGIC                     :='1'); -- Clock for fifo

END ctrfifo;

-----------------------------------------------------------------------

ARCHITECTURE descrip OF ctrfifo IS
```

```
  SIGNAL cnt:    INTEGER RANGE 0 TO capf-1           := capf-1 ; -- Contador de pulsos de reloj para línea
  SIGNAL scs:    STD_LOGIC                           := '0'    ; -- Flag de comienzo de reloj para
semicuadros

BEGIN

  clk_fifos:
  PROCESS (n_rst,clk) BEGIN
    IF n_rst = '0' THEN
      cnt <= capf-1;                    -- reseteo de contador de pulsos
      scs <= '0';                       -- reseteo de flag de inicio de reloj de semi-cuadro
    ELSIF clk'EVENT AND clk = '0' THEN
      IF cnt = 0 THEN                   -- inicio de reloj
        scs <= '1';
      ELSE
        cnt <= cnt - 1;                   -- decremento de contador de pulsos
      END IF;
    END IF;
  END PROCESS;

  clks <= NOT(clk) WHEN (scs = '1') ELSE '1';

END;
```

B6. DELAYCP1: Retardador para coeficientes de promediado

```
--------------------------------------------------------------------------
-- DELAYCP1                                                             --
-- Este modulo es la base para un retardador. Es básicamente un latch --
--                                                                      --
-- Autor: Aitzol Zuloaga I.                                             --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea  --
--        Departamento de electronica y telecomunicaciones            --
--                                                                      --
-- Fecha: 29/11/97                                                      --
--------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY delaycp1 IS
  GENERIC(
    CONSTANT nbits : INTEGER := 9 );
  PORT(
    SIGNAL d    :IN  STD_LOGIC_VECTOR (nbits-1 DOWNTO 0) ;
    SIGNAL clk  :IN  STD_LOGIC;
    SIGNAL q    :OUT STD_LOGIC_VECTOR (nbits-1 DOWNTO 0));
END delaycp1;

ARCHITECTURE behav OF delaycp1 IS
BEGIN

  PROCESS (clk) BEGIN
    IF clk = '1' THEN
      q <= d;
    END IF;
  END PROCESS;

END;
```

B7. DELAYCP2: Retardador para coeficientes de promediado (Interno)

```
--------------------------------------------------------------------------
-- DELAYCP2                                                             --
-- RETARDADOR SECUNDARIO (INTERNO) PARA EL CALCULO DE LAPLACIANOS     --
--                                                                      --
-- Algoritmo:                                                           --
--   *Este módulo toma la aquellas velocidades ya retardadas previa-  --
--    mente en los retardadores primarios (externos), de éstos        --
--    obtiene la velocidad retardada un semi-cuadro, un semi-cuadro y --
--    una linea, y un semi-cuadro y dos líneas                        --
--   *Este retardador tiene que dejar pasar una de las señales sin    --
--    retardar, retardada un pixel y retardada dos pixels. con lo     --
--    que se obtienen 9 señales.                                      --
--                                                                      --
-- Autor: Aitzol Zuloaga I.                                             --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea  --
--        Departamento de electrónica y telecomunicaciones            --
--                                                                      --
-- Fecha: 01/11/98                                                      --
--------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;
--------------------------------------------------------------------------

ENTITY delaycp2 IS

  PORT(
    SIGNAL ve0:  IN   STD_LOGIC_VECTOR ( 7 DOWNTO 0) ; -- campo vectorial de velocidad
    SIGNAL ve1:  IN   STD_LOGIC_VECTOR ( 7 DOWNTO 0) ; -- campo vectorial de velocidad
    SIGNAL ve2:  IN   STD_LOGIC_VECTOR ( 7 DOWNTO 0) ; -- campo vectorial de velocidad
```

```
    SIGNAL clk:  IN   STD_LOGIC                       ; -- reloj de entrada de datos
    SIGNAL v00:  OUT  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL v01:  OUT  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL v02:  OUT  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL v10:  OUT  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL v11:  OUT  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL v12:  OUT  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL v20:  OUT  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL v21:  OUT  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL v22:  OUT  STD_LOGIC_VECTOR ( 7 DOWNTO 0));

END delaycp2;

-----------------------------------------------------------------------

ARCHITECTURE struc OF delaycp2 IS

  CONSTANT  nb: INTEGER := 8; --Número de bits de shift-registers grandes

  COMPONENT delaycp1
    GENERIC(
      CONSTANT nbits : INTEGER);
    PORT(
      SIGNAL d    :IN  STD_LOGIC_VECTOR ;
      SIGNAL clk  :IN  STD_LOGIC;
      SIGNAL q    :OUT STD_LOGIC_VECTOR );
  END COMPONENT;

  SIGNAL v00x: STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
  SIGNAL v01x: STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
  SIGNAL v02x: STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
  SIGNAL v10x: STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
  SIGNAL v11x: STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
  SIGNAL v12x: STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
  SIGNAL v20x: STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
  SIGNAL v21x: STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
  SIGNAL v22x: STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;

BEGIN

  -- retardos sobre v0
  u00: v00x <= ve0;
  u10: delaycp1 GENERIC MAP (nb)    PORT MAP (v00x, clk, v10x);
  u20: delaycp1 GENERIC MAP (nb)    PORT MAP (v10x, clk, v20x);

  -- retardos sobre v1
  u01: v01x <= ve1;
  u11: delaycp1 GENERIC MAP (nb)    PORT MAP (v01x, clk, v11x);
  u21: delaycp1 GENERIC MAP (nb)    PORT MAP (v11x, clk, v21x);

  -- retardos sobre v2
  u02: v02x <= ve2;
  u12: delaycp1 GENERIC MAP (nb)    PORT MAP (v02x, clk, v12x);
  u22: delaycp1 GENERIC MAP (nb)    PORT MAP (v12x, clk, v22x);

  -- asignación para salir del módulo
  v00 <= v00x;
  v10 <= v10x;
  v20 <= v20x;
  v01 <= v01x;
  v11 <= v11x;
  v21 <= v21x;
  v02 <= v02x;
  v12 <= v12x;
  v22 <= v22x;

END;
```

B8. DELAYCP3: Retardador para coeficientes de promediado (Externo)

```
-------------------------------------------------------------------------
-- DELAYCP3                                                            --
-- RETARDADOR PRIMARIO (EXTERNO) PARA EL CALCULO DE LAPLACIANOS        --
--                                                                     --
-- Algoritmo:                                                          --
--   *Este módulo toma la secuencia de velocidades y ejecuta los       --
--    retardos primarios, aquellos que involucran gran cantidad de     --
--    memoria.                                                         --
--   *Este retardador tiene que retardar todas las señales un          --
--    semi-cuadro, para lograr compensar el retardo de las señales     --
--    en el lazo principal. El semi-cuadro tiene un tamaño igual       --
--    al tamaño del cuadro de la imagen menos el retardo del lazo      --
--    principal. Esto es calculado por el generador de sincronismos    --
--    de las FIFOs y en este módulo sólo tiene interes para espe-      --
--    cificar el tamaño de la FIFO que será el del FIFO integrado      --
--    comercialmente disponible.                                       --
--                                                                     --
-- Autor: Aitzol Zuloaga I.                                            --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea   --
--        Departamento de electrónica y telecomunicaciones             --
--                                                                     --
-- Fecha: 02/11/98                                                     --
-------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-----------------------------------------------------------------------
```

```
ENTITY delaycp3 IS
  GENERIC(
    CONSTANT diml: IN  INTEGER                 := 64    ; -- Dimension de línea en pixels 256
    CONSTANT dimf: IN  INTEGER               := 4096   ); -- Dimensión de cuadro en pixels 65536
  PORT(
    SIGNAL v:      IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)    ; -- secuencia de velocidades
    SIGNAL n_prst: IN  STD_LOGIC                         ; -- Reset
    SIGNAL pclk:   IN  STD_LOGIC                         ; -- Reloj de procesamiento para escritura en fifo
    SIGNAL pclkl:  IN  STD_LOGIC                         ; -- Reloj de procesamiento para lectura de fifo de
líneas
    SIGNAL pclks:  IN  STD_LOGIC                         ; -- Reloj de procesamiento para lectura de fifo de
semi-cuadros
    SIGNAL ve0:    OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)    ; -- campo vectorial de velocidad
    SIGNAL ve1:    OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)    ; -- campo vectorial de velocidad
    SIGNAL ve2:    OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)   ); -- campo vectorial de velocidad

END delaycp3;

--------------------------------------------------------------------

ARCHITECTURE struc OF delaycp3 IS

  COMPONENT memfifo   -- Memoria fifo configurable para simulación
    GENERIC(
      CONSTANT capacity: IN  INTEGER                        ; -- Capacidad de la fifo
      CONSTANT nbytes:   IN  INTEGER                       ); -- número de bytes de la fifo
    PORT(
      SIGNAL   datain:   IN  STD_ULOGIC_VECTOR ( nbytes DOWNTO 0) ; -- Entrada de datos (8 bits)
      SIGNAL   n_wrclk:  IN  STD_LOGIC                      ; -- Reloj de escritura
      SIGNAL   n_rdclk:  IN  STD_LOGIC                      ; -- Reloj de lectura
      SIGNAL   n_reset:  IN  STD_LOGIC                      ; -- Reset
      SIGNAL   dataout:  OUT STD_ULOGIC_VECTOR ( nbytes DOWNTO 0)); -- Salida de datos
  END COMPONENT;

  SIGNAL v0x:    STD_ULOGIC_VECTOR ( 7 DOWNTO 0)            ; -- Datos sin retardo. Auxiliar
  SIGNAL ve0x:   STD_ULOGIC_VECTOR ( 7 DOWNTO 0)            ; -- Datos con retardo de un semi-cuadro.
Auxiliar
  SIGNAL ve1x:   STD_ULOGIC_VECTOR ( 7 DOWNTO 0)            ; -- Datos con retardo de un semi-cuadro y una
línea. Auxiliar
  SIGNAL ve2x:   STD_ULOGIC_VECTOR ( 7 DOWNTO 0)            ; -- Datos con retardo de un semi-cuadro y dos
líneas. Auxiliar
BEGIN

  v0x <= To_StdULogicVector(v);

  -- Instanciación de fifos
  umemfifo0:
  memfifo GENERIC MAP (dimf,8)
          PORT MAP    (v0x,pclk,pclks,n_prst,ve0x); -- fifo de cuadro

  umemfifo1:
  memfifo GENERIC MAP (diml,8)
          PORT MAP    (ve0x,pclk,pclkl,n_prst,ve1x); -- fifo de línea

  umemfifo2:
  memfifo GENERIC MAP (diml,8)
          PORT MAP    (ve1x,pclk,pclkl,n_prst,ve2x); -- fifo de línea

  -- Salidas
  ve0 <= To_StdLogicVector(ve0x)           ; -- Salida con retardo de un semi-cuadro
  ve1 <= To_StdLogicVector(ve1x)           ; -- Salida con retardo de un semi-cuadro y una línea
  ve2 <= To_StdLogicVector(ve2x)           ; -- Salida con retardo de un semi-cuadro y dos líneas

END;
```

B9. DELAYGR1: Retardador primario para cálculo de gradientes

```
--------------------------------------------------------------------------
-- DELAYGR1                                                             --
-- RETARDADOR PRIMARIO PARA EL CALCULO DE GRADIENTES                    --
--                                                                      --
-- Algoritmo:                                                           --
--   *Este módulo toma la secuencia de imágenes, prepara los retardos --
--    primarios, aquellos que involucran gran cantidad de memoria     --
--                                                                      --
-- Autor: Aitzol Zuloaga I.                                             --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea  --
--        Departamento de electrónica y telecomunicaciones            --
--                                                                      --
-- Fecha: 29/03/98                                                      --
--------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

--------------------------------------------------------------------

ENTITY delaygr1 IS
  GENERIC(
    CONSTANT diml: IN  INTEGER                 := 64 ; -- Dimension de línea en pixels 256
    CONSTANT dimf: IN  INTEGER               := 4096); -- Dimensión de cuadro en pixels 65536
  PORT(
    SIGNAL pdat:   IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ; -- secuencia de imágenes
    SIGNAL n_prst: IN  STD_LOGIC                      ; -- Reset
    SIGNAL pclk:   IN  STD_LOGIC                      ; -- Processing clock for fifo write
    SIGNAL pclkl:  IN  STD_LOGIC                      ; -- Processing clock for line fifo read
    SIGNAL pclkf:  IN  STD_LOGIC                      ; -- Processing clock for frame fifo read
```

```
    SIGNAL e010:   OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0) ; -- imagen con retardo de una linea
    SIGNAL e001:   OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0) ; -- imagen con retardo de un cuadro
    SIGNAL e011:   OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)); -- imagen con retardo de un cuadro y una línea
END delaygr1;

--------------------------------------------------------------------

ARCHITECTURE struc OF delaygr1 IS

  COMPONENT memfifo   -- Memoria fifo configurable para simulación
    GENERIC(
      CONSTANT capacity: IN  INTEGER                       ; -- Capacidad de la fifo
      CONSTANT nbytes:   IN  INTEGER                      ); -- número de bytes de la fifo
    PORT(
      SIGNAL   datain:   IN  STD_ULOGIC_VECTOR ( 7 DOWNTO 0) ; -- Entrada de datos (8 bits)
      SIGNAL   n_wrclk:  IN  STD_LOGIC                       ; -- Reloj de escritura
      SIGNAL   n_rdclk:  IN  STD_LOGIC                       ; -- Reloj de lectura
      SIGNAL   n_reset:  IN  STD_LOGIC                       ; -- Reset
      SIGNAL   dataout:  OUT STD_ULOGIC_VECTOR ( 7 DOWNTO 0)); -- Salida de datos
  END COMPONENT;

  SIGNAL x000:   STD_ULOGIC_VECTOR ( 7 DOWNTO 0)             ; -- Datos sin retardo
  SIGNAL x001:   STD_ULOGIC_VECTOR ( 7 DOWNTO 0)             ; -- Datos con retardo de un cuadro
  SIGNAL x010:   STD_ULOGIC_VECTOR ( 7 DOWNTO 0)             ; -- Datos con retardo de un línea
  SIGNAL x011:   STD_ULOGIC_VECTOR ( 7 DOWNTO 0)             ; -- Datos con retardo de un cuadro y una línea

BEGIN

  x000 <= To_StdULogicVector(pdat);

  -- Instanciación de fifos
  u001: memfifo GENERIC MAP (dimf,8)
                PORT MAP    (x000,pclk,pclkf,n_prst,x001); -- fifo de cuadro

  u010: memfifo GENERIC MAP (diml,8)
                PORT MAP    (x000,pclk,pclkl,n_prst,x010); -- fifo de línea

  u011: memfifo GENERIC MAP (diml,8)
                PORT MAP    (x001,pclk,pclkl,n_prst,x011); -- fifo de línea

  -- Salidas
  e001 <= To_StdLogicVector(x001); -- Salida con retardo de un cuadro
  e010 <= To_StdLogicVector(x010); -- Salida con retardo de una línea
  e011 <= To_StdLogicVector(x011); -- Salida con retardo de un cuadro y una línea

END;
```

B10. DELAYGR2: Retardador secundario para cálculo de gradientes

```
------------------------------------------------------------------------
-- DELAYGR2                                                           --
-- RETARDADOR SECUNDARIO PARA EL CALCULO DE GRADIENTES                --
--                                                                    --
-- Algoritmo:                                                         --
--   *Este módulo toma los pixels que envia el retardador primario    --
--    (que es el encargado de los retardos de cuadro 64KB y líneas    --
--    256B) para proveer a la etapa de gradientes de todos los        --
--    pixels que requiere                                             --
--                                                                    --
-- Autor: Aitzol Zuloaga I.                                           --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea  --
--        Departamento de electrónica y telecomunicaciones            --
--                                                                    --
-- Fecha: 18/12/98                                                    --
------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

--------------------------------------------------------------------

ENTITY delaygr2 IS
  PORT(
    SIGNAL x000:   IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ; -- imagen sin retardo
    SIGNAL x010:   IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ; -- imagen con retardo de una linea
    SIGNAL x001:   IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ; -- imagen con retardo de un cuadro
    SIGNAL x011:   IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ; -- imagen con retardo de un cuadro y una línea
    SIGNAL clk:    IN  STD_LOGIC                       ; -- reloj de entrada de datos
    SIGNAL e000:   OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0) ; -- imagen sin retardo
    SIGNAL e010:   OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0) ; -- imagen con retardo de una linea
    SIGNAL e001:   OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0) ; -- imagen con retardo de un cuadro
    SIGNAL e011:   OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0) ; -- imagen con retardo de un cuadro y una línea
    SIGNAL e100:   OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0) ; -- imagen con retardo de un pixel
    SIGNAL e101:   OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0) ; -- imagen con retardo de una linea y un pixel
    SIGNAL e110:   OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0) ; -- imagen con retardo de un cuadro y un pixel
    SIGNAL e111:   OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)); -- imagen con retardo de un cuadro, una línea y un
pixel

END delaygr2;

--------------------------------------------------------------------

ARCHITECTURE struc OF delaygr2 IS
    SIGNAL s000:     STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL s010:     STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL s001:     STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL s011:     STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL t000:     STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
```

```
    SIGNAL t001:     STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL t010:     STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL t011:     STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL t100:     STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL t101:     STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL t110:     STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL t111:     STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;

BEGIN

  -- Se sincronizan los datos externos con el frente de bajada para evitar los retardos
  -- no controlables de los dispositivos externos
  PROCESS (clk) BEGIN
    IF clk = '0' THEN
      s000 <= x000;
      s001 <= x001;
      s010 <= x010;
      s011 <= x011;
    END IF;
  END PROCESS;

  -- Se sincronizan los datos con el frente de subida
  PROCESS (clk) BEGIN
    IF clk = '1' THEN
      t000 <= s000;
      t001 <= s001;
      t010 <= s010;
      t011 <= s011;
    END IF;
  END PROCESS;

  -- Se generan los retardos de un pixel
  PROCESS (clk) BEGIN
    IF clk = '1' THEN
      t100 <= t000;
      t101 <= t001;
      t110 <= t010;
      t111 <= t011;
    END IF;
  END PROCESS;

  -- Se asignan a los puertos de la entidad
  e000 <= t000;
  e001 <= t001;
  e010 <= t010;
  e011 <= t011;
  e100 <= t100;
  e101 <= t101;
  e110 <= t110;
  e111 <= t111;


END;
```

B11. DELAYSI: Retardador para sincronismo

```
--------------------------------------------------------------------------
-- DELAYSI                                                              --
-- Este modulo es un retardador para señales de sincronismo (1bit)     --
--                                                                      --
-- Autor: Aitzol Zuloaga I.                                             --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea    --
--        Departamento de electronica y telecomunicaciones              --
--                                                                      --
-- Fecha: 09/07/98                                                      --
--------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

--------------------------------------------------------------------------

ENTITY delaysi IS
  GENERIC(
    CONSTANT ndelays: INTEGER := 25);
  PORT(
    SIGNAL d    :IN  STD_LOGIC ;
    SIGNAL clk  :IN  STD_LOGIC ;
    SIGNAL q    :OUT STD_LOGIC);
END delaysi;

--------------------------------------------------------------------------

ARCHITECTURE behav OF delaysi IS

  SIGNAL dat: STD_LOGIC_VECTOR(ndelays DOWNTO 0);

BEGIN

  dat(ndelays) <= d;
  g1:
  FOR i IN (ndelays-1) DOWNTO 0 GENERATE

    PROCESS (clk) BEGIN
      IF clk = '1' THEN
        dat(i) <= dat(i+1);
      END IF;
```

```
    END PROCESS;

  END GENERATE;
  q <= dat(0);

END;

--------------------------------------------------------------------------
```

B12. DELAYVP: Retardador para compensación

```
--------------------------------------------------------------------------
-- DELAYVP                                                              --
-- Este modulo es un retardador para compensar el retardo del divisor --
--                                                                      --
-- Autor: Aitzol Zuloaga I.                                             --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea  --
--        Departamento de electronica y telecomunicaciones            --
--                                                                      --
-- Fecha: 30/06/98                                                      --
--------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

--------------------------------------------------------------------------

ENTITY delayvp IS
  GENERIC(
    CONSTANT ndelays: INTEGER := 22);
  PORT(
    SIGNAL d    :IN  STD_LOGIC_VECTOR (8 DOWNTO 0);
    SIGNAL clk  :IN  STD_LOGIC;
    SIGNAL q    :OUT STD_LOGIC_VECTOR (8 DOWNTO 0));
END delayvp;

--------------------------------------------------------------------------

ARCHITECTURE behav OF delayvp IS

  CONSTANT uld: INTEGER := d'HIGH;
  TYPE sig_dat IS ARRAY (ndelays DOWNTO 0) OF STD_LOGIC_VECTOR (8 DOWNTO 0);
  SIGNAL dat: sig_dat;

BEGIN

  dat(ndelays) <= d;

  g1:
  FOR i IN (ndelays-1) DOWNTO 0 GENERATE
    PROCESS (clk) BEGIN
      IF clk = '1' THEN
        dat(i) <= dat(i+1);
      END IF;
    END PROCESS;

  END GENERATE;
  q <= dat(0);

END;
```

B13. DIVIP03: Divisor por tres para coeficientes de promediado

```
--------------------------------------------------------------------------
-- DIVIP03                                                              --
-- Divisor de un número de 11 bits (10 + signo) entre 3                 --
--                                                                      --
-- Algoritmo:                                                           --
--   +El proceso se efectua por etapas, controladas por una señal     --
--    de reloj (clk). El tiempo total de cálculo es igual a 10        --
--    períodos de reloj                                                 --
--                                                                      --
-- Autor: Aitzol Zuloaga I.                                             --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea  --
--        Departamento de electrónica y telecomunicaciones            --
--                                                                      --
-- Fecha: 29/01/99                                                      --
--------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;

--------------------------------------------------------------------------

ENTITY divip03 IS
  PORT(
    n:   IN  STD_LOGIC_VECTOR (10 DOWNTO 0)               ; -- numerador
    clk: IN  STD_LOGIC                                    ; -- reloj de operacion
    r:   OUT STD_LOGIC_VECTOR (9 DOWNTO 0):=(OTHERS=>'0')); -- resultado  r=n/d
END divip03;
```

```
--------------------------------------------------------------------------
ARCHITECTURE struct OF divip03 IS

  CONSTANT uln: INTEGER := n'HIGH;                                -- Limite superior del numerador
  CONSTANT uld: INTEGER := 1;                                     -- Limite superior del denominador
  CONSTANT ulr: INTEGER := r'HIGH;                                -- Limite superior del resultado
  CONSTANT dnd: INTEGER := uln-uld;                               -- Diferencia entre numerador y denominador
  CONSTANT gnd: STD_LOGIC_VECTOR (uln DOWNTO 0) := (OTHERS=>'0'); -- Arreglo de bits a cero
  SIGNAL   sgn: STD_LOGIC_VECTOR (uln-1 DOWNTO 0) := (OTHERS=>'0'); -- Desplazamiento del signo

--Variables donde se van guardando los restos (na)
--TYPE sig_na IS ARRAY (uln DOWNTO 0) OF STD_LOGIC_VECTOR (uln DOWNTO 0);
--SIGNAL na: sig_na;
  SIGNAL na08:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na07:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na06:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na05:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na04:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na03:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na02:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na01:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na00:  STD_LOGIC_VECTOR (uln DOWNTO 0);

--Variables donde se van guardando las restas que es necesario realizar (nb)
--TYPE sig_nb IS ARRAY (uln DOWNTO 0) OF STD_LOGIC_VECTOR (uln+1 DOWNTO 0);
--SIGNAL nb: sig_nb;
  SIGNAL nb08:  STD_LOGIC_VECTOR (uld+2 DOWNTO 0);
  SIGNAL nb07:  STD_LOGIC_VECTOR (uld+2 DOWNTO 0);
  SIGNAL nb06:  STD_LOGIC_VECTOR (uld+2 DOWNTO 0);
  SIGNAL nb05:  STD_LOGIC_VECTOR (uld+2 DOWNTO 0);
  SIGNAL nb04:  STD_LOGIC_VECTOR (uld+2 DOWNTO 0);
  SIGNAL nb03:  STD_LOGIC_VECTOR (uld+2 DOWNTO 0);
  SIGNAL nb02:  STD_LOGIC_VECTOR (uld+2 DOWNTO 0);
  SIGNAL nb01:  STD_LOGIC_VECTOR (uld+2 DOWNTO 0);
  SIGNAL nb00:  STD_LOGIC_VECTOR (uld+2 DOWNTO 0);

--Variables donde se van guardando los restos (nc)
--TYPE sig_nc IS ARRAY (uln DOWNTO 1) OF STD_LOGIC_VECTOR (uln DOWNTO 0);
--SIGNAL nc: sig_nc;
  SIGNAL nc09: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc08: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc07: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc06: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc05: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc04: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc03: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc02: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc01: STD_LOGIC_VECTOR (uln DOWNTO 0);

--Variables donde se van guardando los bits del resultado
  SIGNAL ra07:  STD_LOGIC;
  SIGNAL ra06:  STD_LOGIC_VECTOR (uln-09 DOWNTO 0);
  SIGNAL ra05:  STD_LOGIC_VECTOR (uln-08 DOWNTO 0);
  SIGNAL ra04:  STD_LOGIC_VECTOR (uln-07 DOWNTO 0);
  SIGNAL ra03:  STD_LOGIC_VECTOR (uln-06 DOWNTO 0);
  SIGNAL ra02:  STD_LOGIC_VECTOR (uln-05 DOWNTO 0);
  SIGNAL ra01:  STD_LOGIC_VECTOR (uln-04 DOWNTO 0);
  SIGNAL ra00:  STD_LOGIC_VECTOR (uln-03 DOWNTO 0);

  SIGNAL rs:    STD_LOGIC_VECTOR (ulr    DOWNTO 0);

BEGIN
  ge:
  FOR i IN uln-1 DOWNTO 0 GENERATE

    g09:
    IF i = 09 GENERATE
      p09:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
            sgn(i) <= n(i+1);
          IF   n(i+1)='1' THEN
            nc09 <= -n;
          ELSE
            nc09 <= n;
          END IF;
        END IF;
      END PROCESS;
    END GENERATE;

    g08:  --Punto de cambio del algoritmo (i=uln-uld-1)---------------------------------------
    IF i = 08 GENERATE
      p08:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na08 <= nc09;
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb08 <= ("0" & na08(uld+i+1 DOWNTO i)) - ("00" & "11");
      nc08 <= nb08(uld+1 DOWNTO  0) & na08(i-1 DOWNTO 0) WHEN nb08(uld+2) = '0' ELSE
              na08;
    END GENERATE;

    g07:
    IF i = 07 GENERATE
      p07:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na07 <= nc08;
```

```
          ra07 <= NOT nb08(uld+2);
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb07 <= ("0" & na07(uld+i+1 DOWNTO i)) - ("00" & "11");
      nc07 <= gnd((dnd-i-2) DOWNTO 0) & nb07(uld+1 DOWNTO  0) & na07(i-1 DOWNTO 0) WHEN nb07(uld+2) = '0' ELSE
              na07;
    END GENERATE;

    g06:
    IF i = 06 GENERATE
      p06:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na06 <= nc07;
          ra06 <= ra07 & (NOT nb07(uld+2)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb06 <= ("0" & na06(uld+i+1 DOWNTO i)) - ("00" & "11");
      nc06 <= gnd((dnd-i-2) DOWNTO 0) & nb06(uld+1 DOWNTO  0) & na06(i-1 DOWNTO 0) WHEN nb06(uld+2) = '0' ELSE
              na06;
    END GENERATE;

    g05:
    IF i = 05 GENERATE
      p05:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na05 <= nc06;
          ra05 <= ra06 & (NOT nb06(uld+2)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb05 <= ("0" & na05(uld+i+1 DOWNTO i)) - ("00" & "11");
      nc05 <= gnd((dnd-i-2) DOWNTO 0) & nb05(uld+1 DOWNTO  0) & na05(i-1 DOWNTO 0) WHEN nb05(uld+2) = '0' ELSE
              na05;
    END GENERATE;

    g04:
    IF i = 04 GENERATE
      p04:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na04 <= nc05;
          ra04 <= ra05 & (NOT nb05(uld+2));
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb04 <= ("0" & na04(uld+i+1 DOWNTO i)) - ("00" & "11");
      nc04 <= gnd((dnd-i-2) DOWNTO 0) & nb04(uld+1 DOWNTO  0) & na04(i-1 DOWNTO 0) WHEN nb04(uld+2) = '0' ELSE
              na04;
    END GENERATE;

    g03:
    IF i = 03 GENERATE
      p03:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na03 <= nc04;
          ra03 <= ra04 & (NOT nb04(uld+2));
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb03 <= ("0" & na03(uld+i+1 DOWNTO i)) - ("00" & "11");
      nc03 <= gnd((dnd-i-2) DOWNTO 0) & nb03(uld+1 DOWNTO  0) & na03(i-1 DOWNTO 0) WHEN nb03(uld+2) = '0' ELSE
              na03;
    END GENERATE;

    g02:
    IF i = 02 GENERATE
      p02:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na02 <= nc03;
          ra02 <= ra03 & (NOT nb03(uld+2));
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb02 <= ("0" & na02(uld+i+1 DOWNTO i)) - ("00" & "11");
      nc02 <= gnd((dnd-i-2) DOWNTO 0) & nb02(uld+1 DOWNTO  0) & na02(i-1 DOWNTO 0) WHEN nb02(uld+2) = '0' ELSE
              na02;
    END GENERATE;

    g01:
    IF i = 01 GENERATE
      p01:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na01 <= nc02;
          ra01 <= ra02 & (NOT nb02(uld+2));
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb01 <= ("0" & na01(uld+i+1 DOWNTO i)) - ("00" & "11");
      nc01 <= gnd((dnd-i-2) DOWNTO 0) & nb01(uld+1 DOWNTO  0) & na01(i-1 DOWNTO 0) WHEN nb01(uld+2) = '0' ELSE
              na01;
```

```
    END GENERATE;

    g00:
    IF i = 00 GENERATE
      p00:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na00 <= nc01;
          ra00 <= ra01 & (NOT nb01(uld+2));
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb00 <= ("0" & na00(uld+i+1 DOWNTO i)) - ("00" & "11");
    END GENERATE;

  END GENERATE;

  -- Sincronización del resultado
  rs <= '0' & ra00 & (NOT nb00(uld+1)) WHEN sgn(0)='0' ELSE -('0' & ra00 & (NOT nb00(uld+1)));  -- Resultado
final
  l:
  PROCESS (clk)
  BEGIN
    IF clk= '1' THEN
      r <= rs;
    END IF;
  END PROCESS;

END;
```

B14. DIVISOR: Divisor para el cálculo de términos R

```
--------------------------------------------------------------------------
-- DIVISOR                                                              --
-- Divisor de 26 a 18 bits                                              --
--                                                                      --
-- Algoritmo:                                                           --
--   +El proceso se efectua por etapas, controladas por una señal       --
--    de reloj (clk). El tiempo total de cálculo es igual a tantos      --
--    períodos de reloj como bits tiene el numerador.                   --
--                                                                      --
-- Limitaciones:                                                        --
--   *El resultado de una división por cero será todos los bits en 1    --
--   *El resultado tendrá tantos bits como el numerador                 --
--   *Sólo se considera el signo del numerador. El denominador se       --
--    se considera siempre positivo.                                    --
--   *El resultado no será en complemento a 2, sólo tendrá el bit       --
--    más significativo en '1' para indicar que el resultado es         --
--    negativo, de esta manera se hace más sencilla la operación        --
--    de ajuste de número de bits que se requerira posteriormente       --
--    y se reducen los recursos a utilizar dentro de la PLD             --
--                                                                      --
-- Autor: Aitzol Zuloaga I.                                             --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea    --
--        Departamento de electrónica y telecomunicaciones              --
--                                                                      --
-- Fecha: 26/06/98                                                      --
--------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;

--------------------------------------------------------------------------

ENTITY divisor IS
  PORT(
    n:   IN  STD_LOGIC_VECTOR (25 DOWNTO 0)               ; -- numerador
    d:   IN  STD_LOGIC_VECTOR (16 DOWNTO 0)               ; -- denominador
    clk: IN  STD_LOGIC                                    ; -- reloj de operacion
    r:   OUT STD_LOGIC_VECTOR (25 DOWNTO 0):=(OTHERS=>'0')); -- resultado  r=n/d
END divisor;

--------------------------------------------------------------------------
ARCHITECTURE struct OF divisor IS

  CONSTANT uln: INTEGER := n'HIGH;                              -- Limite superior del numerador
  CONSTANT uld: INTEGER := d'HIGH;                              -- Limite superior del denominador
  CONSTANT dnd: INTEGER := uln-uld;                             -- Diferencia entre numerador y denominador
  CONSTANT gnd: STD_LOGIC_VECTOR (uln DOWNTO 0) := (OTHERS=>'0'); -- Arreglo de bits a cero
  SIGNAL   sgn: STD_LOGIC_VECTOR (uln DOWNTO 0) := (OTHERS=>'0'); -- Desplazamiento del signo

--Variables donde se van guardando los restos (na)
--TYPE sig_na IS ARRAY (uln DOWNTO 0) OF STD_LOGIC_VECTOR (uln DOWNTO 0);
--SIGNAL na: sig_na;
  SIGNAL na25:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na24:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na23:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na22:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na21:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na20:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na19:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na18:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na17:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na16:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na15:  STD_LOGIC_VECTOR (uln DOWNTO 0);
```

```
  SIGNAL na14:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na13:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na12:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na11:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na10:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na09:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na08:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na07:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na06:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na05:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na04:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na03:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na02:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na01:  STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL na00:  STD_LOGIC_VECTOR (uln DOWNTO 0);

--Variables donde se van guardando las restas que es necesario realizar (nb)
--TYPE sig_nb IS ARRAY (uln DOWNTO 0) OF STD_LOGIC_VECTOR (uln+1 DOWNTO 0);
--SIGNAL nb: sig_nb;
  SIGNAL nb24:  STD_LOGIC_VECTOR (uld+1 DOWNTO 0);
  SIGNAL nb23:  STD_LOGIC_VECTOR (uld+1 DOWNTO 0);
  SIGNAL nb22:  STD_LOGIC_VECTOR (uld+1 DOWNTO 0);
  SIGNAL nb21:  STD_LOGIC_VECTOR (uld+1 DOWNTO 0);
  SIGNAL nb20:  STD_LOGIC_VECTOR (uld+1 DOWNTO 0);
  SIGNAL nb19:  STD_LOGIC_VECTOR (uld+1 DOWNTO 0);
  SIGNAL nb18:  STD_LOGIC_VECTOR (uld+1 DOWNTO 0);
  SIGNAL nb17:  STD_LOGIC_VECTOR (uld+1 DOWNTO 0);
  SIGNAL nb16:  STD_LOGIC_VECTOR (uld+1 DOWNTO 0);
  SIGNAL nb15:  STD_LOGIC_VECTOR (uld+1 DOWNTO 0);
  SIGNAL nb14:  STD_LOGIC_VECTOR (uld+1 DOWNTO 0);
  SIGNAL nb13:  STD_LOGIC_VECTOR (uld+1 DOWNTO 0);
  SIGNAL nb12:  STD_LOGIC_VECTOR (uld+1 DOWNTO 0);
  SIGNAL nb11:  STD_LOGIC_VECTOR (uld+1 DOWNTO 0);
  SIGNAL nb10:  STD_LOGIC_VECTOR (uld+1 DOWNTO 0);
  SIGNAL nb09:  STD_LOGIC_VECTOR (uld+1 DOWNTO 0);
  SIGNAL nb08:  STD_LOGIC_VECTOR (uld+2 DOWNTO 0);
  SIGNAL nb07:  STD_LOGIC_VECTOR (uld+2 DOWNTO 0);
  SIGNAL nb06:  STD_LOGIC_VECTOR (uld+2 DOWNTO 0);
  SIGNAL nb05:  STD_LOGIC_VECTOR (uld+2 DOWNTO 0);
  SIGNAL nb04:  STD_LOGIC_VECTOR (uld+2 DOWNTO 0);
  SIGNAL nb03:  STD_LOGIC_VECTOR (uld+2 DOWNTO 0);
  SIGNAL nb02:  STD_LOGIC_VECTOR (uld+2 DOWNTO 0);
  SIGNAL nb01:  STD_LOGIC_VECTOR (uld+2 DOWNTO 0);
  SIGNAL nb00:  STD_LOGIC_VECTOR (uld+2 DOWNTO 0);

--Variables donde se van guardando los restos (nc)
--TYPE sig_nc IS ARRAY (uln DOWNTO 1) OF STD_LOGIC_VECTOR (uln DOWNTO 0);
--SIGNAL nc: sig_nc;
  SIGNAL nc24: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc23: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc22: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc21: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc20: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc19: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc18: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc17: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc16: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc15: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc14: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc13: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc12: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc11: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc10: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc09: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc08: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc07: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc06: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc05: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc04: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc03: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc02: STD_LOGIC_VECTOR (uln DOWNTO 0);
  SIGNAL nc01: STD_LOGIC_VECTOR (uln DOWNTO 0);

--Variables donde se van guardando los denominadores (da)
--TYPE sig_da IS ARRAY (uln DOWNTO 0) OF STD_LOGIC_VECTOR (uld DOWNTO 0);
--SIGNAL da: sig_da;
  SIGNAL da25: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da24: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da23: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da22: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da21: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da20: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da19: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da18: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da17: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da16: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da15: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da14: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da13: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da12: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da11: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da10: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da09: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da08: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da07: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da06: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da05: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da04: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da03: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da02: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da01: STD_LOGIC_VECTOR (uld DOWNTO 0);
  SIGNAL da00: STD_LOGIC_VECTOR (uld DOWNTO 0);
```

```
 --Variables donde se van guardando los bits del resultado
   SIGNAL ra23:  STD_LOGIC_VECTOR (uln-25 DOWNTO 0);
   SIGNAL ra22:  STD_LOGIC_VECTOR (uln-24 DOWNTO 0);
   SIGNAL ra21:  STD_LOGIC_VECTOR (uln-23 DOWNTO 0);
   SIGNAL ra20:  STD_LOGIC_VECTOR (uln-22 DOWNTO 0);
   SIGNAL ra19:  STD_LOGIC_VECTOR (uln-21 DOWNTO 0);
   SIGNAL ra18:  STD_LOGIC_VECTOR (uln-20 DOWNTO 0);
   SIGNAL ra17:  STD_LOGIC_VECTOR (uln-19 DOWNTO 0);
   SIGNAL ra16:  STD_LOGIC_VECTOR (uln-18 DOWNTO 0);
   SIGNAL ra15:  STD_LOGIC_VECTOR (uln-17 DOWNTO 0);
   SIGNAL ra14:  STD_LOGIC_VECTOR (uln-16 DOWNTO 0);
   SIGNAL ra13:  STD_LOGIC_VECTOR (uln-15 DOWNTO 0);
   SIGNAL ra12:  STD_LOGIC_VECTOR (uln-14 DOWNTO 0);
   SIGNAL ra11:  STD_LOGIC_VECTOR (uln-13 DOWNTO 0);
   SIGNAL ra10:  STD_LOGIC_VECTOR (uln-12 DOWNTO 0);
   SIGNAL ra09:  STD_LOGIC_VECTOR (uln-11 DOWNTO 0);
   SIGNAL ra08:  STD_LOGIC_VECTOR (uln-10 DOWNTO 0);
   SIGNAL ra07:  STD_LOGIC_VECTOR (uln-09 DOWNTO 0);
   SIGNAL ra06:  STD_LOGIC_VECTOR (uln-08 DOWNTO 0);
   SIGNAL ra05:  STD_LOGIC_VECTOR (uln-07 DOWNTO 0);
   SIGNAL ra04:  STD_LOGIC_VECTOR (uln-06 DOWNTO 0);
   SIGNAL ra03:  STD_LOGIC_VECTOR (uln-05 DOWNTO 0);
   SIGNAL ra02:  STD_LOGIC_VECTOR (uln-04 DOWNTO 0);
   SIGNAL ra01:  STD_LOGIC_VECTOR (uln-03 DOWNTO 0);
   SIGNAL ra00:  STD_LOGIC_VECTOR (uln-02 DOWNTO 0);

 BEGIN
   g:
   FOR i IN uln DOWNTO 0 GENERATE

     g25:
     IF i = 25 GENERATE
       p25:
       PROCESS (clk)
       BEGIN
         IF clk= '1' THEN
           -- Se guarda el signo del numerador y se hace una división entre positivos
           sgn(i) <= n(i);
           IF   n(i)='1' THEN
             na25 <= -n;
           ELSE
             na25 <= n;
           END IF;
           da25 <= d;
         END IF;
       END PROCESS;
     END GENERATE;

     g24:
     IF i = 24 GENERATE
       p24:
       PROCESS (clk)
       BEGIN
         IF clk= '1' THEN
           na24 <= na25;
           da24 <= da25;
           sgn(i) <= sgn(i+1);
         END IF;
       END PROCESS;
       nb24 <= (gnd(i-dnd DOWNTO 0) & na24(uln DOWNTO i)) - ("0" & da24);
       nc24 <= nb24((uln-i) DOWNTO  0) & na24(i-1 DOWNTO 0) WHEN nb24(uld+1) = '0' ELSE
               na24;
     END GENERATE;

     g23:
     IF i = 23 GENERATE
       p23:
       PROCESS (clk)
       BEGIN
         IF clk= '1' THEN
           na23 <= nc24;
           da23 <= da24;
           ra23(0 DOWNTO 0) <= NOT (nb24(uld+1 DOWNTO uld+1)); --bits de resultado
           sgn(i) <= sgn(i+1);
         END IF;
       END PROCESS;
       nb23 <= (gnd(i-dnd DOWNTO 0) & na23(uln DOWNTO i)) - ("0" & da23);
       nc23 <= nb23((uln-i) DOWNTO  0) & na23(i-1 DOWNTO 0) WHEN nb23(uld+1) = '0' ELSE
               na23;
     END GENERATE;

     g22:
     IF i = 22 GENERATE
       p22:
       PROCESS (clk)
       BEGIN
         IF clk= '1' THEN
           na22 <= nc23;
           da22 <= da23;
           ra22 <= ra23 & NOT (nb23(uld+1)); --bits de resultado
           sgn(i) <= sgn(i+1);
         END IF;
       END PROCESS;
       nb22 <= (gnd(i-dnd DOWNTO 0) & na22(uln DOWNTO i)) - ("0" & da22);
       nc22 <= nb22((uln-i) DOWNTO  0) & na22(i-1 DOWNTO 0) WHEN nb22(uld+1) = '0' ELSE
               na22;
     END GENERATE;

     g21:
     IF i = 21 GENERATE
       p21:
       PROCESS (clk)
       BEGIN
```

```
        IF clk= '1' THEN
          na21 <= nc22;
          da21 <= da22;
          ra21 <= ra22 & NOT (nb22(uld+1)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb21 <= (gnd(i-dnd DOWNTO 0) & na21(uln DOWNTO i)) - ("0" & da21);
      nc21 <= nb21((uln-i) DOWNTO  0) & na21(i-1 DOWNTO 0) WHEN nb21(uld+1) = '0' ELSE
              na21;
    END GENERATE;

    g20:
    IF i = 20 GENERATE
      p20:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na20 <= nc21;
          da20 <= da21;
          ra20 <= ra21 & NOT (nb21(uld+1)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb20 <= (gnd(i-dnd DOWNTO 0) & na20(uln DOWNTO i)) - ("0" & da20);
      nc20 <= nb20((uln-i) DOWNTO  0) & na20(i-1 DOWNTO 0) WHEN nb20(uld+1) = '0' ELSE
              na20;
    END GENERATE;

    g19:
    IF i = 19 GENERATE
      p19:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na19 <= nc20;
          da19 <= da20;
          ra19 <= ra20 & NOT (nb20(uld+1)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb19 <= (gnd(i-dnd DOWNTO 0) & na19(uln DOWNTO i)) - ("0" & da19);
      nc19 <= nb19((uln-i) DOWNTO  0) & na19(i-1 DOWNTO 0) WHEN nb19(uld+1) = '0' ELSE
              na19;
    END GENERATE;

    g18:
    IF i = 18 GENERATE
      p18:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na18 <= nc19;
          da18 <= da19;
          ra18 <= ra19 & NOT (nb19(uld+1)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb18 <= (gnd(i-dnd DOWNTO 0) & na18(uln DOWNTO i)) - ("0" & da18);
      nc18 <= nb18((uln-i) DOWNTO  0) & na18(i-1 DOWNTO 0) WHEN nb18(uld+1) = '0' ELSE
              na18;
    END GENERATE;

    g17:
    IF i = 17 GENERATE
      p17:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na17 <= nc18;
          da17 <= da18;
          ra17 <= ra18 & NOT (nb18(uld+1)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb17 <= (gnd(i-dnd DOWNTO 0) & na17(uln DOWNTO i)) - ("0" & da17);
      nc17 <= nb17((uln-i) DOWNTO  0) & na17(i-1 DOWNTO 0) WHEN nb17(uld+1) = '0' ELSE
              na17;
    END GENERATE;

    g16:
    IF i = 16 GENERATE
      p16:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na16 <= nc17;
          da16 <= da17;
          ra16 <= ra17 & NOT (nb17(uld+1)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb16 <= (gnd(i-dnd DOWNTO 0) & na16(uln DOWNTO i)) - ("0" & da16);
      nc16 <= nb16((uln-i) DOWNTO  0) & na16(i-1 DOWNTO 0) WHEN nb16(uld+1) = '0' ELSE
              na16;
    END GENERATE;

    g15:
    IF i = 15 GENERATE
      p15:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
```

```
          na15 <= nc16;
          da15 <= da16;
          ra15 <= ra16 & (NOT nb16(uld+1)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb15 <= (gnd(i-dnd DOWNTO 0) & na15(uln DOWNTO i)) - ("0" & da15);
      nc15 <= nb15((uln-i) DOWNTO  0) & na15(i-1 DOWNTO 0) WHEN nb15(uld+1) = '0' ELSE
              na15;
    END GENERATE;

    g14:
    IF i = 14 GENERATE
      p14:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na14 <= nc15;
          da14 <= da15;
          ra14 <= ra15 & (NOT nb15(uld+1)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb14 <= (gnd(i-dnd DOWNTO 0) & na14(uln DOWNTO i)) - ("0" & da14);
      nc14 <= nb14((uln-i) DOWNTO  0) & na14(i-1 DOWNTO 0) WHEN nb14(uld+1) = '0' ELSE
              na14;
    END GENERATE;

    g13:
    IF i = 13 GENERATE
      p13:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na13 <= nc14;
          da13 <= da14;
          ra13 <= ra14 & (NOT nb14(uld+1)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb13 <= (gnd(i-dnd DOWNTO 0) & na13(uln DOWNTO i)) - ("0" & da13);
      nc13 <= nb13((uln-i) DOWNTO  0) & na13(i-1 DOWNTO 0) WHEN nb13(uld+1) = '0' ELSE
              na13;
    END GENERATE;

    g12:
    IF i = 12 GENERATE
      p12:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na12 <= nc13;
          da12 <= da13;
          ra12 <= ra13 & (NOT nb13(uld+1)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb12 <= (gnd(i-dnd DOWNTO 0) & na12(uln DOWNTO i)) - ("0" & da12);
      nc12 <= nb12((uln-i) DOWNTO  0) & na12(i-1 DOWNTO 0) WHEN nb12(uld+1) = '0' ELSE
              na12;
    END GENERATE;

    g11:
    IF i = 11 GENERATE
      p11:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na11 <= nc12;
          da11 <= da12;
          ra11 <= ra12 & (NOT nb12(uld+1)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb11 <= (gnd(i-dnd DOWNTO 0) & na11(uln DOWNTO i)) - ("0" & da11);
      nc11 <= nb11((uln-i) DOWNTO  0) & na11(i-1 DOWNTO 0) WHEN nb11(uld+1) = '0' ELSE
              na11;
    END GENERATE;

    g10:
    IF i = 10 GENERATE
      p10:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na10 <= nc11;
          da10 <= da11;
          ra10 <= ra11 & (NOT nb11(uld+1)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb10 <= (gnd(i-dnd DOWNTO 0) & na10(uln DOWNTO i)) - ("0" & da10);
      nc10 <= nb10((uln-i) DOWNTO  0) & na10(i-1 DOWNTO 0) WHEN nb10(uld+1) = '0' ELSE
              na10;
    END GENERATE;

    g09:
    IF i = 09 GENERATE
      p09:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na09 <= nc10;
```

```
          da09 <= da10;
          ra09 <= ra10 & (NOT nb10(uld+1)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb09 <= (gnd(i-dnd DOWNTO 0) & na09(uln DOWNTO i)) - ("0" & da09);
      nc09 <= nb09((uln-i) DOWNTO  0) & na09(i-1 DOWNTO 0) WHEN nb09(uld+1) = '0' ELSE
              na09;
    END GENERATE;

    g08:  --Punto de cambio del algoritmo (i=uln-uld-1)----------------------------------------
    IF i = 08 GENERATE
      p08:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na08 <= nc09;
          da08 <= da09;
          ra08 <= ra09 & (NOT nb09(uld+1)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb08 <= ("0" & na08(uld+i+1 DOWNTO i)) - ("00" & da08);
      nc08 <= nb08(uld+1 DOWNTO  0) & na08(i-1 DOWNTO 0) WHEN nb08(uld+2) = '0' ELSE
              na08;
    END GENERATE;

    g07:
    IF i = 07 GENERATE
      p07:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na07 <= nc08;
          da07 <= da08;
          ra07 <= ra08 & (NOT nb08(uld+2)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb07 <= ("0" & na07(uld+i+1 DOWNTO i)) - ("00" & da07);
      nc07 <= gnd((dnd-i-2) DOWNTO 0) & nb07(uld+1 DOWNTO  0) & na07(i-1 DOWNTO 0) WHEN nb07(uld+2) = '0' ELSE
              na07;
    END GENERATE;

    g06:
    IF i = 06 GENERATE
      p06:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na06 <= nc07;
          da06 <= da07;
          ra06 <= ra07 & (NOT nb07(uld+2)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb06 <= ("0" & na06(uld+i+1 DOWNTO i)) - ("00" & da06);
      nc06 <= gnd((dnd-i-2) DOWNTO 0) & nb06(uld+1 DOWNTO  0) & na06(i-1 DOWNTO 0) WHEN nb06(uld+2) = '0' ELSE
              na06;
    END GENERATE;

    g05:
    IF i = 05 GENERATE
      p05:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na05 <= nc06;
          da05 <= da06;
          ra05 <= ra06 & (NOT nb06(uld+2)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb05 <= ("0" & na05(uld+i+1 DOWNTO i)) - ("00" & da05);
      nc05 <= gnd((dnd-i-2) DOWNTO 0) & nb05(uld+1 DOWNTO  0) & na05(i-1 DOWNTO 0) WHEN nb05(uld+2) = '0' ELSE
              na05;
    END GENERATE;

    g04:
    IF i = 04 GENERATE
      p04:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na04 <= nc05;
          da04 <= da05;
          ra04 <= ra05 & (NOT nb05(uld+2)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb04 <= ("0" & na04(uld+i+1 DOWNTO i)) - ("00" & da04);
      nc04 <= gnd((dnd-i-2) DOWNTO 0) & nb04(uld+1 DOWNTO  0) & na04(i-1 DOWNTO 0) WHEN nb04(uld+2) = '0' ELSE
              na04;
    END GENERATE;

    g03:
    IF i = 03 GENERATE
      p03:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na03 <= nc04;
          da03 <= da04;
```

```
          ra03 <= ra04 & (NOT nb04(uld+2)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb03 <= ("0" & na03(uld+i+1 DOWNTO i)) - ("00" & da03);
      nc03 <= gnd((dnd-i-2) DOWNTO 0) & nb03(uld+1 DOWNTO  0) & na03(i-1 DOWNTO 0) WHEN nb03(uld+2) = '0' ELSE
              na03;
    END GENERATE;

    g02:
    IF i = 02 GENERATE
      p02:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na02 <= nc03;
          da02 <= da03;
          ra02 <= ra03 & (NOT nb03(uld+2)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb02 <= ("0" & na02(uld+i+1 DOWNTO i)) - ("00" & da02);
      nc02 <= gnd((dnd-i-2) DOWNTO 0) & nb02(uld+1 DOWNTO  0) & na02(i-1 DOWNTO 0) WHEN nb02(uld+2) = '0' ELSE
              na02;
    END GENERATE;

    g01:
    IF i = 01 GENERATE
      p01:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na01 <= nc02;
          da01 <= da02;
          ra01 <= ra02 & (NOT nb02(uld+2)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb01 <= ("0" & na01(uld+i+1 DOWNTO i)) - ("00" & da01);
      nc01 <= gnd((dnd-i-2) DOWNTO 0) & nb01(uld+1 DOWNTO  0) & na01(i-1 DOWNTO 0) WHEN nb01(uld+2) = '0' ELSE
              na01;
    END GENERATE;

    g00:
    IF i = 00 GENERATE
      p00:
      PROCESS (clk)
      BEGIN
        IF clk= '1' THEN
          na00 <= nc01;
          da00 <= da01;
          ra00 <= ra01 & (NOT nb01(uld+2)); --bits de resultado
          sgn(i) <= sgn(i+1);
        END IF;
      END PROCESS;
      nb00 <= ("0" & na00(uld+i+1 DOWNTO i)) - ("00" & da00);
    END GENERATE;

  END GENERATE;

  r <= sgn(0) & ra00 & (NOT nb00(uld+2)); -- resultado final

--  r <= '0' & ra00 & (NOT nb00(uld+1)) WHEN sgn(0)='0' ELSE -('0' & ra00 & (NOT nb00(uld+1)));  -- Resultado
final

END;
```

B15. FPGABLK: Bloque principal de la FPGA

```
-------------------------------------------------------------------------
-- FPGABLK                                                             --
-- BLOQUE DE CIRCUITO DE ESTIMACION DE FLUJO OPTICO FPGA               --
--                                                                     --
-- Algoritmo:                                                          --
--   *Este bloque VHDL agrupa todo el circuito que debe ir en          --
--    la FPGA                                                          --
--                                                                     --
-- Autor: Aitzol Zuloaga I.                                            --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea   --
--        Departamento de electrónica y telecomunicaciones             --
--                                                                     --
-- Fecha: 01/11/98                                                     --
-------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;

-------------------------------------------------------------------------

ENTITY fpgablk IS
  GENERIC(
    CONSTANT diml:   IN  INTEGER                  :=64   ; -- Dimensión de linea  en pixels
    CONSTANT dimf:   IN  INTEGER                  :=4096 ; -- Dimensión de cuadro en pixels
    CONSTANT dlyves: IN  INTEGER                  :=28   ; -- Retardo en la estimación de velocidad
    CONSTANT dlycp:  IN  INTEGER                  :=10  ); -- Retardo en el calculo coef de promd
```

```
  PORT(
    SIGNAL n_rst:  IN  STD_LOGIC                    :='1'; -- External reset
    SIGNAL n_prst: OUT STD_LOGIC                    :='1'; -- External reset

    -- Entradas principales compatibles con la salida del digitalizador de vídeo
    SIGNAL vdat:   IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Video data from camera
    SIGNAL vclk:   IN  STD_LOGIC                          ; -- Video data clock
    SIGNAL n_hs:   IN  STD_LOGIC                          ; -- Vertical sync
    SIGNAL n_vs:   IN  STD_LOGIC                          ; -- Vertical sync
    SIGNAL dv:     IN  STD_LOGIC                          ; -- Data valid

    -- Entrada de reloj principal de procesamiento
    SIGNAL clk:    IN  STD_LOGIC                          ; -- reloj principal (deberá ser pclk tomado de
afuera del FPGA)

    -- Salida de señales sincronizadas para usarlas con los FIFO y algunas para que sean
    -- reintroducidas a las FPGA para mantener todo el sistema sincronizado
    SIGNAL pdat:   OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Video data for processing
    SIGNAL pclk:   OUT STD_LOGIC                          ; -- Processing clock
    SIGNAL pclkl:  OUT STD_LOGIC                          ; -- Processing clock for external line fifo
    SIGNAL pclkf:  OUT STD_LOGIC                          ; -- Processing clock for external frame fifo
    SIGNAL pclks:  OUT STD_LOGIC                          ; -- Processing clock for external semi-frame fifo
    SIGNAL pclkp:  OUT STD_LOGIC                          ; -- PRUEBA

    -- Entradas de pixels retardados en FIFO externa a la FPGA
    SIGNAL x010:   IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Imagen con retardo de una linea
    SIGNAL x001:   IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Imagen con retardo de un cuadro
    SIGNAL x011:   IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Imagen con retardo de un cuadro y una línea

    -- Entradas de realimentación. Serán las salidas de retardadores FIFO externos.
    SIGNAL vex0:   IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Campo vectorial de velocidad
    SIGNAL vex1:   IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Campo vectorial de velocidad
    SIGNAL vex2:   IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Campo vectorial de velocidad
    SIGNAL vey0:   IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Campo vectorial de velocidad
    SIGNAL vey1:   IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Campo vectorial de velocidad
    SIGNAL vey2:   IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Campo vectorial de velocidad

    -- Entradas de velocidades para convertir
    SIGNAL vcx:    OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Velocidad horizontal formato video
    SIGNAL vcy:    OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Velocidad vertical formato video

    -- Salidas finales del circuito
    SIGNAL n_hso:  OUT STD_LOGIC                          ; -- Vertical sync de salida
    SIGNAL n_vso:  OUT STD_LOGIC                          ; -- Vertical sync de salida
    SIGNAL dvo:    OUT STD_LOGIC                         ); -- Data valid de salida

END fpgablk;

---------------------------------------------------------------------

ARCHITECTURE struct OF fpgablk IS

  COMPONENT sincroni
    GENERIC(
      CONSTANT dlysyn: IN INTEGER                        ); -- Retardo para la salida de sincronismos
    PORT(
      SIGNAL vdat:   IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)    ; -- Video data from camera
      SIGNAL vclk:   IN  STD_LOGIC                         ; -- Video data clock
      SIGNAL n_hs:   IN  STD_LOGIC                         ; -- Horizontal sync
      SIGNAL n_vs:   IN  STD_LOGIC                         ; -- Vertical sync
      SIGNAL dv:     IN  STD_LOGIC                         ; -- Data valid
      SIGNAL n_rst:  IN  STD_LOGIC                         ; -- External reset

      SIGNAL n_hso:  OUT STD_LOGIC                         ; -- Vertical sync de salida
      SIGNAL n_vso:  OUT STD_LOGIC                         ; -- Vertical sync de salida
      SIGNAL dvo:    OUT STD_LOGIC                         ; -- Data valid de salida

      SIGNAL pclk:   OUT STD_LOGIC                         ; -- Processing clock
      SIGNAL n_prst: OUT STD_LOGIC                         ; -- Synchronized reset
      SIGNAL pdat:   OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)    ; -- Video data for processing
      SIGNAL x000:   OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)   ); -- Video data for processing
  END COMPONENT;

  COMPONENT gradext  -- Extractor de gradientes
    PORT(
      SIGNAL a0:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)       ; -- Entradas: pixels actual y anteriores
      SIGNAL a1:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)       ; --           que dependen del gradiente a
      SIGNAL a2:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)       ; --           calcular,
      SIGNAL a3:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)       ;
      SIGNAL a4:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)       ;
      SIGNAL a5:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)       ;
      SIGNAL a6:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)       ;
      SIGNAL a7:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)       ;
      SIGNAL clk: IN  STD_LOGIC                            ; -- reloj de operacion
      SIGNAL e:   OUT STD_LOGIC_VECTOR ( 8 DOWNTO 0))      ; -- Resultado: gradiente
  END COMPONENT;

  COMPONENT delaygr2 -- Obtención de retardos de un pixel a partir de los retardos primarios
    PORT(
      SIGNAL x000: IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ; -- imagen sin retardo
      SIGNAL x010: IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ; -- imagen con retardo de una linea
      SIGNAL x001: IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ; -- imagen con retardo de un cuadro
      SIGNAL x011: IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ; -- imagen con retardo de un cuadro y una línea
      SIGNAL clk:  IN  STD_LOGIC                           ; -- reloj de entrada de datos
      SIGNAL e000: OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ; -- imagen sin retardo
      SIGNAL e010: OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ; -- imagen con retardo de una linea
      SIGNAL e001: OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ; -- imagen con retardo de un cuadro
      SIGNAL e011: OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ; -- imagen con retardo de un cuadro y una línea
      SIGNAL e100: OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ; -- imagen con retardo de un pixel
      SIGNAL e101: OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ; -- imagen con retardo de una linea y un pixel
      SIGNAL e110: OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ; -- imagen con retardo de un cuadro y un pixel
      SIGNAL e111: OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ); -- imagen con retardo de un cuadro, una línea y
un pixel
```

```
END COMPONENT;

COMPONENT cpromed -- Coeficientes de promediado
  PORT(
    SIGNAL v00:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ;
    SIGNAL v01:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ;
    SIGNAL v02:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ;
    SIGNAL v10:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ;
    SIGNAL v12:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ;
    SIGNAL v20:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ;
    SIGNAL v21:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ;
    SIGNAL v22:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)      ;
    SIGNAL clk:  IN  STD_LOGIC                           ; -- reloj de operacion
    SIGNAL cp:  OUT STD_LOGIC_VECTOR ( 8 DOWNTO 0))      ; -- salida coeficiente de promediado
END COMPONENT;

COMPONENT delayvp  -- Retardo de compensacion
  GENERIC(
    CONSTANT ndelays: INTEGER);
  PORT(
    SIGNAL d    :IN  STD_LOGIC_VECTOR (8 DOWNTO 0);
    SIGNAL clk  :IN  STD_LOGIC;
    SIGNAL q    :OUT STD_LOGIC_VECTOR (8 DOWNTO 0));
END COMPONENT;

COMPONENT conver
  PORT(
    SIGNAL vi:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ; -- Velocidad en módulo 2
    SIGNAL clk: IN  STD_LOGIC                      ; -- reloj de operacion
    SIGNAL vc:  OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)); -- Velocidad con offset
END COMPONENT;

COMPONENT ctrfifo
  GENERIC(
    CONSTANT capf: IN  INTEGER                           ); -- Capacidad de la FIFO
  PORT(
    SIGNAL clk:   IN  STD_LOGIC                           ; -- Señal auxiliar de reloj
    SIGNAL n_rst: IN  STD_LOGIC                      :='1' ; -- Señal auxiliar de reset
    SIGNAL clks:  OUT STD_LOGIC                      :='1'); -- Clock for fifo
END COMPONENT;

COMPONENT delaycp2
  PORT(
    SIGNAL ve0:  IN   STD_LOGIC_VECTOR ( 7 DOWNTO 0)    ; -- campo vectorial de velocidad
    SIGNAL ve1:  IN   STD_LOGIC_VECTOR ( 7 DOWNTO 0)    ; -- campo vectorial de velocidad
    SIGNAL ve2:  IN   STD_LOGIC_VECTOR ( 7 DOWNTO 0)    ; -- campo vectorial de velocidad
    SIGNAL clk:  IN   STD_LOGIC                         ; -- reloj de entrada de datos
    SIGNAL v00:  OUT  STD_LOGIC_VECTOR ( 7 DOWNTO 0)    ;
    SIGNAL v01:  OUT  STD_LOGIC_VECTOR ( 7 DOWNTO 0)    ;
    SIGNAL v02:  OUT  STD_LOGIC_VECTOR ( 7 DOWNTO 0)    ;
    SIGNAL v10:  OUT  STD_LOGIC_VECTOR ( 7 DOWNTO 0)    ;
    SIGNAL v11:  OUT  STD_LOGIC_VECTOR ( 7 DOWNTO 0)    ;
    SIGNAL v12:  OUT  STD_LOGIC_VECTOR ( 7 DOWNTO 0)    ;
    SIGNAL v20:  OUT  STD_LOGIC_VECTOR ( 7 DOWNTO 0)    ;
    SIGNAL v21:  OUT  STD_LOGIC_VECTOR ( 7 DOWNTO 0)    ;
    SIGNAL v22:  OUT  STD_LOGIC_VECTOR ( 7 DOWNTO 0))   ;
END COMPONENT;

-- Cálculo del términos P
COMPONENT cpterm
  PORT(
    SIGNAL ex:  IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0) ; -- Gradiente x
    SIGNAL ey:  IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0) ; -- Gradiente y
    SIGNAL et:  IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0) ; -- Gradiente t
    SIGNAL vpx: IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0) ; -- Velocidad horizontal promedio (realimentación)
    SIGNAL vpy: IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0) ; -- Velocidad vertical   promedio (realimentación)
    SIGNAL clk: IN  STD_LOGIC                      ; -- reloj de operacion
    SIGNAL px:  OUT STD_LOGIC_VECTOR (25 DOWNTO 0) ; -- Término Px
    SIGNAL py:  OUT STD_LOGIC_VECTOR (25 DOWNTO 0)); -- Término Py
END COMPONENT;

-- Cálculo del término D
COMPONENT cdterm
  PORT(
    SIGNAL ex:  IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0) ; -- Gradiente x
    SIGNAL ey:  IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0) ; -- Gradiente y
    SIGNAL clk: IN  STD_LOGIC                      ; -- reloj de operacion
    SIGNAL d:   OUT STD_LOGIC_VECTOR (16 DOWNTO 0)); -- Término D
END COMPONENT;

-- Divisor para calcular el término R
COMPONENT divisor
  PORT(
    SIGNAL n:   IN  STD_LOGIC_VECTOR (25 DOWNTO 0) ; -- numerador
    SIGNAL d:   IN  STD_LOGIC_VECTOR (16 DOWNTO 0) ; -- denominador
    SIGNAL clk: IN  STD_LOGIC                      ; -- reloj de operacion
    SIGNAL r:   OUT STD_LOGIC_VECTOR (25 DOWNTO 0)); -- resultado  r=n/d
END COMPONENT;

-- Estimador de velocidad
COMPONENT vestim
  PORT(
    SIGNAL vp:  IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0) ; -- Velocidad promedio v
    SIGNAL rt:  IN  STD_LOGIC_VECTOR (25 DOWNTO 0) ; -- Término r
    SIGNAL clk: IN  STD_LOGIC                      ; -- reloj de operacion
    SIGNAL v:   OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)); -- Velocidad
END COMPONENT;


SIGNAL x000:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)           ; -- imagen sin retardos
SIGNAL e000:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)           ; -- imagen sin retardos
SIGNAL e001:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)           ; -- imagen con retardo de una linea
SIGNAL e010:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)           ; -- imagen con retardo de un cuadro
```

```
  SIGNAL e011:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ; -- imagen con retardo de un cuadro, una línea
  SIGNAL e100:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ; -- imagen con retardo de un pixel
  SIGNAL e101:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ; -- imagen con retardo de una linea y un pixel
  SIGNAL e110:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ; -- imagen con retardo de un cuadro y un pixel
  SIGNAL e111:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ; -- imagen con retardo de un cuadro, una línea y un
pixel

  SIGNAL ex:     STD_LOGIC_VECTOR ( 8 DOWNTO 0)          ; -- Gradiente x
  SIGNAL ey:     STD_LOGIC_VECTOR ( 8 DOWNTO 0)          ; -- Gradiente y
  SIGNAL et:     STD_LOGIC_VECTOR ( 8 DOWNTO 0)          ; -- Gradiente t

  SIGNAL sex:    STD_LOGIC_VECTOR ( 8 DOWNTO 0)          ; -- gradiente x
  SIGNAL sey:    STD_LOGIC_VECTOR ( 8 DOWNTO 0)          ; -- gradiente y
  SIGNAL set:    STD_LOGIC_VECTOR ( 8 DOWNTO 0)          ; -- gradiente t
  SIGNAL rex:    STD_LOGIC_VECTOR ( 8 DOWNTO 0)          ; -- gradiente x latcheado
  SIGNAL rey:    STD_LOGIC_VECTOR ( 8 DOWNTO 0)          ; -- gradiente y latcheado
  SIGNAL ret:    STD_LOGIC_VECTOR ( 8 DOWNTO 0)          ; -- gradiente t latcheado

  SIGNAL sv:     STD_LOGIC_VECTOR (19 DOWNTO 0)          ;

  SIGNAL vx:     STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ; -- Velocidad horizontal
  SIGNAL vy:     STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ; -- Velocidad vertical

  SIGNAL vpxa:   STD_LOGIC_VECTOR ( 8 DOWNTO 0)          ; -- Coef. de promediado previo (auxiliar)
  SIGNAL vpya:   STD_LOGIC_VECTOR ( 8 DOWNTO 0)          ; -- Coef. de promediado previo (auxiliar)

  SIGNAL xpclk:  STD_LOGIC                               ; -- Señal de reloj auxiliar
  SIGNAL n_xrst: STD_LOGIC                               ; -- Señal de reset auxiliar

  SIGNAL vx00:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ; -- Velocidades retardadas para calculo de coef
promed
  SIGNAL vx01:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ;
  SIGNAL vx02:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ;
  SIGNAL vx10:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ;
  SIGNAL vx11:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ;
  SIGNAL vx12:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ;
  SIGNAL vx20:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ;
  SIGNAL vx21:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ;
  SIGNAL vx22:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ;

  SIGNAL vy00:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ; -- Velocidades retardadas para calculo de coef. de
promediado
  SIGNAL vy01:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ;
  SIGNAL vy02:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ;
  SIGNAL vy10:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ;
  SIGNAL vy11:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ;
  SIGNAL vy12:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ;
  SIGNAL vy20:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ;
  SIGNAL vy21:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ;
  SIGNAL vy22:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ;

  SIGNAL d:       STD_LOGIC_VECTOR (16 DOWNTO 0)         ; -- Término D
  SIGNAL px:      STD_LOGIC_VECTOR (25 DOWNTO 0)         ; -- Término Px
  SIGNAL py:      STD_LOGIC_VECTOR (25 DOWNTO 0)         ; -- Término Px
  SIGNAL rx:      STD_LOGIC_VECTOR (25 DOWNTO 0)         ; -- Término Rx
  SIGNAL ry:      STD_LOGIC_VECTOR (25 DOWNTO 0)         ; -- Término Ry

  SIGNAL vpx:     STD_LOGIC_VECTOR ( 8 DOWNTO 0)         ; -- Coef. de promediado de velocidad previa
  SIGNAL vpy:     STD_LOGIC_VECTOR ( 8 DOWNTO 0)         ; -- Coef. de promediado de velocidad previa
  SIGNAL vprx:    STD_LOGIC_VECTOR ( 8 DOWNTO 0)         ; -- Coef. de promediado de velocidad previa
retardado
  SIGNAL vpry:    STD_LOGIC_VECTOR ( 8 DOWNTO 0)         ; -- Coef. de promediado de velocidad previa
retardado


BEGIN

  ----- Sincronizacion --------------------------------------------------------------------------

  -- Sincronización de las señales con el reloj de entrada para todos los bloques del sistema
  usinc:
  sincroni GENERIC MAP (dlyves+1)
           PORT MAP    (vdat, vclk, n_hs, n_vs, dv, n_rst, n_hso, n_vso, dvo, xpclk, n_xrst, pdat, x000);

  pclk <= xpclk;
  n_prst <= n_xrst;

  ----- Gradientes ------------------------------------------------------------------------------

  -- Obtención de los pixels para el cálculo de gradientes
  u000:
  delaygr2 PORT MAP (x000,x010,x001,x011,clk,e000,e010,e001,e011,e100,e101,e110,e111);

  -- Cálculo de los gradientes
  ux:
  gradext PORT MAP(e000, e100, e010, e110, e001, e101, e011, e111, clk, ex);
  uy:
  gradext PORT MAP(e110, e100, e010, e000, e111, e101, e011, e001, clk, ey);
  ut:
  gradext PORT MAP(e000, e001, e010, e011, e100, e101, e110, e111, clk, et);



  ----- Control de FIFOs tanto de promediado como de gradientes ---------------------------------

  uctrfifo1:
  ctrfifo  GENERIC MAP (diml)
           PORT MAP    (xpclk, n_xrst, pclkl);

  uctrfifo2:
  ctrfifo  GENERIC MAP (dimf)
           PORT MAP    (xpclk, n_xrst, pclkf);
```

```
  uctrfifo3:
  ctrfifo  GENERIC MAP (dimf-diml-dlyves-2-dlycp)
           PORT MAP    (xpclk, n_xrst, pclks);

  ----- Coeficientes de promediado ------------------------------------------------------------

  udelaycp21:
  delaycp2 PORT MAP (vex0, vex1, vex2, clk, vx00, vx01, vx02, vx10, vx11, vx12, vx20, vx21, vx22);

  udelaycp22:
  delaycp2 PORT MAP (vey0, vey1, vey2, clk, vy00, vy01, vy02, vy10, vy11, vy12, vy20, vy21, vy22);

  ----- Cálculo de los coeficientes de promediado
  ucpromedx: cpromed PORT MAP (vx00, vx01, vx02, vx10, vx12, vx20, vx21, vx22, clk, vpxa);
  ucpromedy: cpromed PORT MAP (vy00, vy01, vy02, vy10, vy12, vy20, vy21, vy22, clk, vpya);
  vpx <= vpxa;
  vpy <= vpya;

  -- Retardo para compensar el retardo de los divisores
  cvxpr:
  delayvp GENERIC MAP (27) PORT MAP (vpxa,clk,vprx);
  cvypr:
  delayvp GENERIC MAP (27) PORT MAP (vpya,clk,vpry);


  ----- Conversión a niveles de video ---------------------------------------------------------

  uconverx:
  conver PORT MAP (vx,clk,vcx);

  uconvery:
  conver PORT MAP (vy,clk,vcy);

  ----- Terminos DPR --------------------------------------------------------------------------

  -- Cálculo del término D
  ucd:
  cdterm PORT MAP (ex,ey,clk,d);

  -- Cálculo del término P
  ucp:
  cpterm PORT MAP (ex,ey,et,vpx,vpy,clk,px,py);

  -- Cálculo de términos R
  ucrx:
  divisor PORT MAP (px,d,clk,rx); -- Término R horizontal
  ucry:
  divisor PORT MAP (py,d,clk,ry); -- Término R vertical


  -- Cálculo de la velocidad
  uvx:
  vestim PORT MAP (vprx,rx,clk,vx); -- Velocidad horizontal
  uvy:
  vestim PORT MAP (vpry,ry,clk,vy); -- Velocidad vertical


END;
```

B16. GRADEXT: Extractor de gradientes

```
--------------------------------------------------------------------------
-- GRADEXT                                                              --
-- EXTRACTOR DE GRADIENTES                                              --
--                                                                      --
-- Algoritmo:                                                           --
--   *Este módulo toma los diferentes pixels de la/las imágenes y       --
--    calcula la ecuación de gradiente.                                 --
--   *El cálculo se hace para agrupación de pixels espacio-temporal     --
--    de 2x2x2.                                                         --
--   *Dependiendo de cuales pixels sean tomados será el gradiente       --
--    en x, y ó z.                                                      --
--   *La ecuación es la siguiente:                                      --
--          A0 - A1 + A2 - A3 + A4 - A5 + A6 - A7                       --
--                                                                      --
-- Autor: Aitzol Zuloaga I.                                             --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea    --
--        Departamento de electrónica y telecomunicaciones              --
--                                                                      --
-- Fecha: 30/11/98                                                      --
--------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;

ENTITY gradext IS
  PORT(
    SIGNAL a0:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ; -- Entradas: pixels actual y anteriores
    SIGNAL a1:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ; --           que dependen del gradiente a
    SIGNAL a2:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ; --           calcular,
    SIGNAL a3:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL a4:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL a5:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL a6:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL a7:  IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0) ;
    SIGNAL clk: IN  STD_LOGIC                      ; -- reloj de operacion
```

```
    SIGNAL e:   OUT STD_LOGIC_VECTOR ( 8 DOWNTO 0)); -- Resultado: gradiente
END gradext;

ARCHITECTURE behav OF gradext IS

  SIGNAL ssa: STD_LOGIC_VECTOR ( 8 DOWNTO 0) := (OTHERS=>'0');
  SIGNAL ssb: STD_LOGIC_VECTOR ( 8 DOWNTO 0) := (OTHERS=>'0');
  SIGNAL ssc: STD_LOGIC_VECTOR ( 8 DOWNTO 0) := (OTHERS=>'0');
  SIGNAL ssd: STD_LOGIC_VECTOR ( 8 DOWNTO 0) := (OTHERS=>'0');
  SIGNAL sse: STD_LOGIC_VECTOR ( 9 DOWNTO 0) := (OTHERS=>'0');
  SIGNAL ssf: STD_LOGIC_VECTOR ( 9 DOWNTO 0) := (OTHERS=>'0');
  SIGNAL res: STD_LOGIC_VECTOR (10 DOWNTO 0) := (OTHERS=>'0');

BEGIN
  ssa <= ('0' &  a0) + ('0' &  a2);
  ssb <= ('0' &  a4) + ('0' &  a6);
  ssc <= ('0' &  a1) + ('0' &  a3);
  ssd <= ('0' &  a5) + ('0' &  a7);
  sse <= ('0' & ssa) + ('0' & ssb);
  ssf <= ('0' & ssc) + ('0' & ssd);
  res <= ('0' & sse) - ('0' & ssf);

    -- Latcheo de los gradientes
  p1:
  PROCESS (clk) BEGIN
    IF      clk='1' THEN
      e <= res(10 DOWNTO 2);
    END IF;
  END PROCESS;

END;
```

B17. MEMFIFO: Descripción VHDL comportamental de memorias FIFO

```
--------------------------------------------------------------------------
-- MEMFIFO                                                              --
-- Memoria fifo configurable para simulación                            --
--                                                                      --
-- Algoritmo:                                                           --
--  +Trata de representar un banco de memorias del tipo IDT72xx         --
--  +Ya que es una descripción comportamental, los datos se almacenan   --
--   como enteros, con la finalidad de no colapsar la memoria de        --
--   simulación del computador.                                         --
--                                                                      --
-- Autor: Aitzol Zuloaga I.                                             --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea    --
--        Departamento de electrónica y telecomunicaciones              --
--                                                                      --
-- Fecha: 11/04/98                                                      --
--------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;
LIBRARY arithmetic;
USE arithmetic.std_logic_arith.ALL; -- Contiene las conversiones entre integer y std-logic

--------------------------------------------------------------------------

ENTITY memfifo IS
  GENERIC(
    CONSTANT capacity: IN  INTEGER := 256                            ; -- Capacidad de la fifo
    CONSTANT nbytes:   IN  INTEGER := 8                        )     ; -- Número de bytes de la fifo
  PORT(
    SIGNAL   datain:   IN  STD_ULOGIC_VECTOR ( nbytes-1 DOWNTO 0)    ; -- Entrada de datos (8 bits)
    SIGNAL   n_wrclk:  IN  STD_LOGIC                                 ; -- Reloj de escritura
    SIGNAL   n_rdclk:  IN  STD_LOGIC                                 ; -- Reloj de lectura
    SIGNAL   n_reset:  IN  STD_LOGIC                                 ; -- Reset
    SIGNAL   n_ff:     OUT STD_LOGIC                              :='1'; -- Full flag
    SIGNAL   n_ef:     OUT STD_LOGIC                              :='1'; -- Empty flag
    SIGNAL   dataout:  OUT STD_ULOGIC_VECTOR ( nbytes-1 DOWNTO 0))   ; -- Salida de datos
END memfifo;

ARCHITECTURE behav OF memfifo IS
  TYPE memory IS ARRAY (0 TO capacity-1) OF INTEGER;
  SIGNAL ram: memory := (OTHERS => 0);

BEGIN
  pfifo:
  PROCESS (n_wrclk,n_rdclk,n_reset)
    VARIABLE rdpointer: INTEGER :=0;
    VARIABLE wrpointer: INTEGER :=0;
    VARIABLE used:      INTEGER :=0;

  BEGIN
    IF n_reset='0' THEN                          -- Reset
      wrpointer :=  0 ;
      rdpointer :=  0 ;
      used      :=  0 ;
      n_ff      <= '1';
    ELSE
      IF n_wrclk='0' AND n_wrclk'EVENT THEN      -- Write
```

```
        IF used < capacity THEN
          ram (wrpointer) <= TO_INTEGER(datain);
          IF wrpointer<(capacity-1)  THEN
            wrpointer := wrpointer+1;
          ELSE
            wrpointer := 0;
          END IF;
          used := used + 1;
          IF used = capacity THEN
            n_ff <= '0';
          END IF;
          IF used /= 0 THEN
            n_ef <= '1';
          END IF;
        END IF;
      END IF;

      IF n_rdclk='0' AND n_rdclk'EVENT THEN          -- Read
        IF used > 0 THEN
          dataout <= TO_STDULOGICVECTOR(ram(rdpointer),8);
          IF rdpointer<(capacity-1)  THEN
            rdpointer := rdpointer+1;
          ELSE
            rdpointer := 0;
          END IF;
          used := used - 1;
          IF used /= capacity THEN
            n_ff <= '1';
          END IF;
          IF used = 0 THEN
            n_ef <= '0';
          END IF;
       END IF;
      END IF;
    END IF;

  END PROCESS;

END;
```

B18. MULTSGN1: Multiplicador con signo

```
--------------------------------------------------------------------------
-- MULTSGN1                                                             --
-- MULTIPLICADOR CON SIGNO DE 17 BITS                                   --
--                                                                      --
-- Algoritmo:                                                           --
--   *Dado que las multiplicaciones con signo son asumidas de manera    --
--    distinta por el VHDL implementado en ModelSim y en Maxplus,       --
--    se requiere de esta rutina multiplicadora                         --
--   *El resultado tiene un bit más de la cuenta para permitir          --
--    luego las operaciones de suma que no consideran el aumento        --
--    de un bit en VHDL                                                 --
--                                                                      --
-- Autor: Aitzol Zuloaga I.                                             --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea    --
--        Departamento de electrónica y telecomunicaciones              --
--                                                                      --
-- Fecha: 20/06/98                                                      --
--------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;

------------------------------------------------------------------------

ENTITY multsgn1 IS

  PORT(
    SIGNAL m1: IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0) ; -- Primer  multiplicando
    SIGNAL m2: IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0) ; -- Segundo multiplicando
    SIGNAL rm: OUT STD_LOGIC_VECTOR (17 DOWNTO 0)); -- Multiplicacion

END multsgn1;

------------------------------------------------------------------------

ARCHITECTURE struct OF multsgn1 IS
  CONSTANT ulm: INTEGER := m1'HIGH;
  CONSTANT ulr: INTEGER := rm'HIGH;

  SIGNAL sm1:  STD_LOGIC_VECTOR (ulm DOWNTO 0) ;
  SIGNAL sm2:  STD_LOGIC_VECTOR (ulm DOWNTO 0) ;
  SIGNAL srm:  STD_LOGIC_VECTOR (2*ulm+1 DOWNTO 0) ;
  SIGNAL trm:  STD_LOGIC_VECTOR (ulr DOWNTO 0) ;
BEGIN

  sm1 <=  -m1 WHEN m1(ulm)='1' ELSE m1;
  sm2 <=  -m2 WHEN m2(ulm)='1' ELSE m2;

  srm <= sm1 * sm2;

  rm <= srm WHEN (m1(ulm)='0' AND m2(ulm)='0') OR (m1(ulm)='1' AND m2(ulm)='1') ELSE -srm;

END;
```

B19. MULTSGN2: Multiplicador con signo

```
--------------------------------------------------------------------------
-- MULTSGN2                                                             --
-- MULTIPLICADOR CON SIGNO DE 18 BITS x 9 BITS                          --
--                                                                      --
-- Algoritmo:                                                           --
--   *Dado que las multiplicaciones con signo son asumidas de manera    --
--    distinta por el VHDL implementado en ModelSim y en Maxplus,       --
--    se requiere de esta rutina multiplicadora                         --
--   *El resultado tiene un bit más de la cuenta para permitir          --
--    luego las operaciones de suma que no consideran el aumento        --
--    de un bit en VHDL                                                 --
--                                                                      --
-- Autor: Aitzol Zuloaga I.                                             --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea    --
--        Departamento de electrónica y telecomunicaciones              --
--                                                                      --
-- Fecha: 11/06/98                                                      --
--------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;

--------------------------------------------------------------------------

ENTITY multsgn2 IS

  PORT(
    SIGNAL m1: IN  STD_LOGIC_VECTOR (17 DOWNTO 0) ; -- Primer  multiplicando
    SIGNAL m2: IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0) ; -- Segundo multiplicando
    SIGNAL rm: OUT STD_LOGIC_VECTOR (25 DOWNTO 0)); -- Multiplicacion

END multsgn2;

--------------------------------------------------------------------------

ARCHITECTURE struct OF multsgn2 IS
  CONSTANT ulm1: INTEGER := m1'HIGH;
  CONSTANT ulm2: INTEGER := m2'HIGH;
  CONSTANT ulr: INTEGER := rm'HIGH;

  SIGNAL sm1:  STD_LOGIC_VECTOR (ulm1 DOWNTO 0) ;
  SIGNAL sm2:  STD_LOGIC_VECTOR (ulm2 DOWNTO 0) ;
  SIGNAL srm:  STD_LOGIC_VECTOR (ulm1+ulm2+1 DOWNTO 0) ;
  SIGNAL trm:  STD_LOGIC_VECTOR (ulr DOWNTO 0) ;
BEGIN

  sm1 <=  -m1 WHEN m1(ulm1)='1' ELSE m1;
  sm2 <=  -m2 WHEN m2(ulm2)='1' ELSE m2;

  srm <= sm1 * sm2;
  trm <= srm (ulr DOWNTO 0);

  rm <= trm WHEN (m1(ulm1)='0' AND m2(ulm2)='0') OR (m1(ulm1)='1' AND m2(ulm2)='1') ELSE -trm;

END;
```

B20. MULTSGN3: Multiplicador con signo

```
--------------------------------------------------------------------------
-- MULTSGN3                                                             --
-- MULTIPLICADOR CON SIGNO DE 17 BITS PARA ELEVACION AL CUADRADO        --
--                                                                      --
-- Algoritmo:                                                           --
--   *Dado que las multiplicaciones con signo son asumidas de manera    --
--    distinta por el VHDL implementado en ModelSim y en Maxplus,       --
--    se requiere de esta rutina multiplicadora                         --
--   *Como esta rutina está planteada para elevar al cuadrado           --
--    sólo requiere una entrada, que puede es  con signo, y el          --
--    resultado será siempre positivo (bit MSb siempre 0)               --
--   *Lo del último bit en cero no debería hacer falta pero las         --
--    incompatibilidades entre Altera y ModelSim en la interpretacion   --
--    del número de bits de resultado de una suma lo hacen necesario    --
--                                                                      --
-- Autor: Aitzol Zuloaga I.                                             --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea    --
--        Departamento de electrónica y telecomunicaciones              --
--                                                                      --
-- Fecha: 26/06/98                                                      --
--------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;

--------------------------------------------------------------------------

ENTITY multsgn3 IS

  PORT(
    SIGNAL m:  IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0) ; -- Multiplicando
    SIGNAL rm: OUT STD_LOGIC_VECTOR (16 DOWNTO 0)); -- Resultado de elevar al cuadrado

END multsgn3;
```

```
--------------------------------------------------------------------

ARCHITECTURE struct OF multsgn3 IS
  CONSTANT ulm: INTEGER := m'HIGH;
  CONSTANT ulr: INTEGER := rm'HIGH;

  SIGNAL mp:  STD_LOGIC_VECTOR (ulm DOWNTO 0);
  SIGNAL rp:  STD_LOGIC_VECTOR (ulr+1 DOWNTO 0);

BEGIN

  mp <= -m WHEN m(ulm)='1' ELSE m;
  rp <= mp * mp;
  rm <= rp (ulr DOWNTO 0);

END;
```

B21. SINCRONI: Generacion de señales de reloj y sincronismo

```
--------------------------------------------------------------------------
-- SINCRONI                                                             --
-- GENERACION DE SEÑALES DE RELOJ Y SINCRONISMO Y SINCRONIZACION        --
-- DE DATOS                                                             --
--                                                                      --
-- Algoritmo:                                                           --
--   *Las señales de reloj que se usan para el procesamiento  de        --
--    los pixels de las imágenes                                        --
--   *La señal de procesamiento solo funciona cuando la señal de        --
--    dato válido está activa.                                          --
--                                                                      --
-- Autor: Aitzol Zuloaga I.                                             --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea    --
--        Departamento de electrónica y telecomunicaciones              --
--                                                                      --
-- Fecha: 18/12/98                                                      --
--------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;

--------------------------------------------------------------------------

ENTITY sincroni IS
  GENERIC(
    CONSTANT dlysyn: IN INTEGER                          :=29); -- Retardo para la salida de sincronismos

  PORT(
    SIGNAL vdat:   IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)         ; -- Video data from camera
    SIGNAL vclk:   IN  STD_LOGIC                              ; -- Video data clock
    SIGNAL n_hs:   IN  STD_LOGIC                              ; -- Vertical sync
    SIGNAL n_vs:   IN  STD_LOGIC                              ; -- Vertical sync
    SIGNAL dv:     IN  STD_LOGIC                              ; -- Data valid
    SIGNAL n_rst:  IN  STD_LOGIC                              ; -- External reset

    SIGNAL n_hso:  OUT STD_LOGIC                              ; -- Vertical sync de salida
    SIGNAL n_vso:  OUT STD_LOGIC                              ; -- Vertical sync de salida
    SIGNAL dvo:    OUT STD_LOGIC                              ; -- Data valid de salida

    SIGNAL pclk:   OUT STD_LOGIC                              ; -- Processing clock
    SIGNAL n_prst: OUT STD_LOGIC                         :='1'; -- Synchronized reset
    SIGNAL pdat:   OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)         ; -- Video data for processing
    SIGNAL x000:   OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)        ); -- Video data for processing
END sincroni;

--------------------------------------------------------------------------

ARCHITECTURE descrip OF sincroni IS

  COMPONENT delaysi
    GENERIC(
      CONSTANT ndelays: INTEGER := 26);
    PORT(
      SIGNAL d    :IN  STD_LOGIC ;
      SIGNAL clk  :IN  STD_LOGIC ;
      SIGNAL q    :OUT STD_LOGIC);
  END COMPONENT;

  SIGNAL xdat:   STD_LOGIC_VECTOR ( 7 DOWNTO 0)               ; -- Señal auxiliar de datos de video
  SIGNAL xclk:   STD_LOGIC                                    ; -- Señal auxiliar de reloj (=pclk)
  SIGNAL xdv:    STD_LOGIC                                    ; -- Señal auxiliar de dv
  SIGNAL n_xrst: STD_LOGIC                               :='1'; -- Señal auxiliar de reset sincronizado
(=n_prst)
  SIGNAL crst:   STD_LOGIC                               :='0'; -- Señal auxiliar de reset sincronizado para
relojs de fifos
  SIGNAL cnt:    STD_LOGIC_VECTOR (16 DOWNTO 0):=(OTHERS=>'0'); -- Contador de pulsos de reloj para línea
  SIGNAL scl:    STD_LOGIC                               :='0'; -- Flag de comienzo de reloj para líneas
  SIGNAL scf:    STD_LOGIC                               :='0'; -- Flag de comienzo de reloj para cuadros
BEGIN

  ----------
  -- Generación de los relojes de procesamiento en base al reloj principal
  -- y el data valid
  PROCESS (vclk) BEGIN
    IF vclk'event and vclk = '1' THEN
      xdv <= dv;
    end if;
```

```
  END PROCESS;
  xclk   <=     vclk AND xdv AND dv ;
  pclk   <=     (vclk AND xdv AND dv) OR crst ; -- reloj de procesamiento principal

  ----------
  -- Generación de señales de dato válido y sincronismos desplazados
  udv:
  delaysi GENERIC MAP (dlysyn) PORT MAP (dv,vclk,dvo);
  uhs:
  delaysi GENERIC MAP (dlysyn) PORT MAP (n_hs,vclk,n_hso);
  uvs:
  delaysi GENERIC MAP (dlysyn) PORT MAP (n_vs,vclk,n_vso);

  ----------
  -- Obtención del dato sincronizado (según la especificación de la "camara"
  -- el dato debe ser tomado en el flanco de subida de la señal)
  PROCESS (xclk) BEGIN
    IF xclk = '1' THEN
      xdat <= vdat;
    END IF;
  END PROCESS;
  pdat <= xdat;

  ----------
  -- Obtención del dato del dato sin retardo sincronizado con el
  -- flanco de bajada de la señal, tal y como vendrán el resto
  -- de los datos retardados
  PROCESS (xclk) BEGIN
    IF xclk = '0' THEN
      x000 <= xdat;
    END IF;
  END PROCESS;

  ----------
  -- Generación de un reset sincronizado con el sincronismo vertical
  PROCESS (n_vs,n_rst) BEGIN
    IF n_rst='0' THEN
      n_xrst <= '0';
    ELSE
      IF n_vs = '1' AND n_vs'EVENT THEN
        n_xrst <= n_rst;
      END IF;
    END IF;
  END PROCESS;
  n_prst <= n_xrst; -- reset general interno del sistema

  ----------
  -- Generación de un reset sincronizado con el reloj de escritura de las fifos
  -- dado que éstos requieren que durante el reset se mantenga el reloj a uno
  PROCESS (xclk,n_rst) BEGIN
    IF n_rst='0' THEN
      crst <= '1';
    ELSE
      IF xclk = '1' AND xclk'EVENT THEN
        crst <= not(n_rst);
      END IF;
    END IF;
  END PROCESS;

END;
```

B22. SYSTEM: Sistema completo y banco de pruebas

```
--------------------------------------------------------------------------
-- SYSTEM                                                               --
-- sistema para la determinación del movimiento                         --
--                                                                      --
-- Algoritmo:                                                           --
--                                                                      --
-- Autor: Aitzol Zuloaga I.                                             --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea    --
--        Departamento de electrónica y telecomunicaciones              --
--                                                                      --
-- Fecha: 27/11/98                                                      --
--------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;

--------------------------------------------------------------------------

ENTITY system IS
  GENERIC(
    CONSTANT diml:    IN  INTEGER                 :=   64 ; -- Dimension de línea en pixels (256)
    CONSTANT dimf:    IN  INTEGER                 := 4096 ; -- Dimensión de cuadro en pixels (65536)
    CONSTANT filec:   IN  STRING        :="u:imagenes.hex" ; -- Archivo de imágenes de cámara
    CONSTANT filemx:  IN  STRING        :="u:motion.hxx"   ; -- Archivo destino de imágenes monitor x
    CONSTANT filemy:  IN  STRING        :="u:motion.hxy"  ); -- Archivo destino de imágenes monitor y
  PORT(
    SIGNAL   n_rst:   IN  STD_LOGIC               :=  '1'); -- External reset
END system;

--------------------------------------------------------------------------

ARCHITECTURE struct OF system IS
```

```
  COMPONENT camera
    GENERIC(
      CONSTANT filec:   IN  STRING                        ); -- Archivo de imágenes de cámara
    PORT(
      SIGNAL vdat:      OUT STD_LOGIC_VECTOR (7 DOWNTO 0)  ; -- Datos
      SIGNAL clk:       OUT STD_LOGIC                      ; -- Reloj
      SIGNAL n_hs:      OUT STD_LOGIC                      ; -- Sincronismo horizontal
      SIGNAL n_vs:      OUT STD_LOGIC                      ; -- Sincronismo vertical
      SIGNAL dv:        OUT STD_LOGIC                     ); -- Dato valido
  END COMPONENT;

  COMPONENT monitor
    GENERIC(
      CONSTANT filem:   IN  STRING                        ); -- Archivo destino de imágenes
    PORT(
      SIGNAL   vdat:    IN STD_LOGIC_VECTOR (7 DOWNTO 0)   ; -- Datos
      SIGNAL   clk:     IN STD_LOGIC                := '0' ; -- Reloj
      SIGNAL   n_hs:    IN STD_LOGIC                := '1' ; -- Sincronismo horizontal
      SIGNAL   n_vs:    IN STD_LOGIC                := '1' ; -- Sincronismo vertical
      SIGNAL   dv:      IN STD_LOGIC                := '0'); -- Dato valido
  END COMPONENT;

  COMPONENT fpgablk
  GENERIC(
    CONSTANT diml:      IN  INTEGER                :=64   ; -- Dimensión de linea  en pixels
    CONSTANT dimf:      IN  INTEGER                :=4096); -- Dimensión de cuadro en pixels

  PORT(
    SIGNAL n_rst:       IN  STD_LOGIC              :='1'; -- External reset
    SIGNAL n_prst:      OUT STD_LOGIC              :='1'; -- External reset

    -- Entradas principales compatibles con la salida del digitalizador de vídeo
    SIGNAL vdat:        IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Video data from camera
    SIGNAL vclk:        IN  STD_LOGIC                          ; -- Video data clock
    SIGNAL n_hs:        IN  STD_LOGIC                          ; -- Vertical sync
    SIGNAL n_vs:        IN  STD_LOGIC                          ; -- Vertical sync
    SIGNAL dv:          IN  STD_LOGIC                          ; -- Data valid

    -- Entrada de reloj principal de procesamiento
    SIGNAL clk:         IN  STD_LOGIC                          ; -- reloj principal (deberá ser pclk tomado de
afuera del FPGA)

    -- Salida de señales sincronizadas para usarlas con los FIFO y algunas para que sean
    -- reintroducidas a las FPGA para mantener todo el sistema sincronizado
    SIGNAL pdat:        OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Video data for processing
    SIGNAL pclk:        OUT STD_LOGIC                          ; -- Processing clock
    SIGNAL pclkl:       OUT STD_LOGIC                          ; -- Processing clock for external line fifo
    SIGNAL pclkf:       OUT STD_LOGIC                          ; -- Processing clock for external frame fifo
    SIGNAL pclks:       OUT STD_LOGIC                          ; -- Processing clock for external semi-frame
fifo
    SIGNAL pclkp:       OUT STD_LOGIC                          ; -- PRUEBA

    -- Entradas de pixels retardados en FIFO externa a la FPGA
    SIGNAL x010:        IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Imagen con retardo de una linea
    SIGNAL x001:        IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Imagen con retardo de un cuadro
    SIGNAL x011:        IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Imagen con retardo de un cuadro y una
línea

    -- Entradas de realimentación. Serán las salidas de retardadores FIFO externos.
    SIGNAL vex0:        IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Campo vectorial de velocidad
    SIGNAL vex1:        IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Campo vectorial de velocidad
    SIGNAL vex2:        IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Campo vectorial de velocidad
    SIGNAL vey0:        IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Campo vectorial de velocidad
    SIGNAL vey1:        IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Campo vectorial de velocidad
    SIGNAL vey2:        IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Campo vectorial de velocidad

    -- Entradas de velocidades para convertir
    SIGNAL vcx:         OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Velocidad horizontal formato video
    SIGNAL vcy:         OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)     ; -- Velocidad vertical formato video

    -- Salidas finales del circuito
    SIGNAL n_hso:       OUT STD_LOGIC                          ; -- Vertical sync de salida
    SIGNAL n_vso:       OUT STD_LOGIC                          ; -- Vertical sync de salida
    SIGNAL dvo:         OUT STD_LOGIC                         ); -- Data valid de salida

  END COMPONENT;


  COMPONENT delaygr1
    GENERIC(
      CONSTANT diml: IN  INTEGER                          ; -- Dimension de línea en pixels
      CONSTANT dimf: IN  INTEGER                         ); -- Dimensión de cuadro en pixels
    PORT(
      SIGNAL pdat:   IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)    ; -- secuencia de imágenes
      SIGNAL n_prst: IN  STD_LOGIC                         ; -- Reset
      SIGNAL pclk:   IN  STD_LOGIC                         ; -- Processing clock for fifo write
      SIGNAL pclkl:  IN  STD_LOGIC                         ; -- Processing clock for line fifo read
      SIGNAL pclkf:  IN  STD_LOGIC                         ; -- Processing clock for frame fifo read
      SIGNAL e010:   OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)    ; -- imagen con retardo de una linea
      SIGNAL e001:   OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)    ; -- imagen con retardo de un cuadro
      SIGNAL e011:   OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)   ); -- imagen con retardo de un cuadro y una línea
  END COMPONENT;

  COMPONENT delaycp3
    GENERIC(
      CONSTANT diml: IN  INTEGER                          ; -- Dimension de línea en pixels 256
      CONSTANT dimf: IN  INTEGER                         ); -- Dimensión de cuadro en pixels 65536
    PORT(
      SIGNAL v:      IN  STD_LOGIC_VECTOR ( 7 DOWNTO 0)    ; -- secuencia de velocidades
      SIGNAL n_prst: IN  STD_LOGIC                         ; -- Reset
      SIGNAL pclk:   IN  STD_LOGIC                         ; -- Reloj de procesamiento para escritura en fifo
      SIGNAL pclkl:  IN  STD_LOGIC                         ; -- Reloj de procesamiento para lectura de fifo de
líneas
```

```
      SIGNAL pclks:  IN  STD_LOGIC                        ; -- Reloj de procesamiento para lectura de fifo de
semi-cuadros
      SIGNAL ve0:    OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)    ; -- campo vectorial de velocidad
      SIGNAL ve1:    OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)    ; -- campo vectorial de velocidad
      SIGNAL ve2:    OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)   ); -- campo vectorial de velocidad
  END COMPONENT;

  -- Conexiones de la cámara
  SIGNAL vdat:    STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ; -- Video data from camera
  SIGNAL vclk:    STD_LOGIC                               ; -- Video data clock
  SIGNAL n_hs:    STD_LOGIC                               ; -- Sincronismo horizontal
  SIGNAL n_vs:    STD_LOGIC                               ; -- Sincronismo vertical
  SIGNAL dv:      STD_LOGIC                               ; -- Data valid


  -- Conexiones a los retardadores externos para el cálculo de laplacianos
  SIGNAL vex0:    STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ; -- Campo vectorial de velocidad
  SIGNAL vex1:    STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ; -- Campo vectorial de velocidad
  SIGNAL vex2:    STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ; -- Campo vectorial de velocidad
  SIGNAL vey0:    STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ; -- Campo vectorial de velocidad
  SIGNAL vey1:    STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ; -- Campo vectorial de velocidad
  SIGNAL vey2:    STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ; -- Campo vectorial de velocidad

  SIGNAL pdat:    STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ; -- Video data for processing
  SIGNAL pclk:    STD_LOGIC                               ; -- Processing clock


  SIGNAL vx:      STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ; -- velocidad horizontal
  SIGNAL vy:      STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ; -- velocidad vertical
  SIGNAL n_prst:  STD_LOGIC                               ; -- Reset
  SIGNAL pclkl:   STD_LOGIC                               ; -- Processing clock for line fifo read
  SIGNAL pclkf:   STD_LOGIC                               ; -- Processing clock for frame fifo read
  SIGNAL pclks:   STD_LOGIC                               ; -- Processing clock for frame fifo read

  SIGNAL n_hso:   STD_LOGIC                               ; -- Vertical sync de salida
  SIGNAL n_vso:   STD_LOGIC                               ; -- Vertical sync de salida
  SIGNAL dvo:     STD_LOGIC                               ; -- Data valid de salida

  -- Salida hacia los monitores
  SIGNAL vcx:     STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ; -- Velocidad horizontal formato video
  SIGNAL vcy:     STD_LOGIC_VECTOR ( 7 DOWNTO 0)          ; -- Velocidad vertical formato video

BEGIN

  -- Cámara: toma las imágenes del archivo y las inyecta como señales
  cam:
  camera GENERIC MAP (filec)
         PORT MAP    (vdat, vclk, n_hs, n_vs, dv);

  -- Retardadores de líneas y cuadro externos a la FPGA
  udelaygr1:
  delaygr1 GENERIC MAP (diml, dimf)
           PORT MAP    (pdat, n_prst, pclk, pclkl, pclkf, x010, x001, x011);

  udelaycpx:
  delaycp3 GENERIC MAP (diml, dimf)
           PORT MAP    (vx, n_prst, pclk, pclkl, pclks, vex0, vex1, vex2);

  udelaycpy:
  delaycp3 GENERIC MAP (diml, dimf)
           PORT MAP    (vy, n_prst, pclk, pclkl, pclks, vey0, vey1, vey2);

  -- Bloque dentro de FPGA
  fpga:
  fpgablk GENERIC MAP (diml, dimf)
           PORT MAP    (n_rst, n_prst,
                        vdat, vclk, n_hs, n_vs, dv,
                        pclk,
                        pdat, pclk, pclkl, pclkf, pclks,
                        x010, x001, x011,
                        vex0, vex1, vex2, vey0, vey1, vey2,
                        vcx, vcy,
                        n_hso, n_vso, dvo);


  -- Monitor: toma las señales generadas y las escribe a un archivo
  umonitorx:
  monitor GENERIC MAP (filemx)
          PORT MAP    (vcx, vclk, n_hso, n_vso, dvo);

  umonitory:
  monitor GENERIC MAP (filemy)
          PORT MAP    (vcy, vclk, n_hso, n_vso, dvo);

END;
```

B23. VESTIM: Estimador de velocidad

```
--------------------------------------------------------------------------
-- VESTIM                                                               --
-- ESTIMADOR DE VELOCIDAD V = Vp - R                                    --
--                                                                      --
-- Algoritmo:                                                           --
--   *Este módulo realiza el último cálculo para estimar la velocidad --
--   *Antes de realizar la operación matemática realiza un trunca-     --
--    miento en el número de bits del término r                         --
--   *La velocidad resultante se calcula con 7 bits más signo           --
```

```
--                                                                      --
-- Autor: Aitzol Zuloaga I.                                             --
--        Universidad del País Vasco / Euskal Herriko Unibertsitatea    --
--        Departamento de electrónica y telecomunicaciones              --
--                                                                      --
-- Fecha: 29/06/98                                                      --
--------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;

--------------------------------------------------------------------------

ENTITY vestim IS
  PORT(
    SIGNAL vp:  IN  STD_LOGIC_VECTOR ( 8 DOWNTO 0) ; -- Velocidad promedio v
    SIGNAL rt:  IN  STD_LOGIC_VECTOR (25 DOWNTO 0) ; -- Término r
    SIGNAL clk: IN  STD_LOGIC                      ; -- reloj de operacion
    SIGNAL v:   OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0)); -- Velocidad
END vestim;

--------------------------------------------------------------------------

ARCHITECTURE struct OF vestim IS
  SIGNAL sva: STD_LOGIC_VECTOR (8 DOWNTO 0);
  SIGNAL svb: STD_LOGIC_VECTOR (8 DOWNTO 0);
  SIGNAL svc: STD_LOGIC_VECTOR (8 DOWNTO 0);
  SIGNAL v_a: STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
  -- Se trunca rt, si es mayor a 128. Por supuesto el bit de signo no se considera
  sva(6 DOWNTO 0) <= rt(6 DOWNTO 0) WHEN rt(24 DOWNTO 7) = "0" ELSE "1111111";
  sva(8 DOWNTO 7) <= "00";

  -- Se recupera la notación en complemento a 2 considerando el signo.
  -- Como hay que hacer una resta, aquí mismo se cambia el signo, por eso se pone positivo cuando
  -- el signo de rt es negativo, y viceversa.
  svb <= -sva WHEN rt(25)='0' ELSE sva;

  -- Se efectúa la suma
  svc <= vp + svb;

  -- Se trunca el resultado a 7 bits más signo
  v_a <= "01111111" WHEN svc(8 DOWNTO 7)="01" ELSE
         "10000001" WHEN svc(8 DOWNTO 7)="10" ELSE
         svc(7 DOWNTO 0);

  -- Se sincroniza el resultado
  PROCESS (clk)
    BEGIN
      IF clk= '1' THEN
        v <= v_a;
      END IF;
  END PROCESS;

END;
```

Apéndice C. Descripciones VHDL del banco de pruebas

C1. CAMERA: Cámara digital

```
--------------------------------------------------------------------------------
-- CAMERA                                                                     --
--                                                                            --
-- Camera emulator for image processing test-bench                            --
--                                                                            --
-- Version 3.0                                                                --
-- Universidad del País Vasco / Euskal Herriko Unibertsitatea                 --
--                                                                            --
-- Author: Aitzol Zuloaga I.                                                  --
--                                                                            --
-- Date: 31/07/99                                                             --
--                                                                            --
-- Algorithm:                                                                 --
--  +Se toman imágenes almacenadas en formato HEX específico para             --
--   esta aplicación.                                                         --
--  +Se generan a partir del archivo HEX las señales digitales                --
--   utilizadas generalmente para aplicaciones de procesamiento               --
--   de vídeo: datos, sincronismos vertical y horizontal, indicación          --
--   de dato válido y reloj de muestreo de datos.                             --
--  +El arhivo hex contiene un par de caracteres por cada dato,               --
--   (0-9 y A-F). El sincronismo horizontal se representa por                 --
--   el par de bytes CR-LF, y el sinronismo vertical por un *                 --
--                                                                            --
--                                                                            --
--------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.pkvideo.all;                           -- Hex-video format mannaging library

------------------------------------------------------------------------

ENTITY camera IS

  GENERIC(
    CONSTANT filec:    IN  STRING  := "imagprue2.hex"; -- Hex-video file name
    CONSTANT tclk:     IN  TIME    := 100 ns;          -- Clock speed
    CONSTANT ths:      IN  INTEGER := 3;               -- Horizontal sync time in clock cycles
    CONSTANT tvs:      IN  INTEGER := 29;              -- Vertical sync time in clock cycles
    CONSTANT tlin:     IN  INTEGER := 8);              -- Line time for vertical sync generation in clk cycles

  PORT(
    SIGNAL      vdat:      OUT STD_LOGIC_VECTOR (7 DOWNTO 0);   -- Data
    SIGNAL      clk:       OUT STD_LOGIC;                       -- Clock
    SIGNAL      n_hs:      OUT STD_LOGIC;                       -- Horizontal sync
    SIGNAL      n_vs:      OUT STD_LOGIC;                       -- Vertical sync
    SIGNAL      dv:        OUT STD_LOGIC);                      -- Data valid

END camera;

------------------------------------------------------------------------------

ARCHITECTURE behav OF camera IS

  SIGNAL      clkx:      STD_LOGIC      := '0';
  FILE           image:      hex_image_file IS IN filec;      -- File open

BEGIN

  ------------------------------------------------------------------------------
  ---------- Clock generator
  pclk:
  PROCESS
  BEGIN
    WAIT FOR tclk;
    clkx <= NOT(clkx);
    clk <= clkx;
  END PROCESS;


  ------------------------------------------------------------------------------
  ---------- Camera simulation
  pcamera:
  PROCESS (clkx)
    VARIABLE idat:      STD_LOGIC_VECTOR (7 DOWNTO 0)       ;
    VARIABLE hs_b:      STD_LOGIC                    :='0'      ;
    VARIABLE vs_b:      STD_LOGIC                    :='0'      ;
    VARIABLE dv_b:      STD_LOGIC                    :='0'      ;
    VARIABLE cnt1:      INTEGER                  := 0       ; -- Contador de generación de sinc. horizontal
    VARIABLE cnt2:      INTEGER                     := 0       ; -- Contador de generación de sinc. vertical
    VARIABLE cnt3:      INTEGER                      := 0       ; -- Contador de generación de sinc hor.
dentro de sinc. vertical
```

```
    VARIABLE chr1:      CHARACTER                 ;
    VARIABLE chr2:      CHARACTER                 ;
    VARIABLE hex_nib:       hex                      ;
  BEGIN


    IF clkx = '0' THEN           -- Cambios en el frente de bajada del reloj


      ------------------------------------------------------------------------
      -- Data process time, no vertical or horizontal syncs

      IF cnt1=0 AND cnt2=0 THEN


        IF not (ENDFILE (image)) THEN

          READ (image,chr1);                    -- First character is read


          IF (chr1 = CR) OR (chr1 = LF) THEN    -- CR-LF detected, horizontal sync must be generated
            READ (image,chr2);                     -- Second carachter is read. It must be Line Feed
            vdat <= (OTHERS => '0');            -- Data is cleared
            cnt1 := ths;                  -- Horizontal sync time
            n_hs <= '0';                  -- Horizontal sync
            dv   <= '0';                        -- Data no valid

          ELSIF chr1 = '*' THEN                 -- '*' detected, vertical sync must be generated
            vdat <= (OTHERS => '0');            -- Data is cleared
            cnt2 := tvs;                  -- Vertical sync time
            cnt3 := tlin;                  -- Line time for vertical sync
            n_vs <= '0';                  -- Vertical sync
            dv   <= '0';                        -- Data no valid

          ELSE                                  -- Hex data
            chr2hex(chr1,hex_nib);              -- First character is converted to STD_LOGIC
            hex2std(hex_nib,idat(7 DOWNTO 4));
            READ (image,chr2);                  -- Second carachter is read and
            chr2hex(chr2,hex_nib);              -- converted to STD_LOGIC
            hex2std(hex_nib,idat(3 DOWNTO 0));
            vdat <= idat;                       -- Data out
            n_hs <= '1';                  -- No horizontal sync
            n_vs <= '1';                  -- No vertical sync
            dv   <= '1';                        -- Data valid

          END IF;

        ELSE                                    -- Si se alcanzó el final del archivo

          ASSERT (false)
          REPORT "File end reached...";         -- Console message
          n_hs <= '1';                            -- No horizontal sync
          n_vs <= '0';                          -- Vertical sync
          dv   <= '0';                          -- Data no valid

        END IF;

      END IF;


      ------------------------------------------------------------------------
      -- Horizontal sync generation

      IF cnt1>0 THEN
        n_hs <= '0';
        cnt1 := cnt1-1;                     -- Clock count for horizontal sync time
      ELSE
        n_hs <= '1';
      END IF;


      ------------------------------------------------------------------------
      -- Vertical sync generation

      IF cnt2>0 THEN
        n_vs <= '0';
        cnt2 := cnt2-1;                       -- Clock count for vertical sync time
      cnt3 := cnt3-1;                      -- Clock count for horizontal sync time inside vertical sync
      ELSE
        n_vs <= '1';
      END IF;


      ------------------------------------------------------------------------
      -- Horizontal sync generation inside vertical sync

      IF cnt3=0 AND cnt2>0 THEN
        n_hs <= '0';
        cnt1 := ths-1;
        cnt3 := tlin+ths;
      END IF;

    END IF;

  END PROCESS;

END behav;
```

C2. MONITOR: Monitor digital

```
--------------------------------------------------------------------------------
-- MONITOR                                                                     --
--                                                                             --
-- Monitor emulator for image processing test-bench                            --
--                                                                             --
-- Version 3.0                                                                 --
-- Universidad del País Vasco / Euskal Herriko Unibertsitatea                  --
--                                                                             --
-- Author: Aitzol Zuloaga I.                                                   --
--                                                                             --
-- Date: 31/07/99                                                              --
--                                                                             --
-- Algorithm:                                                                  --
--  +Se toman las imágenes recibidas y se almacenan en formato HEX             --
--   específico para esta aplicación.                                          --
--  +Se reciben las señales digitales utilizadas generalmente para             --
--   aplicaciones de procesamiento de vídeo: datos, sincronismos               --
--   vertical y horizontal, indicación de dato válido y reloj de               --
--   muestreo de datos y se genera un archivo HEX específico.                  --
--  +El arhivo hex contiene un par de caracteres por cada dato,                --
--   (0-9 y A-F). El sincronismo horizontal se representa por                  --
--   el par de bytes CR-LF, y el sinronismo vertical por un *                  --
--                                                                             --
--------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.pkvideo.all       ; -- Libreria de manejo de archivos de video HEX


--------------------------------------------------------------------------------

ENTITY monitor IS
  GENERIC(
    CONSTANT file_name:   IN  STRING  := "u:prueba2.hex" )           ; -- Archivo destino de imágenes

  PORT(
    SIGNAL   vdat:    IN STD_LOGIC_VECTOR (7 DOWNTO 0)               ; -- Datos
    SIGNAL   clk:     IN STD_LOGIC                          := '0' ; -- Reloj
    SIGNAL   n_hs:    IN STD_LOGIC                          := '1' ; -- Sincronismo horizontal
    SIGNAL   n_vs:    IN STD_LOGIC                          := '1' ; -- Sincronismo vertical
    SIGNAL   dv:      IN STD_LOGIC                          := '0'); -- Dato valido

END monitor;

--------------------------------------------------------------------------------

ARCHITECTURE behav OF monitor IS

  FILE     image:     hex_image_file IS OUT file_name                ;

BEGIN

  ---------- Proceso de escritura de datos válidos
  pm1:
  PROCESS (clk)
   VARIABLE nib:         STD_LOGIC_VECTOR (3 DOWNTO 0);
   VARIABLE hex_nib:     hex;
   VARIABLE chr:         CHARACTER;

  BEGIN
    IF clk='1' THEN --frente de subida
      IF dv = '1' THEN                        -- Si es un dato válido se escribe en HEX
        nib := vdat(7 DOWNTO 4);
        std2hex(nib,hex_nib);
        hex2chr(hex_nib,chr);
        WRITE(image,chr);
        std2hex(vdat(3 DOWNTO 0),hex_nib);
        hex2chr(hex_nib,chr);
        WRITE(image,chr);
      END IF;

    END IF;
  END PROCESS;

  -- Pulso de sincronismo horizontal
  pm2:
  PROCESS (n_hs)
  BEGIN
    IF (n_hs'EVENT AND n_hs='0' AND n_vs='1') THEN
      WRITE(image,CR);
      WRITE(image,LF);
    END IF;
  END PROCESS;

  -- Pulso de sincronismo vertical
  pm3:
  PROCESS (n_vs)
  BEGIN
    IF (n_vs'EVENT AND n_vs='0') THEN
      WRITE(image,'*');
    END IF;
  END PROCESS;

END;
--------------------------------------------------------------------------------
```

.