Sistemas Informáticos
Curso 2002-2003

Procesado de imágenes digitales sobre un sistema hardware dinámicamente reconfigurable

Isaac Álvarez Gallego
Sergio Bernabé Villalobos
Gerardo Robledillo Gómez
Mª Pilar Rodríguez Crespo

Dirigido por:
Prof. Hortensia Mecha López
Prof. José M. Mendías Cuadros
Dpto. de Arquitectura de Computadores y Automática

Facultad de Informática
Universidad Complutense de Madrid

Índice

1. RESUMEN DEL PROYECTO

El objetivo principal de nuestro proyecto era implementar sobre la FPGA de una placa XSV-800, un circuito de visualización y procesamiento de imágenes. Este procesamiento consistía en aplicar una serie de filtros a las imágenes.

Como complemento al circuito de visualización, fue necesario crear una herramienta software que transformara imágenes de formato .BMP a un formato propio. Estas imágenes en el nuevo formato, se guardan en la RAM mediante otra herramienta software (heredada de un proyecto anterior) de comunicación entre el PC y la XSV-800 usando el puerto paralelo. Luego las imágenes se leerán de la RAM para ser mostradas en la pantalla por medio de una RAMDAC, un dispositivo contenido en la placa.

En cuanto al procesamiento de las imágenes, se carga en la FPGA el filtro deseado, que trata la imagen, y la almacena de nuevo en la RAM para su posterior visualización. Los filtros disponibles son binarización, escala de grises, negativización, reducción del espectro de color a 8, suavizado y cálculos de los bordes una imagen.

Como objetivo secundario también estaba conseguir reconfigurar dinámicamente la placa de una forma total. Para ello, guardamos en la memoria Flash de la placa dos configuraciones, con distintos filtros, y permitimos configurar la FPGA en cualquier momento con cualquiera de éstas, en función de las peticiones del usuario.

En esta documentación se detalla el proceso de desarrollo seguido, las herramientas utilizadas, los conocimientos necesarios para trabajar con imágenes sobre FPGAs, además de incluir y explicar todos los circuitos y código fuente desarrollados por nosotros durante este proyecto.

Project Outline

The main goal of this project is to implement on the FPGA of a XSV-800 board, a visualization and image processing circuit. This processing is based on the application of several image filters.

As a complement to the visualization circuit, it was necessary to create a software tool that transforms .BMP images into an own format. Images in this new format will be stored inside the RAM through another software tool (inherited from a past SI project) that communicates the PC and the XSV-800 by the parallel port. The images stored inside the RAM memory will afterwards be showed in a VGA monitor through a RAMDAC, which is embedded in the board.

For the image processing, the desired filter is downloaded into the FPGA, and then the image is managed and stored again into the RAM memory for its further visualization. The different filters that are available include: binarization, grey scale, negative effect, reduction of the colour spectrum to 8, softening of the image, and border settings.

A secondary goal was making a total reconfiguration of the board dynamically. Therefore, two settings, with two filters, to reconfigure the FPGA are saved inside the Flash RAM, allowing configuring the FPGA with one or the other of the configurations at any point in time, depeding on user's requests.

This document shows a detailed description of the process, all the tools used to achieve the goals, and the required knowledge to manage images on a FPGA, as well as showing and explaining all the different circuits and codes developed during the project execution.

2. OBJETIVO DEL PROYECTO

El enunciado del proyecto original era:

"El objetivo fundamental de este proyecto es implementar sobre una tarjeta de FPGAs dinámicamente reconfigurable una tarjeta de visualización y procesamiento de imágenes. Se implementarán distintos circuitos hardware que realicen diferentes procesos sobre una imagen, de forma que en la tarjeta de la FPGA siempre se mantenga el algoritmo de visualización y puedan cambiarse, a gusto del usuario, los algoritmos de procesamiento. Se utilizará la posibilidad que proporciona el hardware reconfigurable de modificar parte del circuito mientras el resto sigue ejecutando un algoritmo.

La implementación se realizará en una placa de prototipado APX-240 que dispone de una FPGA virtex 300, utilizando software de Xilinx Foundation, Model Technology y Synopsys."

Sobre este esquema inicial se han ido haciendo modificaciones sobre la marcha, la primera de las cuales consiste en el modelo de FPGA utilizado, que finalmente fue una XSV800-240HQ.

3. CONTEXTO

3.1 Hardware

¿Qué es una FPGA?

Una FPGA es uno de los muchos circuitos prefabricados que existen de lógica programable, como también lo son la PLA, PROM, PAL y los CPLD, en los que se pueden implementar distintos circuitos hardware.

Nuestra placa, además del modelo FPGA antes reseñado, tiene entre otros dispositivos un CPLD, que no es más que una PLA a gran escala con lógica adicional.

Centrándonos en lo que concierne a la FPGA, esta es un array de puertas programable que consta de una serie de componentes:

- Un array de celdas regularmente dispuestas sobre el silicio cuya funcionalidad es programable, denominados CLB's
- Una colección de celdas de entrada / salida dispuestas perimetralmente cuyas características también son programables, denominados IOB's
- Y una colección de bloques de interconexión, que bajo programación permiten conectar CLB's e IOB's entre sí, denominados PSM.

Por tanto, los diseños que se implementan sobre la FPGA no se fabrican sino que se realizan programando adecuadamente los CLB's, IOB's y los bloques de interconexión.

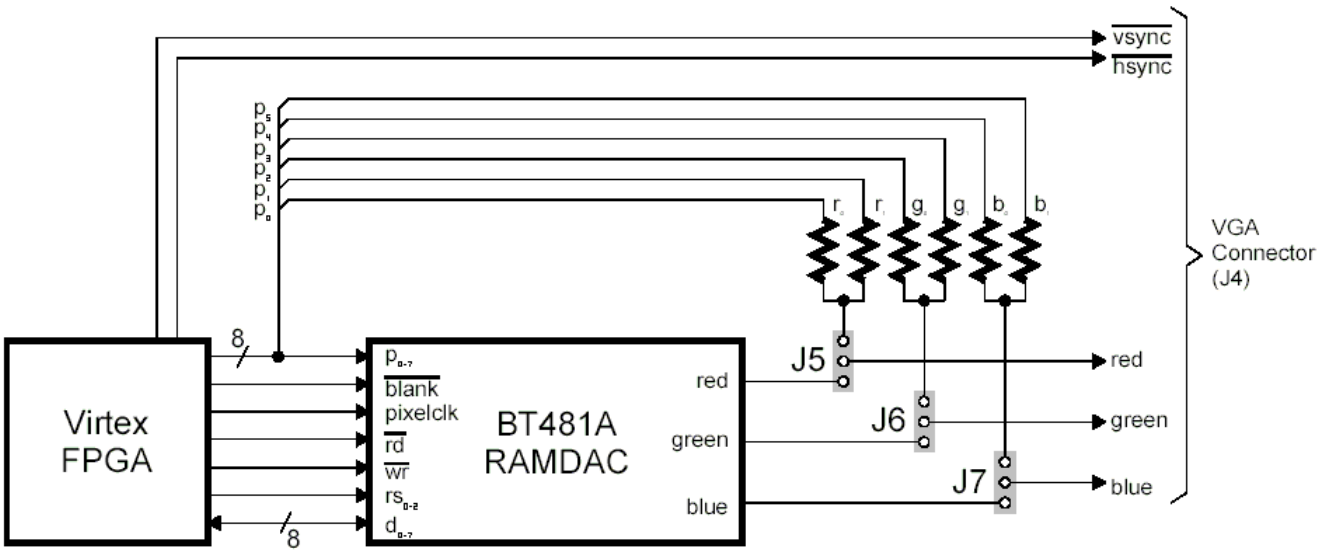


fig. 1 - Modo directo de vídeo de la Ramdac

Modo directo de Vídeo

Las placas XSV pueden generar señales para controlar un monitor VGA [MEND01] bien directamente o bien a través de un chip BT481A RAMDAC incluido en la placa.

En una primera aproximación no se utiliza la RAMDAC, sino que se genera directamente las señales de color en la FPGA, usando 6 bits. Para esto, las líneas de datos de la RAMDAC (p_0-p_5) se entienden como 6 bits de color, siempre que los jumpers J5, J6 y J7 estén en la posición correcta (evitando la RAMDAC y en configuración directa, ver fig. 1). Así, con dos bits para el rojo, dos para el verde y dos para el azul

$p_0 \rightarrow r_0$
$p_1 \rightarrow r_1$
$p_2 \rightarrow g_0$
$p_3 \rightarrow g_1$
$p_4 \rightarrow b_0$
$p_5 \rightarrow b_1$

tendríamos 64 colores distintos.

Por otro lado, las señales de sincronización del monitor, horizontal y vertical (hSync y vSync) deben generarse también en la FPGA. Esto implica decidir a qué frecuencia de reloj se va a trabajar, para ajustar los valores de número de líneas y columnas que se pintan, numero de ciclos hasta los hSync y vSync, ... El significado de estas señales se detalla en el siguiente apartado.

Control un monitor VGA

En este proyecto, las imágenes se muestran en una pantalla de tubo de rayos catódicos (CRT). En estas pantallas, la imagen se forma mediante un haz de electrones que barre la superficie fluorescente de una pantalla siguiendo un determinado patrón de movimiento (ver fig. 2).

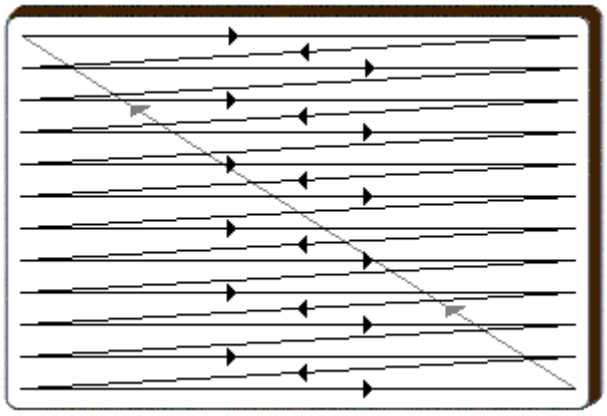


fig. 2 - Patrón de movimiento del haz de electrones CRT

Del control de este barrido se encargan las señales de hSync y vSync, y las señales de intensidad roja, verde y azul del haz de electrones. La señal de hSync marca el comienzo y final de una línea, mientras que por el contrario vSync determina el final de una pantalla completa (fin de la última línea de una pantalla). La intensidad de cada una de las señales de color del haz (RGB) deciden el color del punto (o píxel) que se dibuja en esa zona de la pantalla.

Con esto, se entiende que la sincronización entre la posición del haz en cada momento y los valores de intensidad generados es muy importante. Debe conocerse la velocidad del haz para, en conjunción con las señales hSync y vSync, conocer en cada momento la posición del haz sobre la pantalla y decidir los valores de intensidad a enviar (los del píxel correspondiente).

La RAM

Las placas XSV tienen dos bancos de memoria SRAM formado cada uno por dos módulos de 4 Megabits, organizados en 2^{19} bytes, y accesibles por byte. Ambos bancos funcionan de la misma manera. Disponen de señales oe*, ce* y we* (cuya funcionalidad se explicará al comentar la fig. 5). Dentro de cada uno de los bancos, ambos módulos comparten las líneas de direcciones y de oe*, ce* y we*. Por esto, puede entenderse cada uno de los bancos de memoria como una única SRAM de 8 Megabits, con 2^{19} palabras de 16 bits.

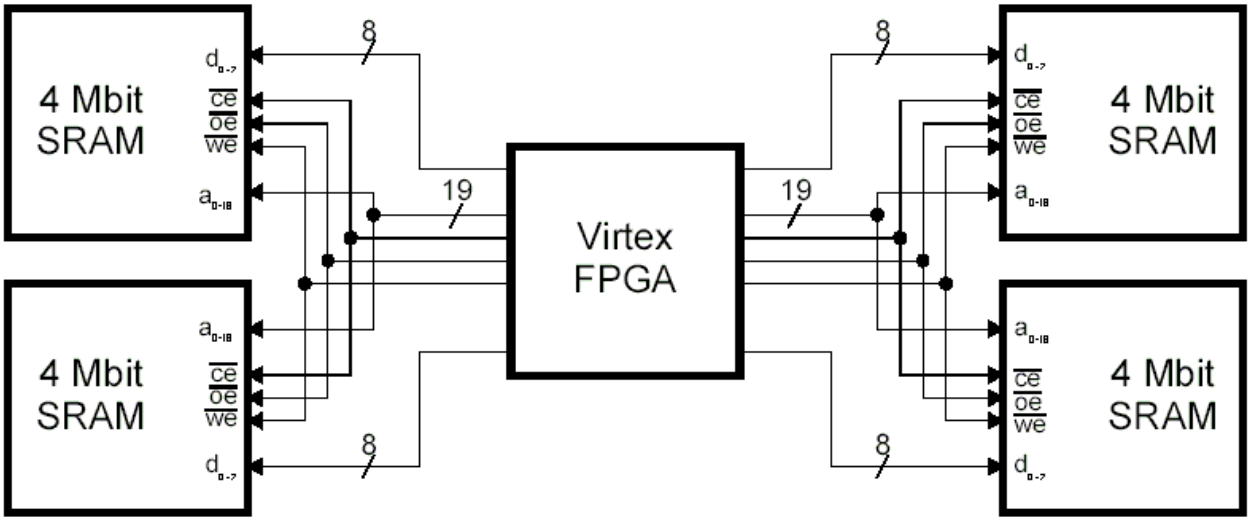


fig. 3 - Configuración de los bancos de memoria RAM.

Por otro lado, las líneas d_0-d_7 del banco derecho de memoria (si entendemos que todo el banco derecho tiene sus 16 bits de datos nombrados d_0-d_{15}, como vienen nombrados en el manual de las placas XSV [XSVB01]), están directamente conectadas a las líneas p_0-p_7 de la RAMDAC.

Es por esta razón (la existencia de dos módulos por banco de memoria) por la que cuando se trabaje con datos de 16 bits, se tendrá que enviar a la RAMDAC en un ciclo el byte menos significativo y en el siguiente el más

significativo, porque sólo el módulo inferior (la parte menos significativa) tiene conexión directa con la RAMDAC.

La RAMDAC

La RAMDAC es un dispositivo hardware de la placa, diseñado para el desarrollo de gráficos de color de alta resolución y que permite mostrar las imágenes con una mayor gama de colores de la que se dispone cuando se hace de forma directa.

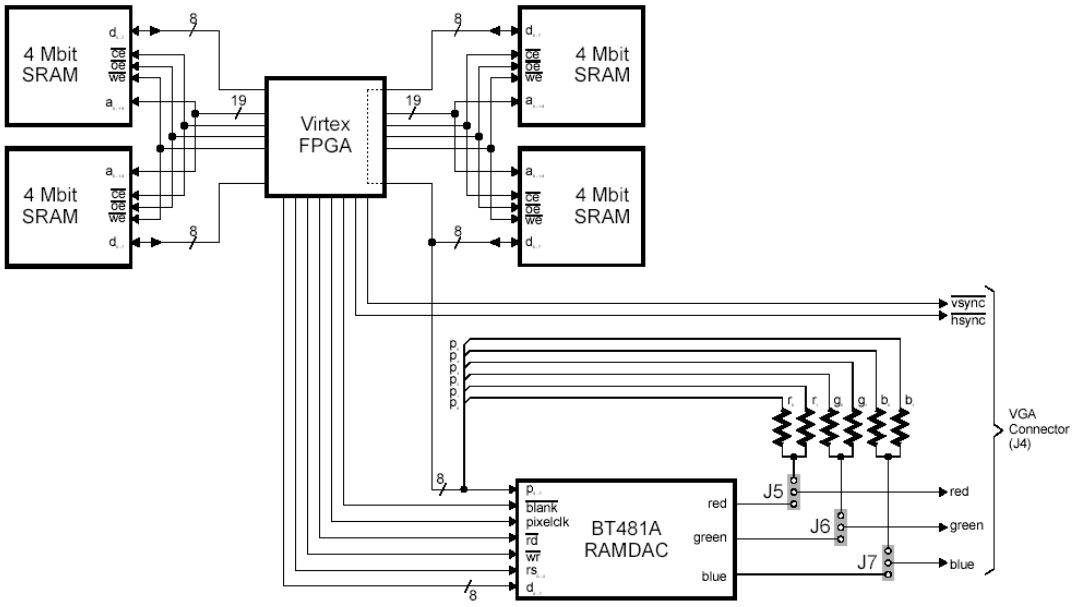


fig. 4 - Conexión entre la FPGA, RAM y RAMDAC

El modelo Bt481A de que disponemos nos permite cuatro modos de color:

1. El formato TARGA, que son 15 bits de la forma 5:5:5 (5 bits para el rojo, 5 para el verde y otros tantos para el azul)

2. El formato XGA, que presenta 16 bits de la forma 5:6:5.

3. El formato RGB/BGR que presenta 24 bits de color de la forma 8:8:8 .

4. Por último suministra un formato de 8 bits de pseudo color.

De todas estas posibles configuraciones, se decide trabajar con el formato TARGA con 15 bits de color, lo que nos da 2^{15} = 32.768 colores distintos, suficientes para tener una buena calidad de imagen, y en el modo single-edge clock que determina que los datos son cargados en el flanco de

subida del reloj, porque si se escoge dual-edge se realiza tanto en el flanco de subida como en el de bajada.

Por otra parte, la RAMDAC posee una serie de registros de configuración que se explican en el siguiente apartado

Configuración de la RAMDAC en formato TARGA y single-edge-clock

Para configurar de este modo la RAMDAC (en TARGA, single-edge), los valores necesarios de los datos de configuración son los de la tabla 3.

A	10100000
RD*	1
$RS_{2..0}$	110
$D_{7..0}$	10100000
BLANK*	0

TABLA 3 - VALORES DE DATOS DE CONFIGURACIÓN

El registro RS[2..0] especifica el tipo de operación de lectura o escritura que se realiza y si es sobre un registro o paleta de colores. En nuestro caso, el valor “110” indica al MPU que debe acceder al registro A.

La entrada de datos D representa el bus de datos para la configuración y contendrá la misma información que se desee escribir en el registro A, es decir D[7..0]=10100000.

El registro A consta de 8 bits y debemos indicarlo con el valor “10100000” que indica lo siguiente:

1. A[7..4]=1010, sirve para seleccionar el modo 5:5:5 single-edge Clock.

2. A[3]=0 que siempre debe contener un cero lógico para asegurar que se opere correctamente.

3. A[2]=0 que indica que la señal RS2 dependerá del valor lógico de RS[2].

4. A[1]=0 que indica que los bits de píxel seguirán el modo RGB.

5. A[0]=0 impide que el registro de selección extendido no pueda ser accedido.

El registro RD* se pone a 0 indicando que se obtendrán datos de los registros RS y D en el flanco de bajada.

El registro BLANK* ignora los valores RGB de los píxeles cuando vale 0 y los pone a negro (0 lógico), por tanto, este registro se inicializa a 0 al

principio, y vale 1 siempre que nos encontremos en la región válida del muestreo.

A continuación se comentan otros aspectos de configuración que pueden resultar interesantes:

- El formato true-color 5:5:5 de carga (formato TARGA), contiene 16 bits organizados de la siguiente forma:
  - El bit B_{15} es ignorado
  - Los bits $B_{14..10}$ pasan por el conversor analógico digital del rojo (red DAC)
  - Los bits $B_{9..5}$ pasan por el conversor de verde (green DAC)
  - Y, por último, los bits $B_{4..0}$ pasan por el conversor de azul (blue DAC).
- El pixelClkOut es el reloj de píxel y coincide con nuestro reloj de sistema (50Mhz aunque necesitamos refresco a 25Mhz)
- El registro SYNC* deshabilita la información de sincronismo con RGB si se activa (con un 0 lógico).
- OL[3..0] indica cúal es la paleta que se usa. Si se pone a 0 se utiliza la paleta por defecto.
- Por último, tenemos el TRUE_COLOR* que deshabilita los modos de color real, necesario en esta parte del proyecto.

Por otra parte, la RAMDAC presenta la entrada de datos P[7..0], que es la entrada de datos, conectada directamente al modulo inferior del banco de memoria, de donde se leen los bits menos significativos, como muestra la fig. 4. Cuando se envíen los más significativos, se tiene que conectar, mediante la FPGA, la parte alta de la RAM con la RAMDAC.

Una vez configurada la RAMDAC, es necesario darle un pulso a WR* durante 50ns para que se quede guardada la configuración introducida. Para esto, se tiene que mantener activo WR* durante un mínimo de 3 ciclos si trabajamos a 50MHz (nosotros lo dejamos activo 4 ciclos para mayor seguridad).

Además de todo esto, hay que deshabilitar el módulo de Ethernet de la placa XSV-800, porque algunas líneas que le unen a la FPGA son compartidas con líneas de configuración de la RAMDAC, en concreto con las líneas $D_{7..0}$ y $RS_{2..0}$. Así, mientras el modulo Ethernet esté activo, estará forzando esas líneas a algún valor, lo que impedirá que se configure correctamente la RAMDAC. Activando la señal TRSTE del Ethernet, este se deshabilita poniendo todas sus líneas a alta impedancia, con lo que se soluciona ese problema.

Reconfiguración

Configuración de la FPGA desde la Flash RAM [DEHD02]

A continuación se explica el funcionamiento del interfaz básico que automáticamente carga el contenido de la FLASH desde la dirección 0 hasta que la FPGA señale el fin de la configuración. Este interfaz es proporcionado por Xess en su página:

- Se selecciona el modo SelectMap para configurar la Virtex desde la Flash RAM. En este modo, la FPGA acepta bytes de configuración en el flanco de subida del reloj de configuración.

- Esto se da siempre que las señales **chip-select** y **write-enable** estén activas.

- Se realiza una división de frecuencia de la señal de reloj. Esto es debido a que la Flash tiene un tiempo de acceso de 85ns, mientras que el oscilador de la placa puede alcanzar una frecuencia de 100Mhz. Por ello, se divide la frecuencia del oscilador entre 16 y se usa un reloj más lento para la configuración de la Virtex.

- Tras el encendido de la placa es necesario esperar un tiempo antes de que ésta esté lista para que la configuración comience. Para ello, se implementa un contador hardware y se deshabilita la configuración hasta que éste haya finalizado.

- A partir de este momento el circuito simplemente incrementa el contador de direcciones para leer el siguiente byte de la Flash RAM y lo envía a la Virtex FPGA.

- Cuando la FPGA indica que la configuración ha finalizado, el CPLD cesa sus operaciones. En este momento se ponen todas las líneas usadas por el CPLD que están compartidas con la FPGA a alta impedancia para que puedan ser utilizadas por el nuevo circuito cargado en la Virtex.

Proceso de arranque de la placa

Tras el encendido, la configuración de la FPGA es automáticamente borrada. Un 0 lógico en la señal **PROG*** limpia la configuración y mantiene a la FPGA en el estado de “memoria de configuración borrada”; estado en el que se mantiene mientras **PROG*** continúe a 0 manteniendo el valor de la señal **INIT*** a 0 para indicar que la memoria está siendo limpiada.

Cuando el valor de **PROG*** cambia, la FPGA continúa manteniendo el valor de **INIT*** a cero hasta que finalice el borrado de toda la memoria. El pulso

mínimo para **PROG*** es de 300ns, no habiendo un valor máximo. El valor de **INIT*** puede ser mantenido a cero desde el exterior de la placa para evitar su configuración.

Una vez que el valor de **INIT*** pasa a ser 1, la configuración puede comenzar. Ya no son necesarias más esperas, pero no es necesario que la configuración comience nada más producirse la transición en el valor de **INIT***.

Durante la configuración se produce una comprobación del valor del CRC (palabra de paridad) del bitstream que se está cargando. Si el valor del CRC no es correcto, la señal **INIT*** pasa a valer 0, indicando así que se ha producido un error en el CRC. La secuencia de comienzo, si se produce esta situación, es abortada y la FPGA no pasa a estado activo.

Para reconfigurar el circuito, el valor de la señal **PROG*** debe ser puesto a 0 lógico para borrar la configuración. Apagando y encendiendo la plca también conseguimos que la FPGA sea borrada para una nueva configuración.

Cuando se ha verificado el valor del CRC, la FPGA entra en una secuencia de encendido, en la que se activa la señal **DONE** para indicar que la configuración se ha realizado correctamente y se activan las entradas/salidas.

3.2 Software

El formato .PACA

Este formato es realmente simple. Los archivos PACA son archivos de texto ASCII (de los generados por cualquier herramienta normal, como el notepad), donde cada entrada está en una línea distinta, acabada en un salto de línea. Hay dos posibles tipos de entrada, de dirección o de datos:

- Una entrada de dirección es un número en hexadecimal terminado en dos puntos.

- Una entrada de datos consiste en un número binario de 16 bits. El numero binario está formado por 1's y 0's ASCII (es decir, en el archivo, abierto con un editor de textos, se verán 1's y 0's y al generarlo se tienen que meter caracteres 1's y 0's, no enteros, ni bits, ni booleanos)

Una entrada de dirección nos pone en esa posición de memoria, y a partir de ahí, las siguientes entradas de datos se guardan de forma consecutiva a partir de la posición de la que se partió. Por ejemplo, con el siguiente archivo PACA:

```
0:
0011001100110011
1100110011001100
1001011010010110
1111111100000000
0000000011111111
```

la palabra 0011001100110011 se almacenará en la posición 0 de memoria, la palabra 1100110011001100 en la 1, y así hasta la 0000000011111111 que se guardará en la dirección 4 de memoria.

El formato .HEX

Los archivos .HEX son archivos de texto en ASCII con registros en formato HEX, uno por fila. El formato HEX es el siguiente:

Marca ":"	Nº de Datos	Desplazamiento	Tipo de registro	Datos	Checksum
1 byte	1 byte	2 bytes	1 byte	N bytes	1 byte

- Marca: Para indicar que lo que viene a continuación es un registro HEX, la marca es el símbolo ":" (dos puntos), y aparecerá por tanto al comienzo de cada fila.
- Nº de Datos: El numero de bytes de información de que constará el registro HEX. En nuestro caso es siempre 02 ya que en cada registro metemos la información de un píxel (16bits)

- Desplazamiento: La dirección en 16 bits donde se comenzara a escribir el registro en la RAM.
- Tipo de registro: Puede tener dos valores fundamentales: “00” en la mayoría de los casos, indicando que el registro es de datos, y “01” para indicar que se trata del ultimo registro del archivo.
- Datos: Pares de dígitos en hexadecimal (tantos como se hayan indicado en el Nº de datos) con la información que se quiera guardar en la RAM.
- Checksum: Un byte, en hexadecimal, con los dos dígitos menos significativos del complemento a dos de la suma de los pares de dígitos de todos los campos anteriores.

El formato .XES

Los archivos .XES son muy parecidos a las .HEX. También son archivos de texto en ASCII con registros, y también uno por fila. El formato XESS se diferencia del HEX en la Marca y en que no tienen Checksum. Hay tres tipos de formato XESS:

- XESS-16, con 16 bits para direccionar la RAM y cuya Marca es “-“. Al tener 16 bits para direccionar, el campo de Desplazamiento tendrá 2 pares de dígitos en hexadecimal (8 bits cada par)
- XESS-24, con 24 bits y Marca “=”.
- XESS-32, con 32bits y Marca”+”.

En el proyecto hemos usado XESS-24 ya que necesitamos direccionar direcciones de 19 bits.

El formato XESS-24 es el siguiente:

Marca “=”	Nº de Datos	Desplazamiento	Tipo de registro	Datos
1 byte	1 byte	3 bytes	1 byte	N bytes

- Marca: En los registros XESS-24 la marca es el símbolo “=” (igual).
- Nº de Datos: El numero de bytes de información de que constará el registro. En nuestro caso se mantiene siempre a 02, ya que en cada registro metemos la información de un píxel (16bits)
- Desplazamiento: La dirección en 24 bits (3 pares de dígitos en hexadecimal) donde se comenzara a escribir el registro en la RAM.
- Tipo de registro: Puede tener dos valores fundamentales: “00” en la mayoría de los casos, indicando que el registro es de datos, y “01” para indicar que se trata del último registro del archivo.
- Datos: Pares de dígitos en hexadecimal (tantos como se hayan indicado en el Nº de datos) con la información que se quiera guardar en la RAM.

4. MÓDULOS INICIALES

4.1 Control de VGA

Varios de los pasos realizados durante el proyecto, se llevaron a cabo en paralelo debido a que presentaban una cierta independencia de diseño, para posteriormente pasar a su debida integración.

Al principio, y como toma de contacto con el VHDL sintetizable, se realizaron un par de pequeños diseños, como un contador, primero normal y luego ascendente descendente y un registro desplazamiento. Estos proyectos también fueron útiles para aprender a usar las distintas herramientas, como Xilinx y las utilidades GXSLoad[1].

Una vez familiarizados con el entorno y el lenguaje, creamos un módulo que, usando las facilidades gráficas de la tarjeta, nos permitía experimentar con ellas.

Cálculo de valores de las constantes VGA

La resolución final de nuestro sistema gráfico dependerá tanto del valor nominal de la pantalla (en nuestro caso 640x480) como del controlador que genera las señales de intensidad. Por tanto, lo primero que tenemos que decidir es la frecuencia de reloj a la que trabajará nuestro controlador. Para ello, es necesario saber que el estándar VGA presenta una frecuencia de reloj de muestreo del monitor de 25.175Mhz, porque la combinación de esta frecuencia con la del reloj de nuestro sistema determinará la resolución gráfica final.

Teniendo en cuenta todo lo dicho, nos decidimos por una frecuencia de 50Mhz que, aunque podría permitirnos teóricamente una resolución de 1256x480, finalmente nos permitirá una resolución de 640x480, debido a que es esta la resolución del estándar VGA.

Los cálculos los hemos realizado con una frecuencia de reloj del controlador de 25Mhz y hemos obtenido un tiempo de ciclo de 1/25=0,04 microsegundos. Los cálculos los realizamos con una frecuencia de 25MHz en vez de 50MHz (que es el valor real del reloj de nuestra sistema, nuestra frecuencia de trabajo) teniendo en cuenta que en un futuro usaremos la RAMDAC, y esta, cuando funciona a 15 bits de color (TARGA 5:5:5), necesita dos ciclos de acceso para cada píxel, uno para enviarle la parte alta del byte, y otro para enviarle la parte baja. Así pues, cuando trabajemos a 50MHz enviaremos píxeles a 25Mhz, porque tardaremos dos ciclos de reloj en enviar un píxel.

[1] ver Apéndices A y B

Una vez que hemos decidido la frecuencia de nuestro controlador, transformamos los valores a ciclos (o columnas) y a líneas. Estos serán los valores que tendrá nuestro sistema grafico.

Datos en MICRO sg.

1 ciclo = 0.04 microsg.

	Tiempo	Tipo	/ 0.04	# ciclos
Barrido completo de una fila	31,77	Exacto	794,25	794
Blanking horizontal	3,77	Exacto	94,25	94
Barrido visible de una fila	25,17	Máximo	629,25	629
Porche delantero	0,94	Mínimo	23,5	24
Porche trasero	1,89	Mínimo	47,25	48

TABLA 1 - CÁLCULO DE VALORES CONSTANTES DE LOS CICLOS O COLUMNAS DE VGA

En la primera columna de la tabla 1 aparecen los tiempos del estándar VGA (que son fijos para cualquier frecuencia de controlador que queramos tener):

- Barrido completo de una fila : indica los microsegundos que transcurren en el muestreo de todos los píxeles de una fila que, dividiéndolo por 0,04 microsegundos que tiene nuestro ciclo, se obtienen 794 ciclos (o columnas) por fila.

- Barrido visible de una fila : No obstante, el valor anterior (barrido completo de una fila) no es el valor real que se mostrará por pantalla, porque el estándar presenta por pantalla durante un máximo de 25.17 microsegundos, lo que nos otorga la posibilidad máxima de 629 columnas de muestreo (se redondea hacia el valor entero más pequeño).

- Blanking horizontal : tenemos 3.77 microsegundos como tiempo de blanking (que resultan ser 94 columnas). Este tiempo es necesario debido a que mientras se realiza la sincronización horizontal (hSync) y el haz baja de línea en diagonal, no se debe pintar la pantalla. Estos 94 ciclos son el tiempo que tarda el haz de electrones en colocarse en el comienzo de la siguiente fila.

- Porche delantero y trasero : no son más que los tiempos de porche delantero y porche trasero del tiempo de sincronización. Son necesarios para evitar que se pinte antes de que el haz se haya colocado y aportan un seguro para una correcta sincronización. Las columnas o ciclos resultantes son 24 y 48 respectivamente.

La suma de los cuatro últimos valores nos da como resultado un valor de 795, que si tenemos en cuenta los redondeos a entero imprescindibles, nos deja el valor 794 ya calculado, correspondiente al número de columnas por fila.

Vamos ahora con las filas:

Datos en MILI sg.

1 línea = 794 * 0.04 = 31.76 microsg.

	tiempo	Tipo	/ 0.03176	# líneas
Barrido completo de la pantalla	16,784	Exacto	528,46	528
Blanking vertical	0,064	Exacto	2,02	2
Barrido visible de la pantalla	15,25	Máximo	480,16	480
Porche delantero	0,45	Mínimo	14,17	14
Porche trasero	1,02	Mínimo	32,12	32

TABLA 2 - CÁLCULO DE VALORES CONSTANTES DE FILAS DE VGA

Módulo de visualización de patrón fijo

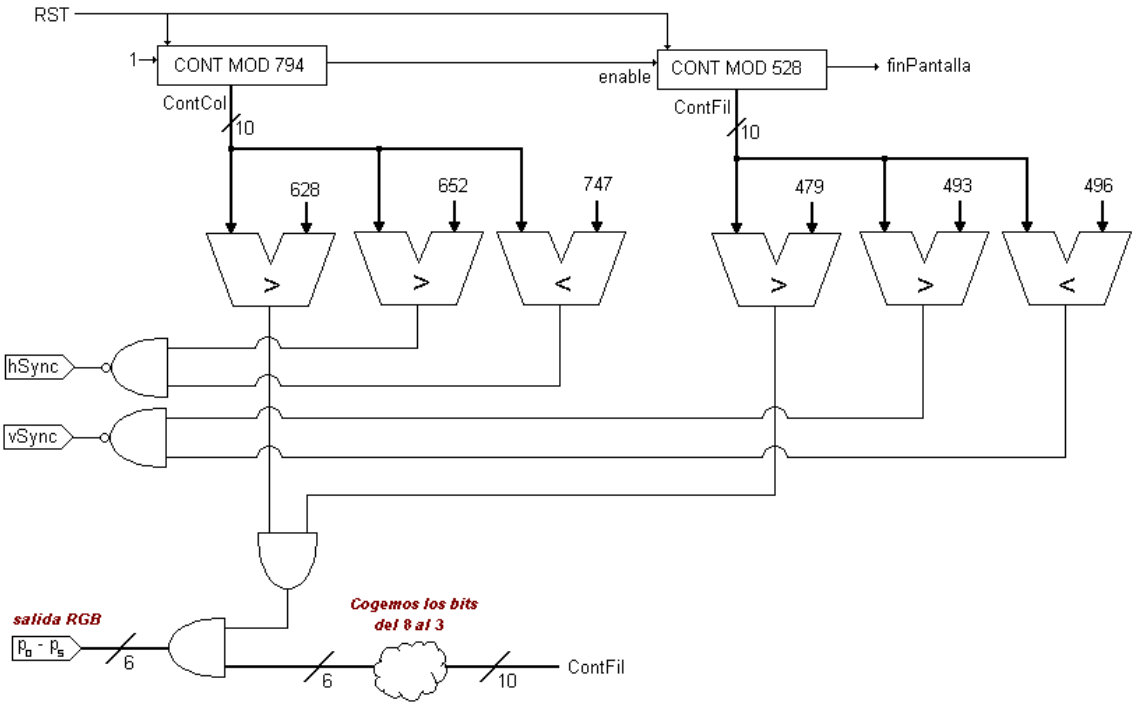


fig. 5 - Módulo de visualización de patrón fijo

Una vez realizados los cálculos necesarios, sólo queda implementar un módulo que, utilizando estos datos, genere directamente las señales de vídeo en la FPGA. Este módulo muestra un patrón fijo, consistente en columnas verticales de ancho fijo y distintos colores.

En la fig. 5 se muestra el diseño del módulo, y su código vhdl puede encontrarse en el Apéndice C, con el nombre contador.vhd.

Como se puede observar, se utiliza dos contadores de 10 bits para la generación de las columnas y filas.

hSync estará activo cuando el número de columnas esté comprendido entre 653 (629 columnas a mostrar + 24 columnas del porche delantero de la

sincronización) y 747 (que es la suma de las columnas a mostrar, el porche delantero más el número de columnas de sincronización horizontal).

vSync esta activo cuando el número de filas esté comprendido entre 494 (480 filas a mostrar + 14 filas del porche delantero de la sincronización) y 496 (que es el valor total máximo de filas).

Además, se cuenta con una señal de enable que aumenta la cuenta de filas cada vez que el contador de columnas termina.

Por otro lado, los valores que se muestran en configuración directa a través de las líneas de datos $p_0..p_5$, se obtienen de los bits 8 a 3 del contador de filas (se cogen estos bits para que el tamaño de las líneas de color mostradas por pantalla tenga un mayor grosor, teniendo en cuenta que se corresponden con los 6 bits de RGB utilizados en este punto). No obstante, sólo se utilizan cuando nos encontramos en la región válida de muestreo (que no es otra que cuando el número de columnas no supera las 628 y cuando las filas no alcanzan las 480).

Posteriormente se hizo una pequeña mejora consistente en mostrar el patrón de líneas vertical u horizontal dependiendo de un switch. Esto no supone más que tomar los bits 8 a 3 del contador de filas o del de columnas, en vez de siempre el de las filas, dependiendo del valor de dicho switch. Este switch viene representado en el código mediante la señal FilCol.

Controlador de monitor con refresco de memoria

Una vez hecho esto, se modificó el módulo de manera significativa. El objetivo de esta modificación era que en el primer barrido de la pantalla se guardara en la memoria el patrón generado a la vez que se mostrara por pantalla, para que en las siguientes pasadas se leyera de la memoria y se enviara a la salida de vídeo para mostrarse. Esto implica tener un minicontrolador de memoria y tener en cuenta si es la primera vez que se genera el patrón o si hay que leerlo de memoria.

En cuanto a leerlo de memoria y mandarlo a la salida de vídeo, esto no es ningún problema, ya que en las placas XSV, la parte baja de los datos del banco derecho de la RAM[2] está directamente conectada a la entrada de la RAMDAC y, por tanto, a la salida de vídeo (si los jumpers están en configuración directa). De este modo, con activar la señal oe* de la RAM y pasarle la dirección correcta, la RAM saca por sus 6 líneas menos significativas los valores de intensidad RGB que se han guardado antes y que son los que interesan. Como hemos dicho antes, debido a que directamente estos 6 bits están conectados a la RAMDAC y mediante los jumpers (si están en modo directo) a la salida de vídeo, no se necesita más lógica para generar la señal de vídeo.

[2] ver sección 3.1, *La Ram*, pg. 8

Con lo único que hay que tener cuidado es con no forzar esas líneas de datos a ningún valor desde la FPGA y dejarlas flotando. Este es el caso, porque se evita la RAMDAC y sólo se usan 6 bits de datos; cuando se empiecen a usar 15 bits de color (en formato TARGA, 5:5:5) a través de la RAMDAC, la cosa se complica.

En cualquier caso, y de momento, las únicas modificaciones que hicieron falta fueron añadir ese minicontrolador y también un multiplexor a la salida de los datos de intensidad de RGB para elegir si se mostraban desde la generación del patrón (en la primera pasada) o si venía de memoria.

Para generar la dirección en la que guardar o de la que leer los datos, se concatenan la parte baja (9 bits) de la señal que lleva la cuenta de las filas con la señal que lleva la cuenta de las columnas. En cualquier caso, esta concatenación sólo se realiza si se está dentro de la zona de visualización (columnas <= 628 y filas <= 479). Si se está en esa zona se genera un cero.

Con esto, el diseño es el mismo que en la fig. 5, pero con los añadidos de la fig. 6. El código de este diseño, con el nombre rayash.vhd se encuentra en el apéndice C.

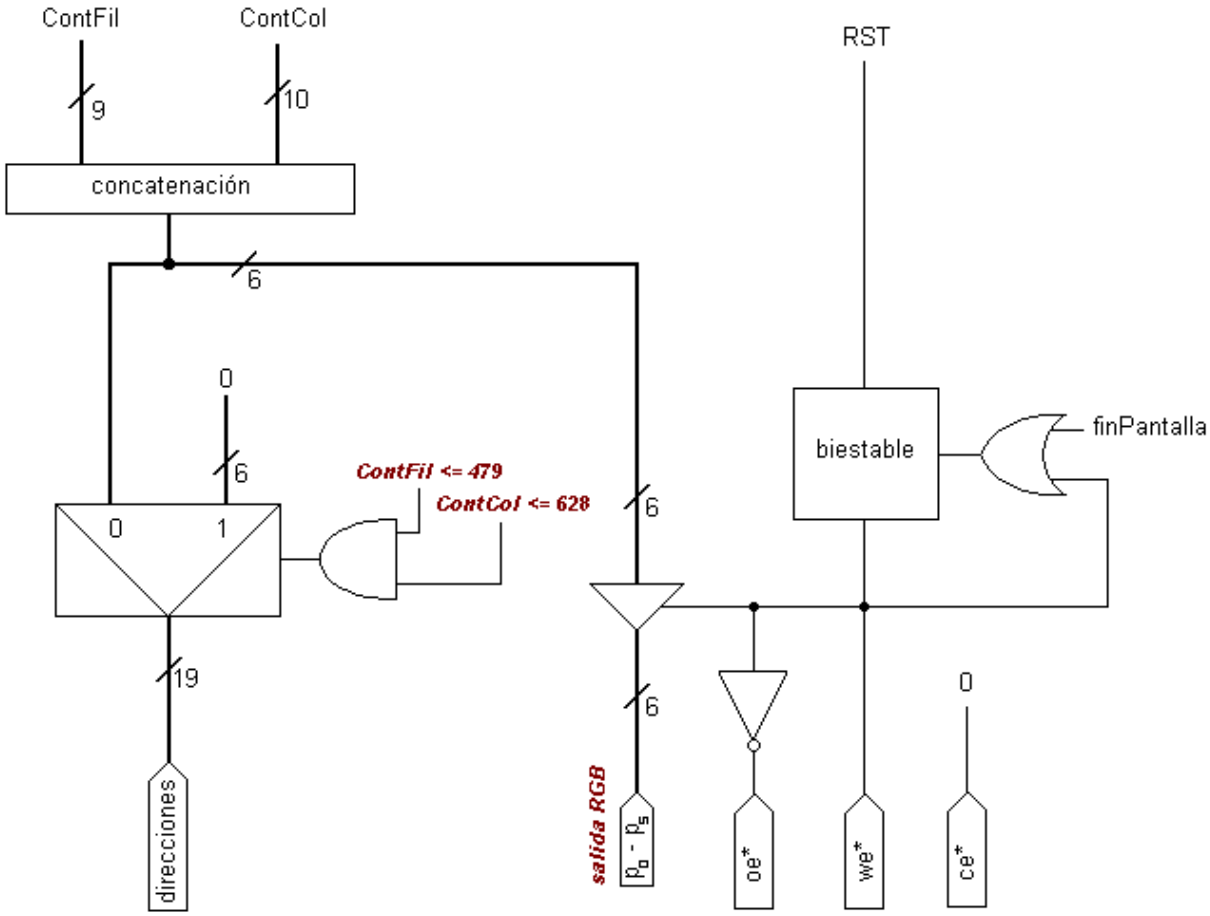


fig. 6 - Controlador de VGA usando memoria

En la figura 6 se pueden observar los siguientes módulos:

- Por un lado tenemos un modulo concatenación, encargado de formar la dirección de la RAM en la que se escribe o lee dependiendo de lo que decida el minicontrolador. Esta dirección de 19 bits esta formada por los 10 bits del contador de columnas (que son los 10 bits menos significativos de la dirección) y los 9 bits menos significativos del contador de filas.
  Como ya se ha dicho, esta concatenación solo se realiza cuando nos hallemos en la zona real de muestreo.

- Por otro lado, los seis bits menos significativos del contador de columnas se cargan en los seis bits menos significativos de la RAM, que son los que están conectados directamente con la RAMDAC, pero sólo la primera vez que se pinta la pantalla, puesto que después se lee de la RAM. De conseguir que en el primer barrido se escriba en la RAM y en los demás se lea de ella, se encargan oe* y we*. Además, mediante un triestado se impide que se envíen valores directamente a la salida una vez que se ha hecho la primera pasada y, excepto en esa primera pasada, deja las líneas que unen directamente la RAM y la RAMDAC flotando para que la salida de vídeo provenga de la memoria.

- El minicontrolador es muy sencillo y consta de un biestable que, inicialmente, almacena un cero. Es en este caso cuando se tiene que escribir en la RAM. Para conseguir que en sucesivas pintadas se lea, se realimenta el biestable con una OR cuyas entradas serán el último valor de dicho biestable y una señal de fin de pantalla que vale 1 cuando ambos contadores (el de columnas y el de filas) lleguen a su fin. A partir de este instante siempre se lee de la RAM, y solamente se vuelve a escribir si se pulsa el reset del sistema, en cuyo caso se vuelve a almacenar un 0 en el biestable.

- Las señales que se utilizan para la sincronización de las lecturas y escrituras son we*, oe* y ce*. Todas estas señales se activan cuando les entra un 0 ya que están implementadas con lógica inversa :

    1. we* recibe el valor actual del biestable. Cuando éste vale 0 la señal de we* (señal de escritura o write enable) se activa y, en consecuencia, se habilita la RAM para su escritura.

    2. En los siguientes pintados de la pantalla, se activa la señal oe* (señal de lectura de la RAM o read enable), ya que recibe el valor negado de lo almacenado en el biestable.

    3. La señal de ce* (o chip enable) simplemente habilita la RAM para su funcionamiento.

Carga de datos en la RAM

A partir de este momento, cuando ya se ha conseguido usar la RAM sin problemas y se lee un patrón previamente guardado, se empieza a investigar la bajada de imágenes a la RAM por el puerto paralelo.

4.2 Carga de imagen en memoria

Para el proyecto se necesita bajar una imagen del ordenador a los bancos de la RAM, por el puerto paralelo. En principio, esto iba a realizarse con la herramienta GXSLoad. Por tanto era necesario crear una herramienta que trasformase un archivo imagen en otro con un formato HEX, XESS o PACA.

Se decidió trabajar con imágenes en formato .BMP, ya que el formato es el más simple (comparándolo con JPEG y GIF, entre otros) Así mismo, se decidió usar archivos en formato HEX para trabajar con la placa, porque en principio parecía el mas simple.

Para crear la aplicación necesaria usamos C++, puesto que ya lo estábamos usando también en Informática Gráfica y podríamos así reutilizar clases, métodos y conocimiento.

De esta manera creamos la aplicación BMP2HEX, que trasforma una imagen en formato BMP a otra en formato HEX para luego bajarla a la placa.

Una vez obtenidos mediante esta aplicación archivos en formato HEX, fuimos incapaces de bajarlos a la placa mediante la herramienta GXSLoad. Pero también nos dimos cuenta de que el formato HEX era insuficiente para lo que nosotros necesitábamos, puesto que sólo te deja manejar direcciones de memoria de hasta 16 bits (debido al campo desplazamiento[3]) y nosotros necesitábamos direcciones de hasta 19 bits, así que decidimos probar con otro formato, el XESS-24

De esta manera creamos la aplicación BMP2XESS. Esta aplicación era una evolución de la anterior, simplemente se le añadieron los métodos necesarios para trabajar con archivos con el formato XESS-24.

Con los archivos en formato XESS-24 tampoco fuimos capaces de bajar imágenes a la placa mediante GXSLoad. Así probamos con el formato paca, creado hace unos años en otro proyecto con la misma placa.

Creamos la aplicación BMP2PACA, evolución también de las anteriores, añadiendo los métodos para trasformar los archivos al formato paca.

[3] Ver sección 3.2, *El formato .HEX,* pg. 14

Para bajar los archivos .paca a la placa no usamos la aplicación GXSLoad sino que usamos el LoadMem, otra aplicación que también fue desarrollada en el proyecto anterior. Esta vez, usando el formato .paca y el LoadMem sí que se consiguió bajar con éxito las imágenes a la memoria.

GXSLoad

Xess (el fabricante de las placas XSV), proporciona una herramienta, el GXSLoad[4], para bajar configuraciones a la FPGA y al CPLD y subir y bajar datos a la RAM y a la FlashRam, en formato HEX, EXO y XESS.

En un principio, parecía que esta herramienta iba a permitir bajar datos a la RAM sin ningún problema, por lo que se desarrolló un conversor software, en C++, para pasar imágenes de formato BMP a formato HEX[5]. Este conversor, *BMP2PACA.exe* se trata con detalle en la sección de Software. No obstante, cabe destacar que este formato no será el que finalmente se utilizará, sino que el formato final pasará a ser .PACA, de ahí que el nombre de la aplicación tenga el sufijo PACA en vez de Hex.

Sin embargo, la herramienta GXSLoad daba problemas. No permitía bajar ni subir más de unos cuantos bytes (entre 0 y 2 Kbytes, aunque no era una cantidad fija, variaba de ejecución en ejecución) antes de dar error. Se pensó en un primer momento que podía ser por problemas de la configuración del puerto paralelo del ordenador, pero se probaron todas las posibles configuraciones y el problema seguía. Además, los problemas sólo aparecían cuando se intentaba subir o bajar a la RAM. No había ningún problema con el puerto si se trataba de bajar datos a la FPGA o la FlashRam. A continuación, se barajó la posibilidad de un fallo en el software de XESS o en el módulo ram800.bit (módulo que se carga en la FPGA para conectar el puerto paralelo con la RAM, pues la RAM no tiene conexión directa con el puerto paralelo). Después de estudiarlos un tiempo, se descartó que el fallo pudiera estar ahí, puesto que son herramientas ampliamente usadas y no se había informado de ningún bug en ellas.

A continuación se descargó e intentó trabajar con distintos programas libres, creados por distintas universidades para bajar y subir datos de la RAM. Estos programas eran el PcToRam de la Universidad de Queensland y el SPP.exe de la universidad de Sevilla. El de la Universidad de Queensland no llegó a funcionar en ningún momento, y el de la Universidad de Sevilla estaba creado para placas XS40 y habría que modificarlo. No se llegó a modificar este software pero sí a intentar reparar el de Queensland. Este intento de reparación significó bastantes problemas (después de una cantidad considerable de trabajo ni siquiera se llegó a poder compilar el código fuente). En ese momento se decidió usar un módulo VHDL y una herramienta Software para bajar imágenes a la RAM de una placa XSV-800 desarrollada en años anteriores en otro proyecto de Sistemas Informáticos [BAPP01].

---

[4] Ver apéndice B

[5] La especificación del formato HEX se encuentra disponible en la página http://www.intel.com/design/zapcode/Intel_HEX_32_Format.doc, o en la sección Software → Archivos .HEX del presente documento

5. VISUALIZACIÓN Y FILTRADO DE IMAGEN EN RAM

5.1 Carga de una imagen en RAM

El software utilizado finalmente para comunicar el PC con la RAM de la placa es la aplicación loadMem.exe, para Windows, que recibe un fichero en formato PACA (formato propietario, creado por los autores del proyecto [BAPP01] y que nosotros hemos heredado) y lo guarda en los bancos de la RAM[6].

Conversión y mostrado

La forma de realizar la conversión, abreviadamente, es la siguiente:

Nuestro programa BMP2PACA transforma una imagen con formato bmp en un pixmap o array de píxeles. Dicho pixmap contiene en cada celda el valor RGB correspondiente a dicho píxel y lo que se hace para la obtención del formato paca deseado es recorrer cada una de las posiciones del pixmap, obteniendo su intensidad de color y traduciendo dicha intensidad a un valor de 15 bits. Cada uno de estos valores contendrá 5 bits para cada una de las intensidades roja, verde y azul del formato RGB de color.

El nuevo fichero .paca esta formado por todos estos valores de píxeles uno a continuación del otro, en serie. Primero el píxel de superior izquierdo, luego el de su derecha y seguimos así por su fila, y luego las siguientes filas de arriba abajo. Este archivo .paca tiene todos los píxeles posibles que pueda albergar la resolución de pantalla utilizada. Por tanto, todos los píxeles que no entren en la región válida (ya comentada antes) de nuestra práctica valdrán 0. Es por este motivo que todos los ficheros .paca tienen el mismo volúmen de datos a pesar de que no se vayan a necesitar muchos de ellos. Este tamaño es del orden de 9.2Mb.

Una vez probado que funcionaba el proyecto original del que se iba a extraer el cargador (y por tanto que funcionaba el módulo cargador mismo), se extrajo de él el módulo de carga de la RAM, que nosotros llamamos cargador, y que es, en realidad, una arquitectura, cuya jerarquía se muestra a en la fig. 8.

[6] en qué banco se guarden los datos, en el izquierdo o en el derecho, dependerá de la asignación de pines que se realice en el archivo ucf del modulo cargador que se baje a la placa. Este módulo cargador (que se carga en la FPGA para comunicar la placa con el PC) se explicará más adelante.

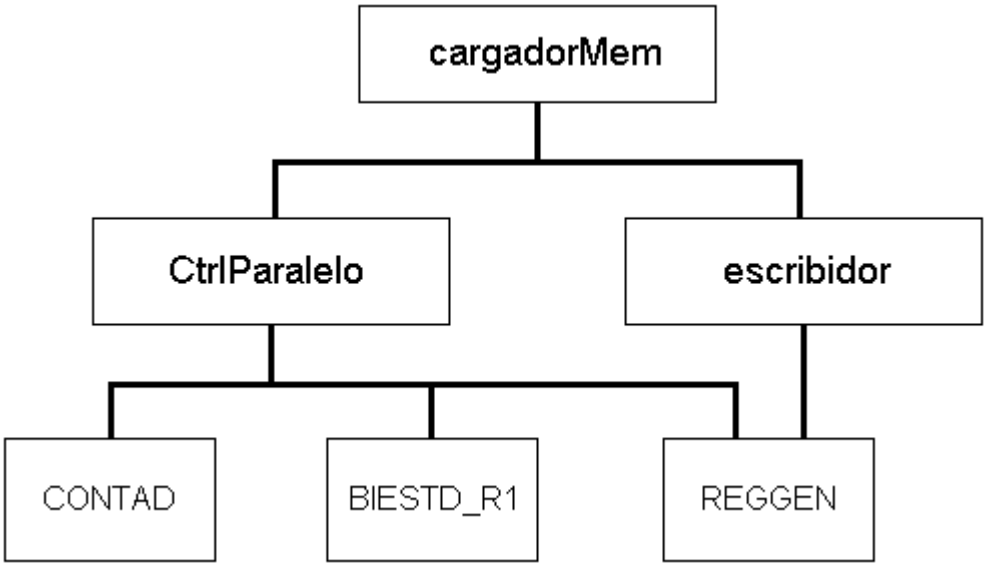


fig. 8 - Arquitectura del módulo Cargador

Dentro de esta arquitectura, CrtlParalelo se encarga de controlar el puerto paralelo, comunicarse con la utilidad LoadMem.exe a través de este puerto (esta aplicación espera confirmaciones desde el puerto paralelo de que los datos han llegado) y leer los datos que llegan a través de él. Luego, escribidor se encarga de ir almacenando estos datos en la memoria RAM.

Su código se encuentra disponible en el apéndice C. A partir de aquí, hay que modificar la herramienta BMP2PACA.exe para que convierta los archivos de imagen del formato BMP (sólo los de 24 bits de color) al nuevo formato PACA. A su vez, se crea un visualizador software de archivos PACA, llamado dibuPaca.exe. Ambos se encuentran comentados en la sección 7.1 y 7.2 de Software.

Una vez que se consiguió bajar correctamente las imágenes a la RAM, era necesario mostrarlas, pero ya no con 6 bits de color, de forma directa, sino pasando por la RAMDAC. Para esto, primero hace falta programar la RAMDAC para trabajar en formato TARGA 5:5:5 single edge. Pasamos a explicar lo que esto significa.

5.2 Visualización de una imagen en la RAM

El siguiente paso es modificar el módulo Controlador de monitor con refresco de memoria[7] para que no guarde nada en la Ram, en ningún momento. Ahora, la información de color es de 16 bits, y la RAMDAC sólo tiene 8 líneas de entrada de datos, así pues, hay que leer de memoria una palabra completa y pasársela a la RAMDAC byte a byte en dos ciclos distintos. Además de esto, se tiene que crear un pequeño subsistema de configuración de la RAMDAC y algo de lógica de control para configurar primero y enviar la imagen a la RAMDAC después. El diseño final del módulo se encuentra en la fig. 9.

[7] sección 4.1, pg. 19

Como puede apreciarse, la figura 9 consta de dos módulos, comentados a continuación:

1. Subsistema de configuración de la RAMDAC:

   - Empezando por la esquina inferior izquierda, vemos que se carga en Pout (entrada de datos de la RAMDAC) el byte más significativo del valor de píxel que, en ese momento, se esté leyendo de memoria. Esto se debe a que, al trabajar con datos de 16 bits, en un ciclo es necesario que leer el byte menos significativo y guardar en un registro el byte más significativo para, en el siguiente ciclo, enviar éste a la RAMDAC. Para realizar esto, se utiliza un triestado que depende de oe*. Cuando esta señal vale 0, el triestado se pone en alta impedancia y no permite la carga de datosIn, sino que se lee el byte menos significativo mediante la conexión directa entre la RAM y la RAMDAC. Cuando, por el contrario, oe* vale 1, se carga en Pout el valor de DatosInGuardados, es decir, el registro donde se guardó la parte más significativa del byte.

   - En los registros RD, Dout y RS se cargan los valores de inicialización[8].

   - Para el cálculo de la señal ini (señal que indica que la configuración de la RAMDAC ya ha sido almacenada) se utiliza un contador de tres bits, es decir, módulo 8, ya que, como ya se comentó, se necesita dar un pulso de tres ciclos a WR*, pero como nosotros lo hacemos durante 4 ciclos, utilizamos el bit más significativo del contador. Cuando éste se pone a 1 por primera vez, ini pasa a valer 1, y en consecuencia también WR, puesto que están conectados. A partir de ese momento, los valores de ini y WR no cambian, debido a la realimentación del biestable con la puerta OR, salvo que pulsemos el reset del sistema, en cuyo caso volvería a iniciarse el proceso.

   - Lo único que queda por ver de este módulo es la señal de blankOut, que valdrá 0 mientras la configuración no se haya realizado (es decir, mientras ini valga 0) y en caso contrario, el valor de BlankOut dependerá de si estamos o no en la región válida de muestro(esto se realiza con la AND de la figura).

2. Subsistema de mostrado del contenido de la memoria:

   - Para empezar, tenemos, como en módulos anteriores, un par de contadores para el recuento de las columnas y las filas, pero en esta ocasión el contador de columnas llega hasta el 1578 (el doble que antes). Este aumento se debe a que necesitamos el bit menos significativo del contador de columnas para cargarlo en oe*. Este bit cambia cada ciclo y precisamente es lo que queremos para oe*

[8] sección 3.1, *Configuración de la RAMDAC en formato TARGA y single-edge-clock*, pg. 10

puesto que debemos poner a alta impedancia las líneas que unen el módulo inferior de la RAM derecha con la RAMDAC[9] para de este modo enviar el primer byte del píxel en un ciclo y conectar el modulo de arriba (y por tanto el segundo byte del píxel) con la RAMDAC en el siguiente ciclo. Así conseguimos mandar el píxel entero en dos ciclos. Además, oe* sólo cogerá este valor cuando ini ya valga 1 (mediante el multiplexor), porque hasta ese momento no necesitamos leer nada, puesto que se esta configurando la RAMDAC.

Además, para los cálculos de hSync, regiones válidas, direcciones y demás, solo cogemos los 10 bits más significativos de dicho contador, y el último lo despreciamos.

- Por otro lado, a ce* le cargamos un 0 para activarlo, para que esté activo siempre, mientras que en we* ponemos un 1 porque nunca vamos a escribir en memoria.

- El cálculo de las señales de sincronización horizontal y vertical (hSync y vSync respectivamente) se realiza como en módulos anteriores, salvo la diferencia de que ahora debemos retardar dichas señales 8 ciclos ya que los píxeles sufren una demora (delay) al atravesar la DAC.

- Por último, está el cálculo de las direcciones de las que se va a leer. Esto se realiza, como siempre, mediante la concatenación del contador de columnas y el de filas.

El código de este módulo (BmpRamda.vhd) se puede encontrar en el apéndice C.

[9] Ver sección *La RAM*

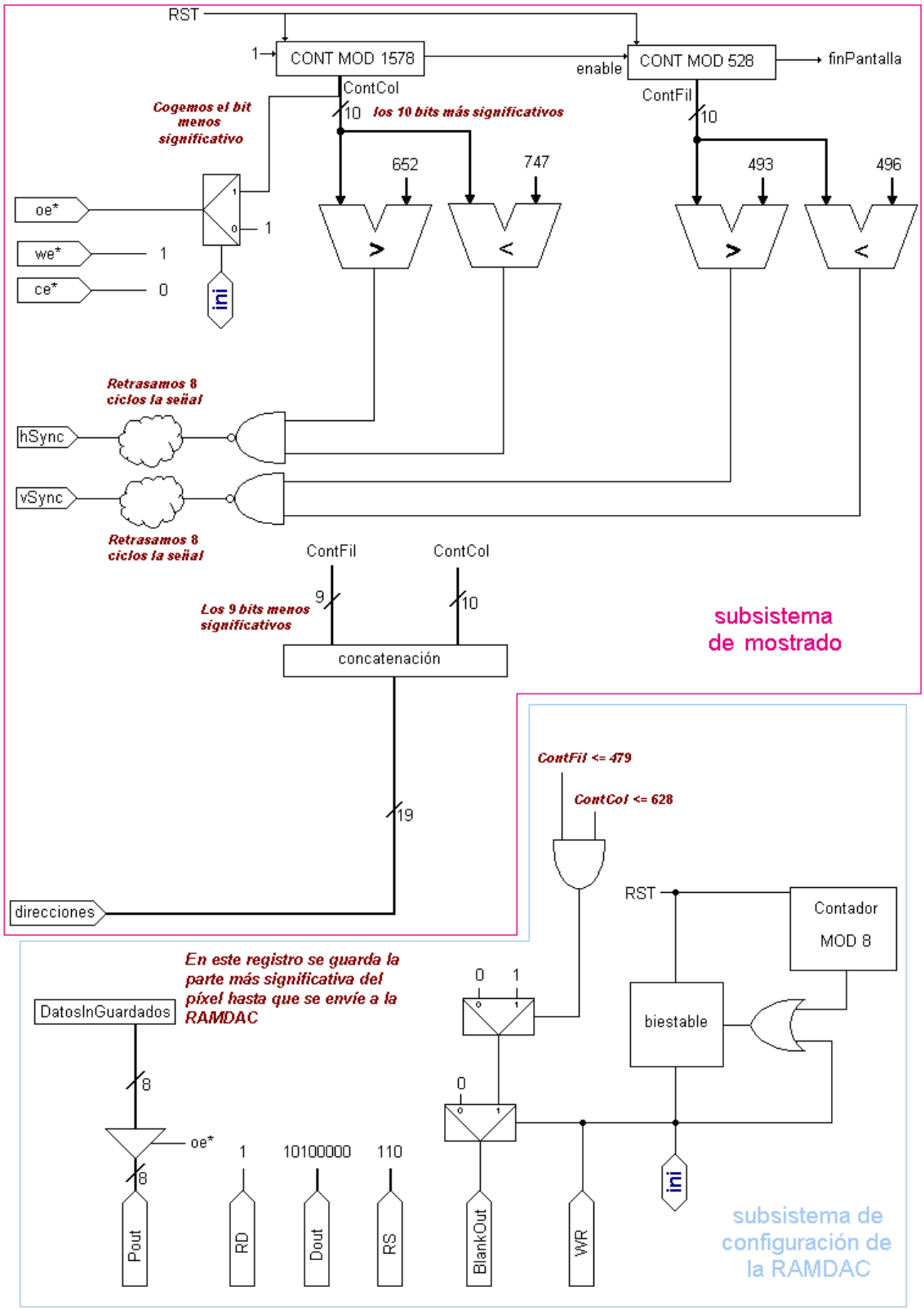


fig. 9 - Subsistema de mostrado una imagen almacenada en memoria usando la RAMDAC

En el diseño y depuración de este módulo, nos encontramos con una serie de problemas, de distinto calibre :

- Problemas de límites de la imagen : La imagen salía torcida y boca abajo por pantalla, como en la fig. 7 .Este era un problema en el software de conversión al formato PACA, que se solucionó cambiando el sentido de un bucle (en vez de recorrer la imagen de abajo a arriba generando en ese sentido el archivo PACA, pasamos a recorrerlo de arriba a abajo), y el valor de una comparación (una vez alcanzado el último píxel de cada línea, se rellenaba con negros hasta alcanzar 1024^{10}, cuando en realidad había que aumentarlo en 1023 posiciones, por que si no se añade un píxel extra)

- Problemas con la configuración de la RAMDAC : Como se ha dicho anteriormente, el módulo Ethernet puede interferir con la configuración de la RAMDAC, cosa que efectivamente pasaba al principio. La imagen salía con colores distintos cada vez que se reiniciaba la placa, y esos colores eran aleatorios. De eso concluimos que la configuración no funcionaba y la RAMDAC estaba funcionando con la tabla de colores. Una vez que nos dimos cuenta del problema con el Ethernet y se inhabilitó el módulo, la configuración, sin ningún cambio más, funcionaba a la perfección y los colores eran ya correctos.

- Para decidir cuándo leíamos un byte del píxel de la RAM (y se mandaba automáticamente la otra parte[11]) y cuándo mandábamos a la RAMDAC ese byte leído inicialmente, usábamos el reloj, dividiendo su frecuencia por la mitad, de modo que en un ciclo leíamos y en otro mandábamos. Pero esto nos causaba problemas de sincronización y la imagen no se mostraba bien. Esto era porque no controlábamos que el primer byte, que iba directamente de la parte baja de la RAM a la RAMDAC, y el byte que enviábamos de la parte alta a continuación fueran del mismo píxel. Lo que efectivamente pasaba es que mandaba la parte alta de un byte y la baja del otro. Esto lo solucionamos haciendo que el que leyera, o mandara, dependiera del último bit de la señal de columnas, (que ahora serían 11 bits, de los cuales 10 servirían para generar la dirección y uno para estar dos ciclos en cada píxel, uno mandando la parte baja y otro mandando la parte alta, que ahora nos asegurábamos de que eran del mismo píxel, porque la dirección (generada por los 10 bits siguientes, que no cambiaban en estos dos ciclos) era la misma durante este proceso).

Una vez solucionados estos problemas, el diseño queda como se ha explicado más arriba (fig. 9), y puede encontrarse su código en el apéndice C.

[10] este valor sale de la forma en que generamos las direcciones en el módulo. Al concatenar las direcciones, cuando pasamos de la última útil de una línea, la siguiente que generamos es aumentando en uno el valor del contador de filas, que al ser el undécimo bit es 2^{10}=1024 posiciones delante.

[11] Ver sección 4.1, *Controlador de monitor con refresco de memoria*, pg. 19

5.3 Filtros de imágenes

Introducción

Las redes combinacionales definidas para los filtros son de tipo Moore, ya que las salidas dependen sólo del estado en el que se encuentre en un momento determinado.

En el desarrollo de cada filtro nos encontramos diferentes problemas que hubo que solucionar sobre la marcha. Estos problemas se comentarán en el apartado correspondiente a cada filtro, ya que, aunque algunos son comunes, muchos de ellos son característicos de cada filtro.

En todos los filtros operamos con el criterio asumido para la RAMDAC, Targa 5:5:5, que marca que cada componente, RGB, de un píxel tendrá 5 bits. Por lo que habrá 2^5 tonos de rojo, de verde y azul, por tanto se conseguirá 32767 colores.

Una utilidad que nos ha servido de mucha ayuda en el desarrollo de estos filtros es la definición de colores personalizados del Paint de Windows. Sobre todo en el caso del filtro Blanco y Negro, en el que pudimos ver el resultado de la formula que aplicaremos, y en el filtro 8 colores.

Todos los filtros siguen un plan de ejecución general, que incluye el uso de dos botones externos, asignados a las señales “rst” y “Accion”:

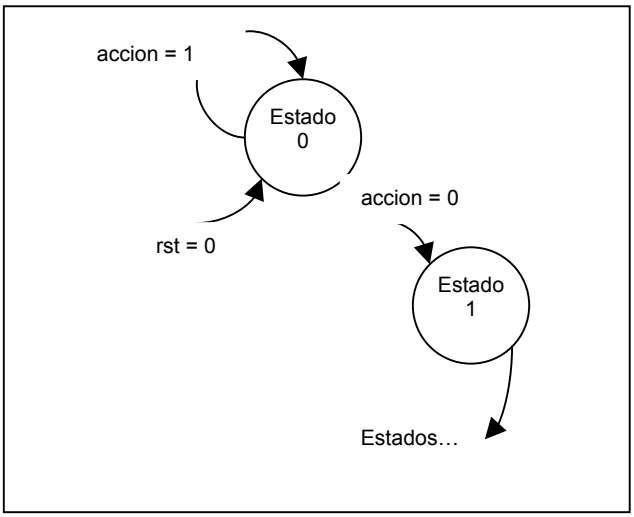


fig. 10 - Diagrama general de estados

Los filtros desarrollados finalmente pueden agruparse en dos grupos, según dos criterios diferentes:

- Según los bancos de memoria utilizados. Por una parte están los filtros que hacen uso de sólo el banco derecho de la memoria RAM, filtros de 8 colores, Blanco y Negro, y Binarizar; Y los filtros que

utilizan tanto el banco derecho como el izquierdo de la memoria RAM, siendo estos los filtros de Negativizado, Suavizado y Bordes.

- Según el modo de operar con los píxeles de la imagen. Los filtros 8 colores, Blanco y Negro, Negativizado y Binarizar operan píxel a píxel, mientras que los filtros de Suavizado y Bordes realizan una convolución de la matriz 3x3 que rodea a cada píxel de la imagen.

Teóricamente, el uso de los dos bancos de la memoria es necesario cuando se va a operar con matrices de píxeles en lugar de píxel a píxel. Para poder almacenar el valor de la convolución de las matrices mientras se opera con el resto de los píxeles de la imagen. Este criterio no se cumple en el filtro de Negativizado que, como veremos, utiliza los dos bancos de memoria aunque opera píxel a píxel.

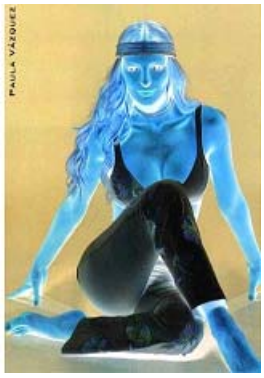
Para estudiar cada filtro por separado utilizamos el criterio de operación con los píxeles.

Filtros Píxel a Píxel:

Filtro de Negativizado:

Funcionamiento:

El objetivo de este filtro es convertir la imagen en su negativo, es decir, en la imagen resultante al sustituir cada píxel por el valor que se obtiene al restar al valor máximo de color que se puede obtener, con ese número de bits, el valor de dicho píxel. Este filtro lee el valor de cada píxel del banco derecho de la memoria RAM y carga este valor negado en el banco izquierdo. Dado que el módulo bmpramda lee siempre del banco derecho de la memoria, el último estado de este filtro se encargará de volcar el contenido del banco izquierdo al derecho. Este filtro utiliza los dos bancos de memoria, aunque teóricamente no sería necesario, porque fue el primer filtro desarrollado con este método. Como se comenta en el apartado de los problemas encontrados, se intentó modificar este filtro para utilizar sólo un banco de memoria, pero no se consiguió.



Para desarrollar este código necesitamos 4 estados:

- Estado 00: Estado de espera. En este estado se entra al pulsar reset. Ambos bancos de memoria están apagados porque no se utilizan. De este estado se pasa al estado 01 cuando se pulsa botón asociado a la señal “accion”.

- Estado 01: Leemos una posición del banco derecho de la memoria, negamos su valor y lo cargamos en la misma dirección de memoria del banco izquierdo. Por este estado pasaremos una vez por posición de memoria, es decir, 524286 veces. Este valor es el resultado del número de columnas (1024) por el número de filas (512).

- Estado 10: Una vez recorrido todo el banco derecho se pasa al estado 10, en el cual se recorre el banco izquierdo de la memoria y se carga cada valor en la misma posición de memoria del banco derecho.

- Estado 11: Estado final. Se apagan las memorias, que ya no se van a utilizar y se activa la señal “yasta”para indicar que el filtro ha terminado.

Diagrama de estados:

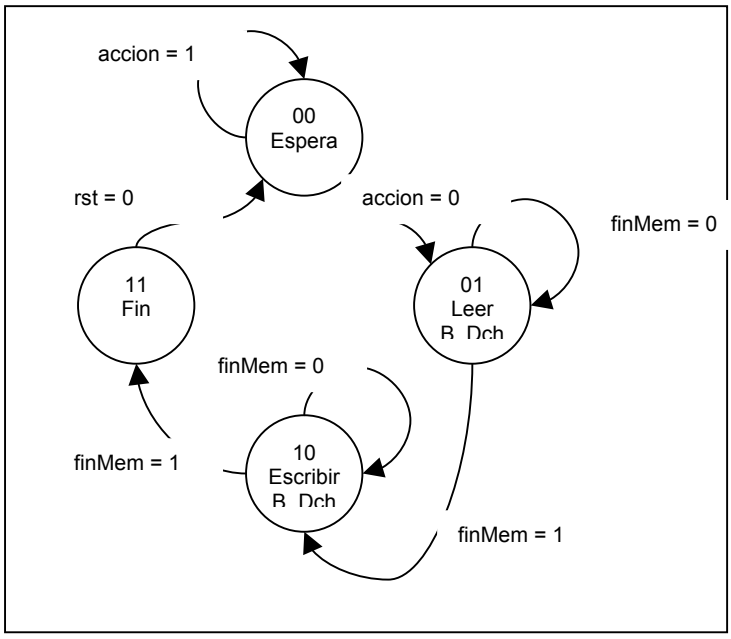


fig. 11 - Diagrama de estados del Filtro Negativizar

Código: Ver anexo C6.4

Problemas encontrados:

El principal problema encontrado en las pruebas de este código fue que la imagen resultante era mostrada unas posiciones más a la derecha que la imagen original. Tras realizar numerosas pruebas determinamos que el problema estaba en el incremento de la dirección de memoria. Aparentemente, incrementábamos este valor más rápidamente y más veces de las necesarias. Para solucionarlo modificamos el código, de forma que el incremento de la dirección de memoria se realizase sólo en el momento del cambio de estado. Posteriormente al desarrollo de este filtro intentamos modificarlo para que utilizase solamente el banco derecho de la memoria, pero no conseguimos que funcionase correctamente con la nueva arquitectura, ya que desplazaba algunas posiciones a la derecha.

Filtro de Blanco y Negro:

Funcionamiento:

El objetivo de este filtro es visualizar la imagen en blanco y negro, con toda su escala de grises. Para que una imagen que use los tres colores RGB se vea solo en escala de grises hay que poner el mismo valor en los tres colores y para ello este filtro se basa en una fórmula de paso Blanco y Negro, facilitado en la asignatura optativa Robótica:

$$Color = 0{,}299 \times Rojo + 0{,}587 \times Verde + 0{,}114 \times Azul$$

Dicha fórmula la redondeamos a los siguientes valores:

$$Color = 0{,}25 * Rojo + 0{,}5 * Verde + 0{,}125 * Azul$$

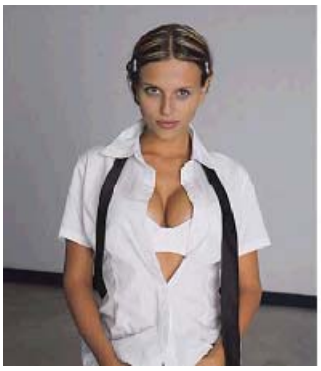
Con este redondeo se pierde información, por lo que la ejecución sucesiva de este filtro sobre la misma imagen hace que se pierdan niveles de gris.

Estas multiplicaciones son equivalentes a dividir por 4, por 2 y por 8, respectivamente. Por tanto, la operación final sería realmente una operación de desplazamiento del número de posiciones necesarias en cada componente:

* 0’5 (x/2) →1000000000→ Desplazar una posición a la derecha el componente R del píxel.

* 0’25 (x/4) →0100000000→ Desplazar dos posiciones a la derecha el componente G del píxel.

* 0’125 (x/8) →0010000000→ Desplazar tres posiciones a la derecha el componente B del píxel.



En esta ocasión utilizamos 5 estados:

- Estado 000: Estado de espera. En este estado se entra al pulsar reset. Ambos bancos de memoria están apagados porque no se utilizan. De este estado se pasa al estado 001 cuando se pulsa el botón asociado a la señal “accion”.

- Estado 001: Se lee la información del banco derecho de la memoria y se carga cada componente del píxel leído en la señal auxiliar correspondiente. Los bits se cargan ya desplazadas las posiciones necesarias. Desde este estado se pasa ineludiblemente al estado 010.

- Estado 010: Estado operación. En este estado se suman los valores de los componentes auxiliares del estado anterior. De esta forma se termina de ejecutar la fórmula anterior y obtenemos los 5 bits con el color definitivo del píxel. Estos 5 bits son guardados en la señal RGBdef. De este estado se pasa al estado 011, para escribir el píxel, ya transformado, en su posición de memoria correspondiente.

- Estado 011: Se carga en el banco derecho de la memoria el valor de la señal RGBdef. En los tres componentes del píxel se repite el mismo valor contenido en RGBdef. Este píxel es cargado en la misma posición de la que fue leído en el estado 001. Este estado y los dos anteriores se ejecutarán tantas veces como posiciones de memoria hay, es decir, 524286

- Estado 100: Estado final. Se desactiva el banco de la memoria utilizada, el derecho. Y se activa la señal “yasta”, para indicar que el filtro ha finalizado.

Diagrama de estados:

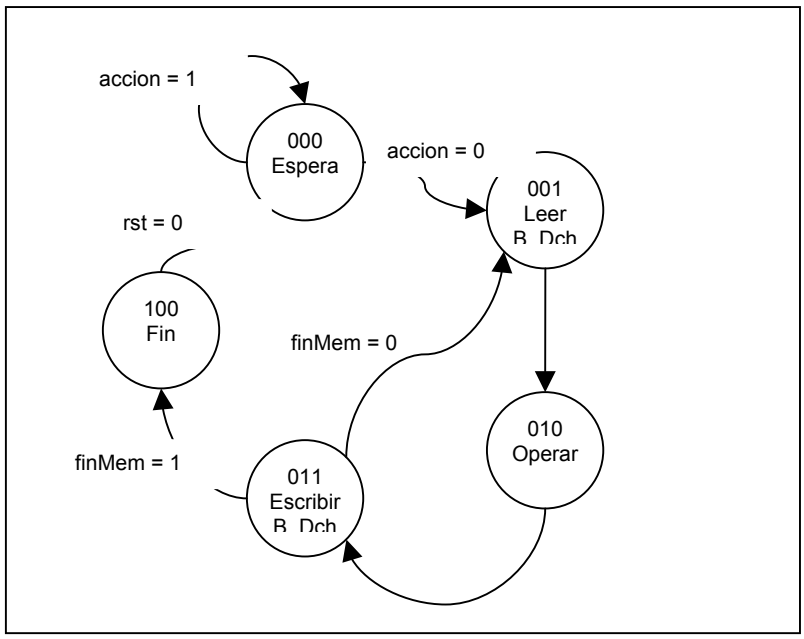


fig. 12 - Diagrama de estados del Filtro Blanco y Negro

Código: Ver anexo C6.3

Problemas encontrados:

Los principales problemas que encontramos fueron derivados de la operación de transformación a Blanco y Negro. En un principio operábamos con los 15 bits en vez de con cada componente por separado. Esto provocaba que la imagen resultante estuviese totalmente distorsionada. Una vez corregido este error, el siguiente problema encontrado fue que el filtro funcionaba de forma aleatoria, es decir, filtraba la imagen unas veces sí y otras no. Para solucionar este problema incluimos el módulo eliminador de rebotes. Para evitar los rebotes de los botones de reset y de "accion". Este módulo lo tomamos de la página de J. M. Mendías[12] y, dado que solucionó el problema en este filtro, lo incluimos por defecto en el resto de ellos.

Filtro de Binarización:

Funcionamiento:

Este filtro parte del filtro de Blanco y Negro para limitar los colores a dos, negro (0) y blanco (32767). Para ello, una vez transformada la imagen a blanco y negro se compara el nivel de gris resultante en cada píxel con un umbral. Si el valor es mayor que dicho umbral se sobrescribe con el valor 32767, blanco, en caso contrario, con el valor 0, negro. Esta comparación se realiza fuera del proceso MaquinaDeEstadosFiltro.

12 http://www.dacya.ucm.es/mendias/143/disenos/debouncer.vhd

Para desarrollar este código utilizamos también cinco estados y sólo un banco de memoria, el derecho:

- Estado 000: Estado de espera. En este estado se entra al pulsar reset. No utilizamos el banco derecho de la memoria todavía, por lo que está apagado. De este estado se pasa al estado 001 cuando se pulsa botón asociado a la señal “accion”.

- Estado 001: Se lee la información del banco derecho de la memoria y se carga cada componente del píxel leído en la señal auxiliar correspondiente. Los bits se cargan ya desplazadas las posiciones necesarias. Desde este estado se pasa ineludiblemente al estado 010.

- Estado 010: Estado operación. En este estado se suman los valores de los componentes auxiliares del estado anterior. De esta forma se completa la transformación de la imagen a Blanco y negro, como parte del proceso de binarización. De este estado se pasa al número 011.

- Estado 011: Se carga en el banco derecho de la memoria el valor de la señal RGBdef. Dado que la comparación con el umbral, y la sustitución de valor correspondiente, se realiza fuera del proceso, en este momento esta señal ya tendrá el valor adecuado. En los tres componentes del píxel se repite el mismo valor contenido en RGBdef. Este píxel es cargado en la misma posición de la que fue leído en el estado 001. Este estado y los dos anteriores se ejecutarán tantas veces como posiciones de memoria hay, es decir, 524286. De este estado pasamos al estado 100, final.

- Estado 100: Estado final. Se desactiva el banco de la memoria utilizada, el derecho. Y se activa la señal “yasta”, para indicar que el filtro ha finalizado.

Diagrama de estados:

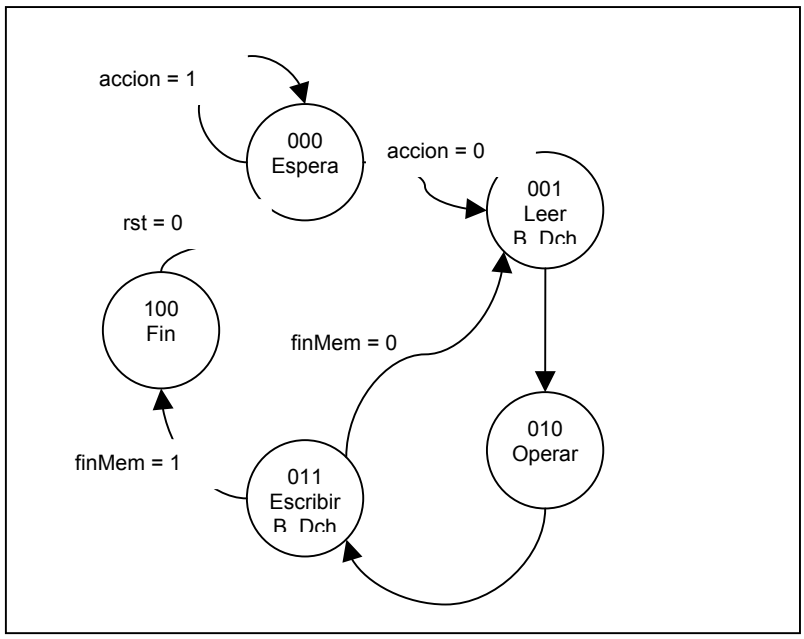


fig. 13 - Diagrama de estados del Filtro Binarizar

Código: Ver anexo C6.2

Problemas encontrados:

Una vez logrado el filtro de Blanco y Negro el desarrollo de este filtro de Binarización fue relativamente sencillo. El único problema encontrado fue la definición del umbral, pero se solucionó fácilmente mediante prueba y error. El umbral seleccionado finalmente fue 16, la mitad de los niveles de gris que puede alcanzar cada componente del píxel. De cualquier manera, es posible que, dependiendo del histograma de cada imagen, haya que cambiar este umbral. Si esto fuese necesario, habría que modificar dicho umbral en el código y volver a sintetizar e implementar el mismo, para generar una nueva versión del fichero .bit.

Filtro de 8 Colores:

Funcionamiento:

Este filtro limita a 8, mediante la utilización de un umbral, los colores de la imagen original. Para ello se realiza una comparación del valor de cada uno de los componentes de los píxeles de la imagen con un umbral, si el valor es mayor o igual, se fija el componente a 1, en caso contrario se fija a 0. Esta operación es equivalente a la realizada en el filtro de binarización, sólo que en este caso se realiza componente a componente.

Este filtro trabaja sólo con un banco de la memoria, el derecho, y con cinco estados:

- Estado 000: Estado de espera. En este estado se entra al pulsar reset. No utilizamos el banco derecho de la memoria todavía, por lo que está apagado. De este estado se pasa al estado 001 cuando se pulsa botón asociado a la señal “accion”.

- Estado 001: Leemos los bits de una posición de memoria y cargamos los correspondientes a cada componente (RGB) en su variable. Desde este estado se pasa ineludiblemente al estado 010.

- Estado 010: Estado Comparación. En este estado se realiza la comparación de cada uno de los componentes del píxel leído y se le asigna el valor correspondiente según el resultado. De este estado se pasa al número 011.

- Estado 011: Se carga en la misma posición del banco derecho de la memoria el valor del píxel transformado. Se escribe bit a bit para ir cargando el valor de cada variable auxiliar, una por componente. Este píxel es cargado en la misma posición de la que fue leído en el estado 001. Este estado y los dos anteriores se ejecutarán tantas veces como posiciones de memoria hay, es decir, 524286

- Estado 100: Estado final. Se desactiva el banco de la memoria utilizada, el derecho. Y se activa la señal “yasta”, para indicar que el filtro ha finalizado.

Diagrama de estados:

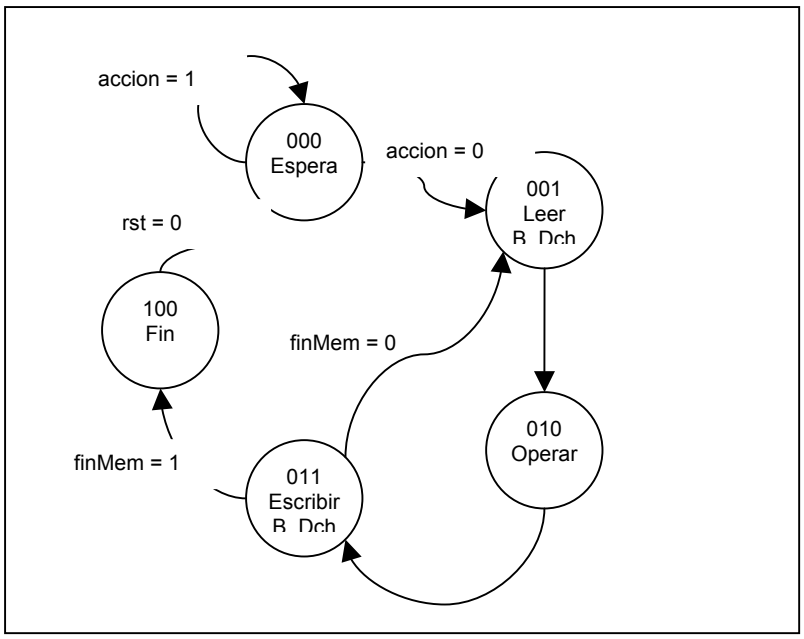


fig. 14 - Diagrama de estados del Filtro 8 Colores

Código: Ver anexo C6.1

Problemas encontrados:

Dado que este filtro tiene una arquitectura muy similar a los anteriores, y que las operaciones a realizar con los píxeles leídos se limitaban a una comparación y sustitución, su desarrollo fue muy sencillo. En las primeras pruebas observamos que algunos píxeles eran desplazados a la derecha. Para solucionar el problema incluimos la variable dirAux, para guardar el valor de la dirección de memoria de la que se lee en el estado 001 y en la que se debe escribir en el estado 011.

Filtros de Convolución de matrices:

En la teoría de convolución de matrices se trata de aplicar un filtro a una imagen incluyendo en cada operación los ocho píxeles que rodean cada píxel de la imagen. Es decir, se aplica la mascara seleccionada sobre una matriz 3x3.

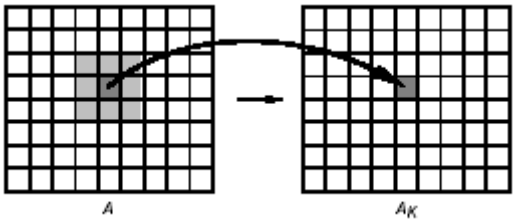


fig. 14

En los dos filtros que hemos desarrollado con esta técnica despreciamos los bordes, es decir, los píxeles de los bordes permanecerán iguales o se

“filtraran” con información no útil situada en la memoria fuera de los límites de la imagen. Este problema sería fácilmente evitable conociendo las dimensiones reales de cada imagen, pero no disponemos de este dato en el momento del desarrollo del código, ya que la información sobre el formato de la imagen no se guarda en memoria, sólo la imagen.

Filtro de Suavizado:

Funcionamiento:

En el caso del filtro de suavizado, la mascara elegida es:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} / 9$$

Esta máscara es un filtro de paso-bajo, que elimina de la imagen original las distorsiones provocadas por los ruidos Gaussiano y Sistemático.

El ruido Gaussiano añade o resta valores aleatoriamente al valor real de cada píxel de la imagen, mientras que el ruido Sistemático aparece como un patrón regular que no forma parte de la imagen actual. Este último tipo de ruido puede ser originado por ejemplo, por la cámara, la digitalizadora, la iluminación, etc.

Una vez extraídos de la memoria los nueve píxeles que conforman la matriz, la operación que habría que realizar con ellos, convolución, sería la siguiente:

$$a_{22} = (a_{11} + a_{12} + a_{13} + a_{21} + a_{22} + a_{23} + a_{31} + a_{32} + a_{33}) / 9$$

Siendo A la matriz de los píxeles de la imagen:

$$\begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{23} & a_{33} \end{bmatrix}$$

El resultado de esta operación será el nuevo valor que, en la imagen transformada, tendrá el píxel del centro, a_{22}.

Realmente, en vez de dividir por nueve, dividimos por 8, para poder simplificar la división a desplazar los bits tres posiciones a la derecha. Esto implica un leve aumento en el valor de los píxeles resultantes, pero la diferencia se puede despreciar.

En este filtro, como en todos los anteriores, las operaciones de transformación hay que realizarlas en cada componente (RGB) de cada píxel

por separado. Esto nos lleva a la utilización de numerosas señales auxiliares en la arquitectura del filtro.

Este filtro necesita utilizar los dos bancos de la memoria, el derecho y el izquierdo, para poder ir almacenando el resultado de la operación de convolución sobre cada píxel sin sobrescribir ningún valor original, hasta haber recorrido toda la memoria.

Para el cálculo de las direcciones, se utilizará la siguiente matriz, donde col tiene el valor 1024:

$$\begin{bmatrix} (dir-col-1) & (dir-col) & (dir-col+1) \\ (dir-1) & dir & (dir+1) \\ (dir+col-1) & (dir+col) & (dir+col+1) \end{bmatrix}$$

Para realizar todas las operaciones necesitamos 15 estados:

- Estado 0000: Estado de espera. En este estado se entra al pulsar reset. Ambos bancos de memoria están apagados. De este estado se pasa al estado 0001 cuando se pulsa botón asociado a la señal “accion”.

- Estado 0001: De este estado al 0110, son los que llamamos estados de inicialización, ya que los utilizamos para cargar la matriz con sus valores iniciales. En este estado leemos del banco derecho de la memoria, el píxel correspondiente a la posición 11 de la matriz. En este punto lo cargamos en la posición 12 porque en estados posteriores se desplaza una posición a la izquierda. La dirección que se lee de la memoria es la 0. De este estado se pasa ineludiblemente al estado 0010.

- Estado 0010: Leemos de la dirección 1 de la memoria el píxel correspondiente a la posición 12 de la matriz. En este estado lo colocamos en la posición 13 porque, al igual que con el píxel anterior, posteriormente se desplaza. De este estado pasamos al 0011.

- Estado 0011: Leemos de la posición 1024 de la memoria el píxel de la posición 21 de la matriz. Como en casos anteriores, lo colocamos en la posición 22 para desplazarlo después. Leemos de la posición 1024 porque tenemos 1024 columnas en la memoria (0-1023). De este estado pasamos al número 0100.

- Estado 0100: Continuamos con la inicialización, en este caso leemos el píxel de la posición 22 de la matriz, dirección de memoria 1025. Lo colocamos en la posición 23 y pasamos al siguiente estado de inicialización 0101.

- Estado 0101: Inicialización de la posición 31 de la matriz. Leemos el píxel de la dirección 2048 (número de columnas x 2) de memoria y lo colocamos en la posición 32 de la matriz.

- Estado 0110: Este es el último estado de inicialización. Se corresponde con el píxel 32 de la matriz, leído de la posición 2049 de la memoria y situado en la posición 33 en este estado. De aquí pasamos al primer estado de desplazamiento y actualización.

- Estado 0111: En este estado y los dos siguientes se desplazan y actualizan los valores de la matriz A. De forma que en dicha matriz estén siempre contenidos los 8 píxeles que rodean al píxel de la posición de memoria que estemos leyendo. En este momento desplazamos una posición a la izquierda todos los valores de la primera fila de la matriz y cargamos el valor de la posición 13. Este valor se corresponde al píxel situado en la posición de memoria "dir" – 1023. Es decir, la dirección de memoria del píxel que queremos tratar menos el número de columnas menos 1. La primera vez que entramos en este estado fijamos el valor de "dir" a 1025, que se corresponde con la segunda columna de la segunda fila de la memoria, para comenzar las operaciones con la primera matriz, situada arriba y a la izquierda de la imagen. De este modo no realizamos la convolución de los píxeles de la primera fila y primera columna de la memoria, que operarían con basura. De este estado pasamos al número 1000.

- Estado 1000: Desplazamos todos los valores de la fila 2 de la matriz y leemos de memoria el píxel de la dirección "dir" +1, para situarlo en la posición 23. Pasamos al último estado de desplazamiento y actualización, el 1001.

- Estado 1001: Desplazamos todos los valores de la fila 3 de la matriz y leemos de memoria el píxel de la dirección "dir" + 1025, para situarlo en la posición 33. La posición de memoria leída se corresponde con la dirección del píxel que estamos tratando más el número de columnas de la memoria más 1.

- Estado 1010: Estado de desplazamiento de componentes. En este estado dividimos por 8, desplazamos tres posiciones a la derecha,

cada componente, RGB, de cada una de las nueve posiciones de la matriz. El tener que desplazar cada componente de cada píxel por separado hace que tengamos que recurrir a numerosas señales auxiliares.

- Estado 1011: Primer estado de operación. En este estado sumamos por separado los componentes de los píxeles de cada fila de la matriz. De forma que nos quedan tres señales con los valores de R, G y B de cada fila. En el apartado de problemas encontrados comentaremos el por que estas sumas se realizan por separado.

- Estado 1100: Segundo estado de operación. En este estado sumamos los tres componentes de cada fila para dejar una sola señal por componente del píxel resultante. Estas tres señales se asignarán directamente a la señal de datos del banco izquierdo de la memoria.
- Estado 1101: Escritura en el banco izquierdo. En este estado cargamos bit a bit cada componente del píxel resultante. La carga se realiza de este modo para facilitar su comprensión, dado el número de señales auxiliares utilizadas durante las operaciones. Este píxel es cargado en la posición de memoria "dir", que se corresponde con la dirección de memoria del banco derecho del que leímos el píxel cuya convolución estamos calculando. El banco derecho de la memoria está apagado, no lo utilizamos. De este estado pasaremos siempre al estado de desplazamiento 1 (0111) hasta que hayamos recorrido todo el banco derecho de la memoria y realizado, por tanto, la convolución de todos los píxeles. Una vez hecho esto y cargados los píxeles resultante en el banco izquierdo procedemos a volcar el contenido de este banco al derecho, de donde se lee para mostrar en pantalla.

- Estado 1110: Volcamos el contenido del banco izquierdo de memoria en el derecho. Una vez recorrido entero dicho banco pasaremos al estado final del filtro.

- Estado 1111: Estado final. Se desactivan ambos bancos de memoria. Y se activa la señal “yasta”, para indicar que el filtro ha finalizado.

Diagrama de estados:

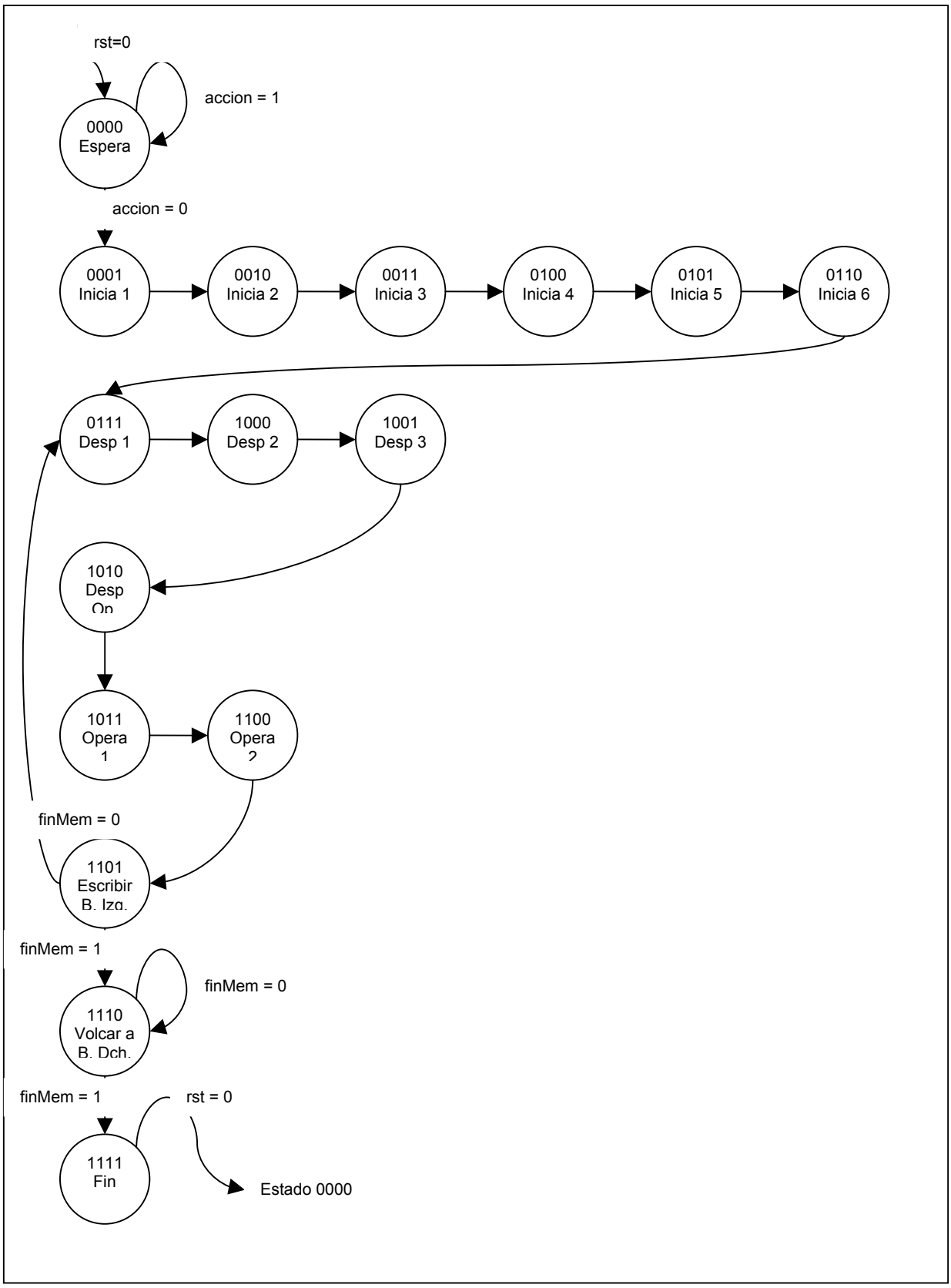


fig. 15 - Diagrama de estados del Filtro de Suavizado

Código: Ver anexo C6.5

Problemas encontrados:

El primer problema al que nos encontramos fue decidir qué método seguir para recorrer todas las posiciones de memoria mientras creábamos las matrices correspondientes a la convolución de cada píxel. Una vez decidido y estudiado dicho método nos encontramos con el problema del desbordamiento en las sumas. Es decir, el valor resultante excedía los bits definidos para la señal que recogía el resultado, lo que no permitía ni compilar el código. Esto nos obligó a separar en varias operaciones y estados las sumas de los componentes ya desplazados. No modificamos la definición de las señales porque estas son utilizadas para cargar datos en memoria. Cuando logramos sintetizar e implementar correctamente el código vimos que la imagen resultante no era la esperada en absoluto. De hecho estaba completamente deformada. En este punto nos dimos cuenta que habíamos realizado mal las operaciones desde el principio, ya que no habíamos hecho la convolución con cada componente de los píxeles sino con los píxeles completos. Después de redefinir de nuevo las operaciones y añadir las señales necesarias nos encontramos que las sumas eran de nuevo demasiado grandes y, aunque el código compilaba sin error, al intentar sintetizarlo llegaba a bloquear el sistema completo. Para solucionar este último problema tuvimos que añadir, nuevamente, más señales auxiliares.

Filtro de detección de bordes:

Funcionamiento:

Los bordes en las imágenes digitales se caracterizan porque delimitan zonas o regiones que difieren significativamente en los respectivos valores de sus niveles de intensidad del gris. Los detectores de bordes, por consiguiente, buscan píxeles donde se produce un cambio en los niveles de intensidad de los colores de la imagen. Para ello hay trabajar sobre una imagen en Blanco y negro, por lo que este filtro implica aplicar primero el filtro de Blanco y Negro a la imagen original. En este filtro nosotros utilizamos el detector de bordes de la Laplaciana, con la siguiente matriz:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Este detector de bordes lo hemos obtenido también de otra asignatura, Robótica.

Como en el filtro anterior, recorreremos la memoria para formar la matriz de convolución de cada píxel y, siendo A tal matriz, operaremos de la siguiente manera:

$$a_{22} = 8 \times a_{22} - a_{11} - a_{12} - a_{13} - a_{21} - a_{23} - a_{31} - a_{32} - a_{33}$$

Como se verá en la explicación de los estados correspondientes, para llevar a cabo estas operaciones se realizarán primero la suma de los componentes y, después, una resta final. Este filtro tiene, en general, el mismo desarrollo que el anterior, el filtro de suavizado. Con la diferencia fundamental de la matriz de filtrado que se aplica.

Aquí utilizamos también 15 estados, de los cuales, los nueve primeros son comunes con el filtro anterior. Así mismo, utilizaremos también el banco izquierdo de la memoria para almacenar los píxeles resultantes después de cada convolución.

- Estado del 0000 al estado 1001: Omitimos la explicación de estos estados dado que, como se ha comentado antes, son comunes con el filtro anterior.

- Estado 1010: Desplazamos tres posiciones a la derecha, para multiplicar por ocho, los tres componentes del píxel central de la matriz. Este píxel se corresponde con la posición de memoria “dir”, es aquel del que estamos haciendo el filtrado.

- Estado 1011: Sumamos por separado los tres componentes de todos los píxeles de la matriz, para restar luego el resultado al componente correspondiente del píxel central, ya multiplicado.

- Estado 1100: Restamos a cada componente del píxel central, ya multiplicado, los resultados de sumar los componentes del resto de los píxeles.

- Estado 1101 al estado 1111: Comunes al filtro anterior.

Diagrama de estados:

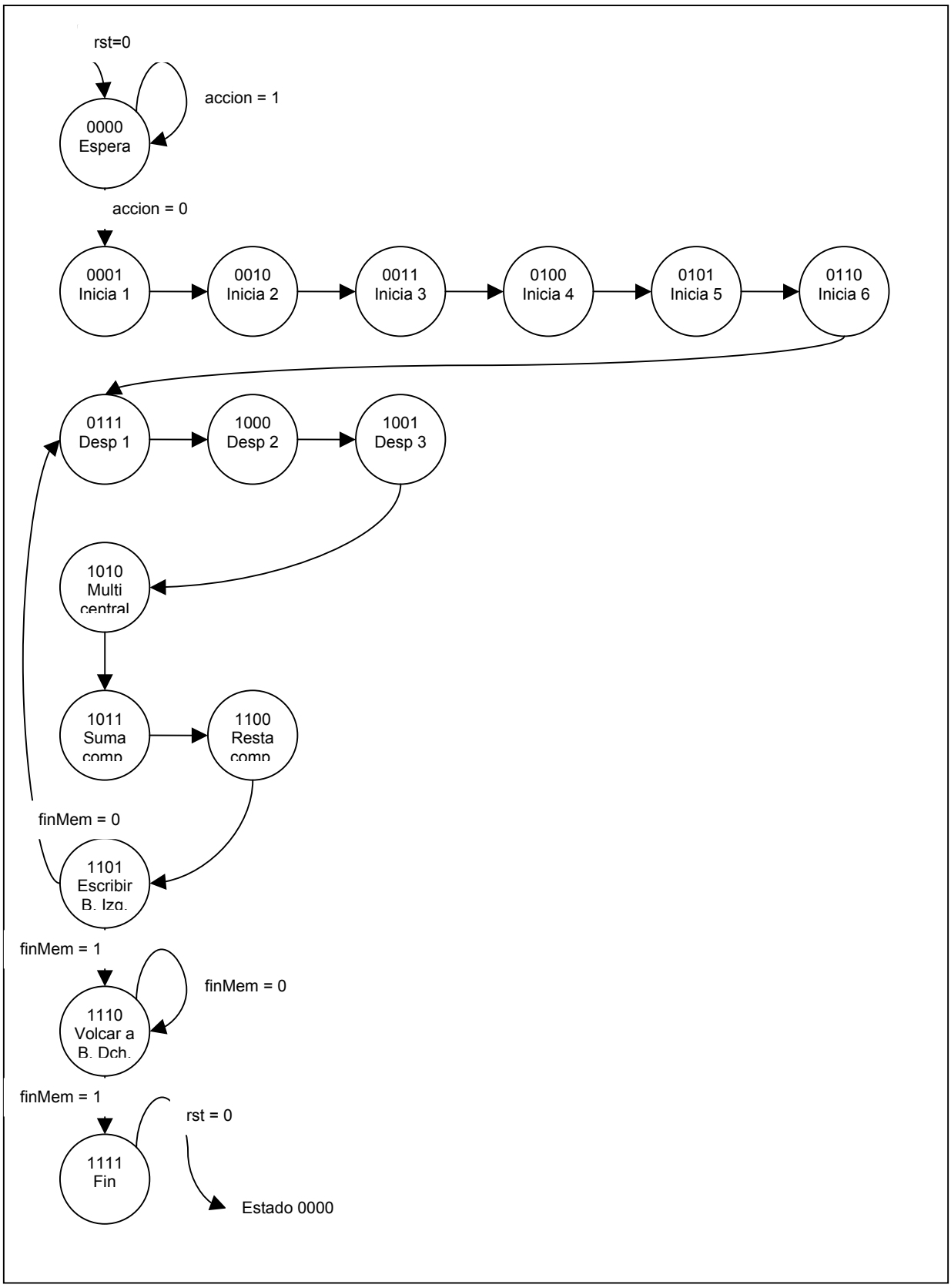


fig. 16 - Diagrama de estados del Filtro de Bordes

Código: Ver anexo C6.6

Problemas encontrados:

Para hacer las pruebas del código de este filtro utilizamos una imagen ya transformada a blanco y negro con el Filtro de paso a Blanco y Negro. Aunque, teóricamente, este filtro de detección de bordes tiene un funcionamiento muy similar al de suavizado, en el momento de redacción de esta memoria todavía no hemos conseguido hacerlo funcionar correctamente. La imagen resultante, no sólo no mostraba las líneas de bordes, sino que, además, incluía colores distintos al blanco y al negro. Continuamos haciendo pruebas con este código para detectar y solucionar los problemas.

5.4 Integración

Llegados a este punto, donde ya teníamos por separado los filtros, el módulo que mostraba una imagen en la RAM (bmpramda) y el cargador, se intentó unirlos todos en dos arquitecturas, cada una con tres filtros y el cargador y el bmpramda, para usarlos luego en la reconfiguración[13].

Este proceso, que en un principio consideramos que iba a resultar muy simple, ha terminado dándonos muchos problemas, hasta el punto de que únicamente se ha conseguido una arquitectura con el bmpramda y el cargador (funcionando perfectamente) y un sólo filtro (el que pasa a escala de grises). De hecho, este filtro no funciona completamente bien (cómo sí lo hace por separado) y deja ciertos residuos en forma de líneas verticales regularmente espaciadas de color más claro del debido.

Hasta llegar a ese punto se han seguido distintos enfoques, cada uno de los cuales era respuesta a los problemas que se iban encontrado.

Proceso

- En un primer momento se unieron los módulos cargador y bmpramda en una arquitectura y se les añadió a cada uno una señal de enable para que sus salidas respectivas estuvieran en alta impedancia cuando los módulos estuvieran desactivados con el enable. Esta señal de enable tomaba para el bmpramda directamente el valor de un dipswitch y para el cargador la misma señal, pero negada. Así, nunca estarían los dos activos al mismo tiempo, con lo que las señales de salida que compartían nunca tendrían dos fuentes, porque una de las dos tendría el enable activo y el otro desactivado, con lo que la línea tendría una fuente activa y la otra en alta impedancia.

  En este momento, la imagen que se mostraba estaba azuleada. Es decir, el byte superior de cada píxel estaba bien, pero el byte

[13] ver sección *Reconfiguración*

menos significativo estaba completamente a 1, así, todo el canal azul y parte del verde estaba a su máximo valor de intensidad, con lo que la imagen parecía vista a través de un cristal azul.

- Después de intentar solucionar el error y no encontrarlo, se decidió intentar juntar el bmpramda con el cargador pero cambiando el enfoque y creando una máquina de estados.

  Surgieron exactamente los mismos problemas.

- Después de trabajar sobre ello durante un tiempo e intentar solucionarlo, se decidió intentar juntar el bmpramda con uno de los filtros y dejar el cargador para más adelante. En este punto, el filtro (que es el de escala de grises, llamado filtroBlancoyNegro) compartía con el bmpramda las entradas y salidas de la RAM, escribiendo y leyendo ambos de ellas a la vez. Obviamente, esto creaba problemas y no hacía nada, mostraba una pantalla completamente blanca.

- Entonces, se separan en el filtro y en el bmpramda las entradas de las salidas de la RAM. Hasta este momento las líneas de datos de la RAM de cada módulo eran de tipo inout. Esto se debía a que cada uno de estos módulos debían funcionar por separado y ellos sólos, con su propio archivo ucf. En estos archivos, un mismo pin no puede ir a dos señales distintas, por lo que las líneas de la RAM, de las que queríamos leer y en las que queríamos escribir, debían ser necesariamente de tipo inout. Ahora, que estaban en una arquitectura, podían tratar por separado, como si conceptualmente fueran dos líneas distintas, las señales de la RAM cuando eran de entrada y de salida.

  Esto hacía necesario que las líneas de la RAM de la arquitectura, que pueden tener dos fuentes distintas (filtro o bmpramda), estén multiplexadas (dependiendo del origen) y tengan fuentes que provengan de triestados, para poder separar las líneas de entrada de las de salida cuando queramos que funcionen como salida y viceversa como muestra la fig. 9.

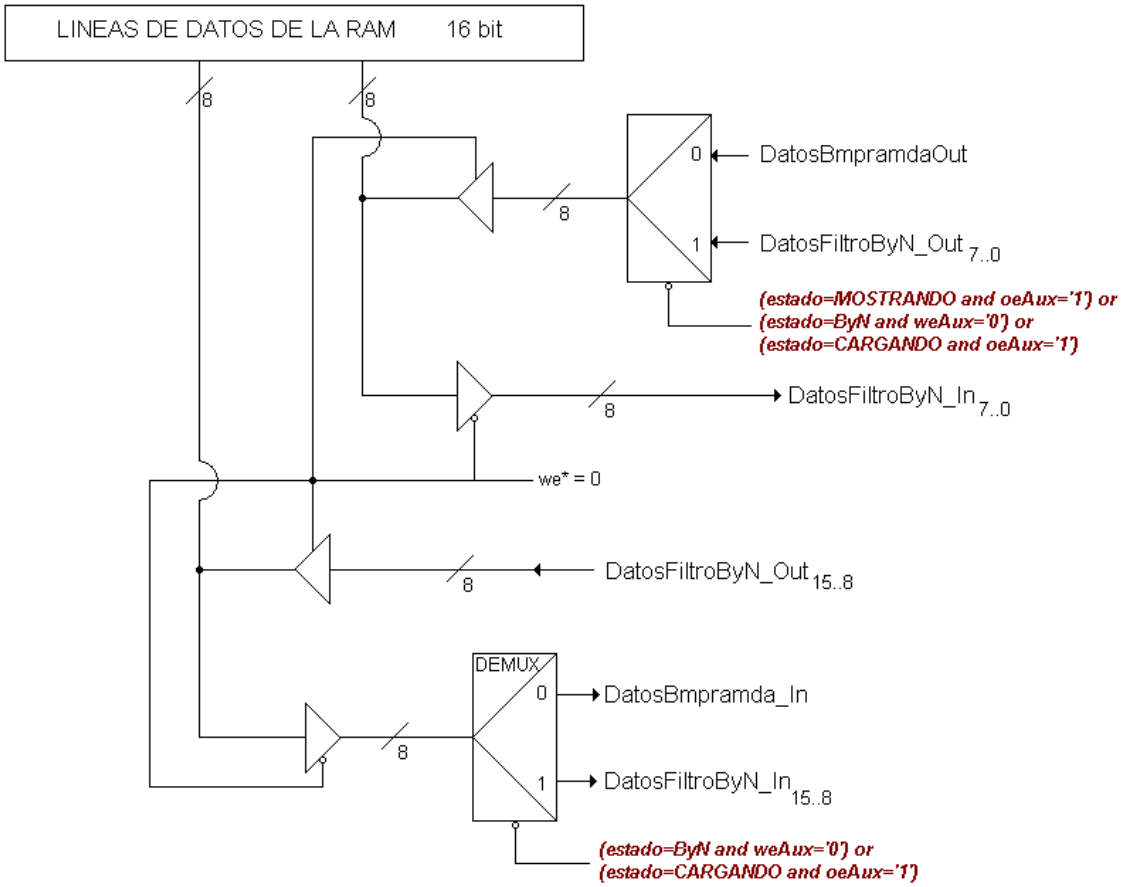


fig. 17 - Comunicación entre la RAM y la Arquitectura global

Llegados a este punto, ya vuelve a funcionar, pero de nuevo azulea, como antes. Después de bastante tiempo depurando y buscando los errores, nos dimos cuenta de que el error estaba (o parecía estar) en el tiempo de propagación de la señal oeaux. Esta señal, por algún motivo, se retrasaba con respecto al estado MOSTRANDO. Este problema (que la imagen saliera azuleada), se solucionó añadiendo la comprobación extra, al controlar el multiplexor de la parte baja de la RAM, `(estado=MOSTRANDO and oeAux='1')`.

Aún así, al filtrar la imagen se generaban unas líneas verticales, regularmente espaciadas, de color más claro del que debería. Este problema no ha conseguido solucionarse.

- Llegados aquí, se añade el módulo cargador, cosa que no da ningún problema, más allá de los retoques necesarios en las comparaciones de los multiplexores y los estados que hacen falta añadir.

- Una vez en este punto, nos dimos cuenta de que no habíamos incluido ningún estado de espera de bajada de la señal de activación del filtro, con lo que una vez que terminaba de funcionar el filtro, si aún se estaba pulsando la señal de

activación, el filtro volvía a activarse. Teniendo en cuenta que nuestros filtros tienen pérdida, esto hacía que la imagen resultante fuera de mala calidad. Lo único que se hizo para solucionar esto fue añadirle un estado de espera de bajada de la señal antes de lanzar el filtro.

Finalmente, la máquina de estados quedo así:

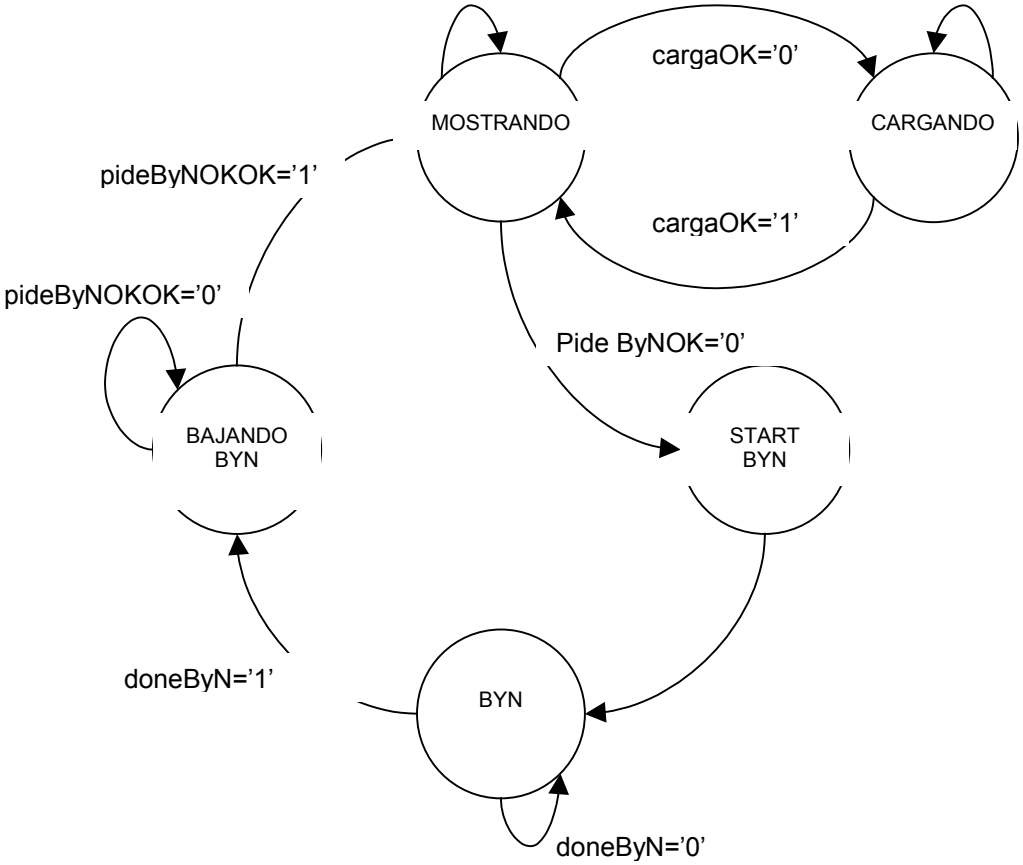


fig.18 - Diagrama de estados de la Arquitectura global

- MOSTRANDO : En este estado tenemos activo el módulo BMPRAMDA y desactivados todos los demás, por lo que se muestra el contenido de la RAM por pantalla. Mantenemos el contador del eliminador de rebotes a '0', es decir, reseteado.
  Saldremos de este estado cuando el usuario solicite una carga o que se aplique el filtro.

- CARGANDO : En este estado desactivamos el BMPRAMDA y activamos el cargador, además de asociar las direcciones de la RAM con las que vienen del módulo cargador. De este estado se sale **sólo** bajo la petición del usuario y en el momento que él diga. Puede salir de este estado antes

de completar la carga (con lo que sólo se cargarían datos hasta el punto en que el usuario bajó el dipswitch). De este estado se vuelve al estado MOSTRANDO.

- STARTByN : En este estado se activa el modulo de filtro, subiendo el reset y dejándo el filtro listo para actuar. De quí sepasa siempre a ByN.

- ByN : En este estado se pide al filtro que empiece a actuar activando la señal accionByN (esta señal se activa a baja). Seguiremos en este estado hasta que se active la señal de doneByN que nos avisa de que el filtro ha terminado. De aquí pasamos a BAJANDOByN.

- BAJANDOByN : Este estado está sólo para esperar a que el usuario deje de pulsar el switch que activa el filtro. Si no estuviera, el filtro se aplicaría continuamente hasta que el usuario deje de pulsar el switch.

A todas las señales se le eliminan rebotes mediante un submódulo eliminador de rebotes llamado rebotes, cuyo código se puede ver en el Apéndice C.

6. RECONFIGURACIÓN

6.1 Objetivo buscado

Para probar la reconfiguración, la idea era crear dos arquitecturas, cada una de ellas con el bmpramda (visualizador de imágenes), el cargador y tres filtros distintos. Con esto, podríamos pedir mediante un dipswitch la reconfiguración total de la placa, y con otro controlar cuál de las dos arquitecturas se cargarían, dependiendo del filtro que quisiéramos usar.

En cualquier caso, en la reconfiguración se empezó a trabajar mientras aún no se habían terminado estas arquitecturas, así que para las pruebas se usaron el contador.vhd (que muestra un patrón de rayas verticales) y una modificación de este (que las muestra horizontales).

6.2 Desarrollo

En un primer momento se intentó crear totalmente por nosotros el archivo de configuración de la Flash, a partir del vhd que controla la configuración del CPLD de la Virtex proporcionado por XESS [VDBO01].

En este primer intento, se crearon una serie de estados que se detallan en la fig. 19 y se añadió un contador de 100ms, para eliminar los rebotes que generaban los dipswitches al código mencionado antes.

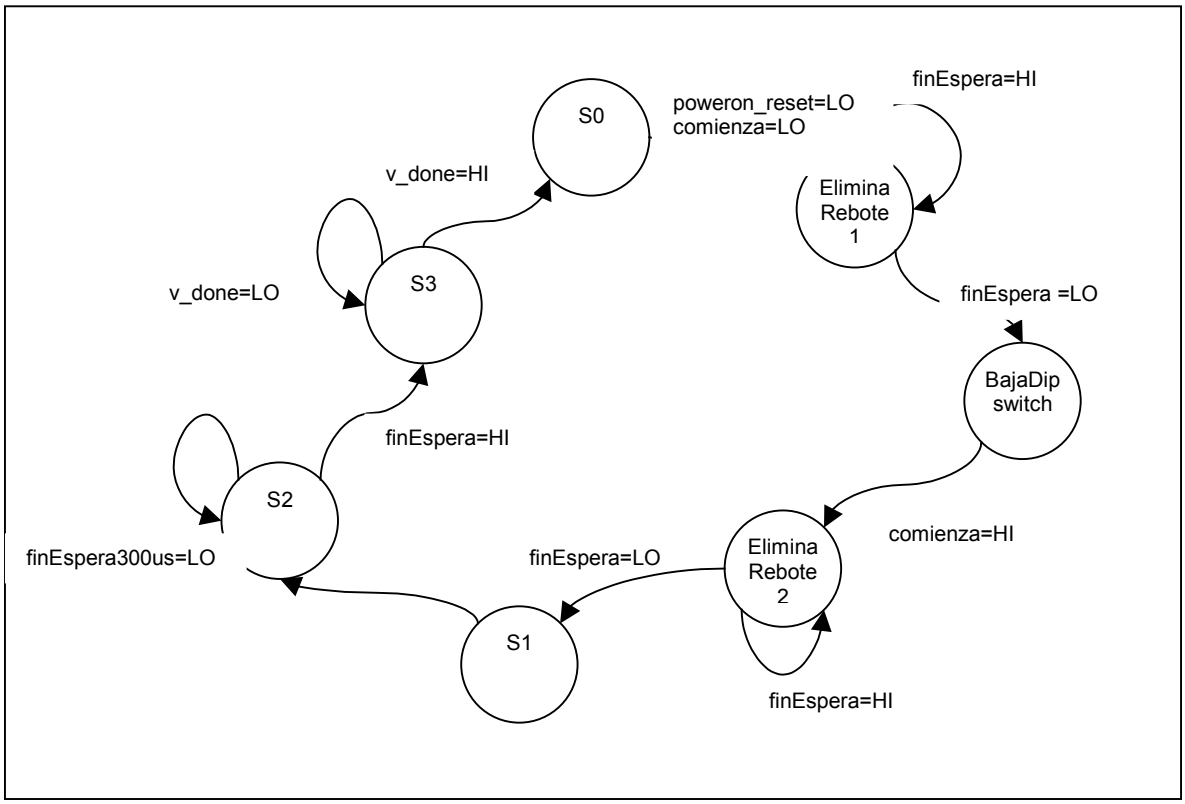


fig. 19 - Diagrama de estados de la reconfiguración

Explicación de los estados:

- S0 : Estado inicial. Seguiremos en este estado hasta que llegue una subida del dipswitch que activa la reconfiguración y además hayan pasado 20 milisegundos desde que se encendió la placa (tiempo que necesita la FPGA para estar lista para reconfigurarse después del encendido).En este estado mantenemos la señal reconf y programb a low para que no se reconfigure la FPGA.
  Al salir de este estado, activamos un contador de 100 milisg. para eliminar rebotes del dipswitch.

- EliminaRebotes1 : Nos quedamos en este estado hasta que sube la señal fin_espera_100ms. De aquí pasamos a BajaDipswitch.

- BajaDipSwitch : Necesitamos esperar a que baje el dipswitch que activa la reconfiguración, porque si no después de terminar la reconfiguración volveríamos a empezarla de nuevo porque el dipswitch seguiría arriba, y así seguiríamos reconfigurando hasta que se bajara el dipswitch, con lo que se harían muchas configuraciones seguidas innecesariamente. Una vez que se detecta la bajada del dipswitch se pasa a un nuevo estado de eliminación de rebotes.

- EliminaRebotes2 : De nuevo nos quedamos en este estado hasta que sube la señal fin_espera_100ms. De aquí pasamos a S1.

- S1 : comenzamos la reconfiguración de la FPGA subiendo la señal programb, lo que hace que se borre. También reseteamos un contador de 300 nanosegundos, tiempo necesario para que se complete el borrado de la FPGA. Pasamos automáticamente al estado S2.

- S2 : me mantengo en este estado, con la señal programb en alta (y por tanto borrando la FPGA), durante 300 nanosg., después de lo cual nos vamos al estado S3.

- S3 : bajamos programb para que se deje de borrar la FPGA y subimos reconf para pedir a la FPGA que se empiece a reconfigurar. La FPGA nos avisará de que ha terminado su reconfiguración cuando eleve la señal V_done, después de lo cual pasaremos a S0 y estaremos listos para una nueva reconfiguración cuando el usuario lo solicite subiendo de nuevo el dipswitch.

Para elegir cuál de las dos configuraciones almacenadas en la FlashRam se carga en la FPGA, en algunas pruebas se tomaba el valor de otro dipswitch para elegirlo. Esto se hacía en el proceso que aumenta la dirección o la inicializa a ceros variando el valor de addr(20), que conincide con el dipswitch 8. El código explicado aquí se puede encontrar en el apéndice D, archivo flashCon.vhd .

Con este código llegamos a conseguir que la placa reconfigurara a petición del usuario con el dipswitch, y no únicamente cuando se encendía. A pesar de esto, no conseguíamos que en tiempo de ejecución se pudiera elegir cuál de los dos códigos almacenados en la FlashRam se debía cargar, por lo que optamos por partir del código realizado en años anteriores por otro proyecto de Sistemas Informáticos [DEHD02].

Se modificó este para que esperara a que se bajara el dipswitch antes de seguir (por lo ya comentado antes) y entonces sí se consiguió que reconfigurara a petición del usuario y además este podía elegir qué configuración se cargaría.

Lamentablemente, la placa en la que se estaba trabajando tuvo algún problema con su CPLD que impedía que pudiera cambiarse su configuración (la del CPLD). Hay ciertos pines que controlan la configuración de la CPLD y que no deben cambiarse de sentido (si son de entrada deben permanecer **siempre** de entrada y viceversa). Si se cambiara su sentido impediría cargar cualquier nueva configuración en la placa. Pero estos pines se comprobaron (en el código, por si se habían cambiado inadvertidamente, y en la misma placa, con un osciloscopio) y estaban bien. Así pues, no conocemos la razón de que el CPLD no pueda ser cambiado de configuración.

De este modo, el CPLD, cargado siempre con la misma configuración y sin posibilidad de cambiarla, inhabilitaba completamente a la placa entera. Al quedar sólo una placa, no podíamos arriesgarnos a seguir trabajando en la reconfiguración hasta no tener reparada la placa dañada, por temor a perder la

única placa que nos quedaba, por lo que el trabajo en esta parte del proyecto se interrumpió.

7. MÓDULOS SOFTWARE DE CARGA Y VISUALIZACIÓN DE IMÁGENES

7.1 Aplicación BMP2PACA

Es la aplicación definitiva para trasformar archivos .BMP a otros formatos. Esta aplicación consta de 5 clases con sus respectivos métodos:

- **Clase principal, GlSkel:**

  Desde aquí se manejan, principalmente, todos las diferentes opciones de los formularios comentados mas abajo. Los métodos mas importantes son:

  - Todos los métodos asociados con el formulario que llaman luego al metodo de la clase **RGBPixmap** relacionado:
    Abrir1Click() → readBMPFile().
    aHEX1Click() → write2HEX().
    aXESS241Click() → write2XESS24().
    apaca1Click() → write2paca().
    PruebaXESS241Click() → writePruebaHEX().
    PruebaHEXClick() → writePruebaXES().
    Todos los métodos relacionados se explican luego en su clase correspondiente.

  - *Parametros1Click():* Metodo asociado al formulario que tambien llama a los métodos de la clase **RGBPixmap** para modificar los parámetros del pixmap según haya especificado el usuario en el formulario **FormParam**

  - *GLScene():* Que se encarga de dibujar en pantalla, si es que ha sido cargado previamente, el pixmap que estemos manejando.

- **Clase matrizRGB:**

  Es la clase que maneja los pixels del pixmap, sólo consta de tres atributos: r,g,b, de tipo unsigned char (8 bits) y los constructores. Parecida a la clase Píxel de la aplicación DibuPACA[14].

- **Clase RGBpixmap:**

[14] ver siguiente sección

Esta es la parte principal de la aplicación; es donde se manejan los pixmaps, se cargan desde el BMP, y se transforman al formato elegido. Estos son los métodos mas destacados:

- *readBMPFile().* Es el metodo que lee los datos del archivo .BMP y los guarda en un objeto de esta clase (pixmap) para poder manejarlo.

- *write2HEX().* Recorre el pixmap y va creando el archivo en ASCII según el formato HEX, metiendo cada píxel del pixmap en un registro HEX.

- *write2XESS24().* Muy similar al anterior pero según el formato XESS-24

- *write2paca().* Este método crea el archivo ASCII, según el formato paca ya explicado anteriormente, a partir del pixmap que previamente se había creado partiendo de la imagen bmp cargada. Para ello, se recorre ese pixmap guardando cada píxel en cada línea del archivo de la siguiente forma:

  Se obtienen del pixmap, los valores correspondientes a las tres componentes de color R, G ,B del píxel en cuestión. Como en el formato bmp se utilizan 8 bits para cada una de estas componentes y nosotros utilizamos 5 bits, se realiza un desplazamiento a la derecha de tres bits para despreciar los tres bits menos significativos. Se escogen estos bits y no los más significativos debido a que estos bits tienen un mayor impacto en el color resultante.

  Tras esto, se desplazan cada una de las componentes un número determinado de bits a la izquierda (el rojo 10 bits, el verde 5 y el azul 0), porque lo siguiente que se hace es sumar estos tres valores para obtener un entero único, que se pasa a un array de caracteres de 16 celdas y este array es lo que se introduce en el archivo final, en la línea por la que vayamos en ese instante. Además, como el formato paca requiere 16 bits, el bit más significativo será un cero, que será despreciado cuando se trate la imagen.

  Por último, en la primera línea del archivo, introducimos un 0 seguido de dos puntos, que indicará la dirección a partir de la cual se empieza a guardar este archivo.

  Este método también se encarga de rellenar con negros la imagen paca desde los bordes de la imagen hasta llenar una imagen de 1024*512, que es el tamaño de la memoria que usaremos.

- *writePruebaHEX().* Aquí se hace una prueba para los HEX: sin partir de ningún pixmap, simplemente se intenta conseguir un archivo en formato en HEX, que luego muestre lo mismo que el contador.vhd

- *writePruebaXES().* Similar al anterior pero siguiendo el formato XESS-24.

Nota: Los métodos para crear archivos en formato HEX y XESS-24, así como las pruebas de los mismos, creemos que están bien implementados en la aplicación, pero no conseguimos probarlos con resultados positivos, debido a los problemas que presentó GXSLoad al trabajar con estos formatos.

- **Clase TGLForm2D (el formulario principal)**

  Es el menú principal de la aplicación, consta de cuatro opciones:

  - Archivo: Que despliega un submenú que te permite cargar una imagen .BMP o salir de la aplicación

  - Pruebas: Permite crear los archivos de prueba para los formatos HEX y XESS-24.

  - Trasformar: Una vez cargado un pixmap en el submenú que aquí se despliega, se podrá elegir el formato al que quieres trasformarlo entre los tres posibles (HEX, XESS-24 y paca)

  - Parámetros: Aparecerá un formulario (comentado acontinuación) para poder cambiar casi cualquier parámetro del pixmap.

- **Clase FormParam (el formulario para los parámetros):**

  - **El formulario para los parámetros (FormParam)**

    Este formulario permite al usuario modificar los siguientes parámetros:

    - Columnas que se pintan: Este campo indica el número de columnas de la pantalla que se van a pintar (en nuestro caso 628 es el valor por defecto), y por tanto el resto estarán en la región de blanking.

    - Filas que se pintan: Este campo es análogo al anterior pero para el caso de las filas. El valor por defecto es de 479.

    - Columnas de toda la pantalla: En realidad, no indica esto sino que indica el tamaño de nuestra memoria o array de

píxeles, es decir, hasta qué número de columnas se tendrá que rellenar con negro desde que se sobrepasa la última columna válida de muestreo. Esto se debe a que el contador que utilizamos para las columnas es un contador de 10 bits, y hasta que no termina la cuenta (es decir, 1024), no llega un nuevo valor a mostrar de la imagen.

- Filas de toda la pantalla: Su función es la misma que en el caso anterior pero para el caso de las filas.

- Desplazamiento Rojo, Verde y Azul: Cuando se lee del array de píxeles en formato .paca, cada uno de estos valores tiene 16 bits, y nuestro código debe quedarse con 5 bits para cada color. Lo que permite este campo es decidir desde que bit empieza el valor de cada una de las intensidades de coloro. En nuestro caso, el valor a despreciar es el bit más significativo y por tanto, los valores por defecto utilizados son 1, 5 y 11.

7.2 Aplicación DIBUPACA

Cuando nos pusimos a intentar mostrar las imágenes con formato .paca, almacenadas en la RAM, decidimos crear una aplicación software en C++ (dibubif.exe), que nos permitiera cargar imágenes .paca y mostrarlas por pantalla. Se optó por esto para facilitar las tareas de depuración, ya que nos podría permitir observar cuál sería el resultado correcto de la visualización de las imágenes.

Además, y con el mismo motivo, se añadió a la aplicación un formulario de parámetros (nº de columnas máximo, nº de filas máximo...) que más adelante explicaremos.

Por último, cuando estábamos trabajando con el filtrado de las imágenes de la RAM, le añadimos a dibubif.exe la funcionalidad de poder realizar una serie de filtros sobre las imágenes cargadas para poder comprobar el correcto funcionamiento de los filtros implementados en vhdl.

La aplicación **dibubif** consta de las siguientes clases y formularios:

- **El formulario para los parámetros (FormParam)**

  Es exactamente el mismo, con los mismos atributos, que el de la aplicación anterior, BMP2PACA

- **El formulario principal (FormPrincipal)**

  Es el formulario principal que le aparece al usuario cuando ejecuta la aplicación**.** Consta de tres menús:

1. *Archivo:* Que al desplegarse ofrece la opción de cargar una imagen o de abandonar el programa.

2. *Parámetros:* Que muestra el formulario de los parámetros

3. *Filtros:* Que al desplegarse da la opción al usuario de realizar un filtro a la imagen cargada. Estos filtros son de Escala de grises , negativizar, binarizar (es decir, pasar blanco y negro puros), 8 colores, suavizado y bordes.

- **La clase Píxel**

  Consta únicamente de los tres atributos (r,g,b) que permiten definir la intensidad de color de ese píxel, y una atributo entero que indica qué posición ocupa dicho píxel dentro del array de píxeles que conforman la imagen.

- **La clase Principal (que en nuestro caso se llama GlSkel)**

  Esta es la clase que realiza todas las posibilidades que muestran los formularios y esta formada por los siguientes métodos relevantes:

  - MenuCargarClick:

    Este método es el encargado de cargar la imagen. Para ello, lee el fichero de la ruta especificada por el usuario y lo va recorriendo guardando, en un pixmap (o array de píxeles), cada valor de píxel obtenido de cada una de las líneas del fichero, así como su posición en dicho array. Es en esta clase donde se utilizan los desplazamientos introducidos en la clase parámetros, para extraer de las líneas del fichero la intensidad correspondiente al rojo, verde y azul del formato RGB que utilizamos.

    Al final de este método se llama a GLScene.

  - GLScene

    Este método es el encargado de pintar en la pantalla el pixmap que hemos leído con el método cargar. Para ello, recorre todo el pixmap, llamando al método draw, que recibe como parámetro un píxel.

  - Draw

    Este método dibuja en pantalla el píxel dado, dándole el color que lleve asociado y colocándolo en las coordenadas correctas, que se calculan mediante el atributo de posición

en el array del píxel. Para la coordenada **x** se calcula el módulo de la posición del píxel, con respecto al número de columnas en pantalla, y para la coordenada **y** análogamente con el número de filas en pantalla.

También utilizamos en este método los métodos TraduceX, TraduceY cuya función es la de conseguir que las x's e y's estén en el rango correcto para OpenGL.

- Escaladegrises1Click

  Este método pasa la imagen cargada a una imagen sólo de grises. Para ello, y conociendo que para obtener una gama de gris basta con asignar el mismo valor al rojo, al verde y al azul, se recorre el pixmap modificando cada píxel mediante la asignación de un mismo tono de gris a las tres componentes de color. Este tono de gris se obtiene de multiplicar cada componente RGB por una constante predefinida, y sumándolas al final.

- Negativizar1Click

  Este método pasa la imagen cargada a una imagen negativizada. Esto no es más que coger cada posición del pixmap y realizar el complemento a 1 de dicho valor (es decir, 1 menos el valor que tuviesen)

- Binarizar1Click

  Este método pasa la imagen cargada a una imagen binarizada, o lo que es lo mismo, una imagen cuyos valores únicos serán el blanco o el negro. Para ello, se realizan los mismos pasos seguidos en el método de la escala de grises, con la diferencia de que cuando se ha obtenido el gris correspondiente para ese píxel, si este valor es inferior a un cierto umbral (en nuestro caso 0.5) el color del píxel pasa a ser 0, es decir negro, y en caso contrario 1 que es el valor del blanco.

- N8coloresClick

  Este método pasa la imagen cargada a una imagen de ocho colores puesto que realizamos lo mismo que en el caso de la binarización (con el mismo umbral) pero para cada una de las 3 componentes de color (rojo, verde y azul), y por tanto, al haber 2 posibilidades por componente tenemos 8 valores posibles.

- Suavizado1Click

Este método pasa la imagen cargada a una imagen suavizada, que consiste en la eliminación de ruidos de la imagen original mediante el cálculo de la media de los colores de los píxeles vecinos. Para ello, se recorre el pixmap, y a cada píxel se le aplica una máscara de convolución que realiza la ya mencionada media de los valores vecinos. Esta máscara es una matriz 3*3 de valores, donde todos son 1 salvo el central que es el valor del píxel a analizar.

- Bordes1Click

Este método pasa la imagen cargada a una imagen donde se pueden apreciar los bordes de la misma, es decir, líneas que separan zonas donde se refleja un claro contraste de color. Para ello, se utiliza el laplaciano (o segunda derivada) que no es más que una máscara isótopa (es decir, la suma de todos los valores de la máscara es 0). Dicha máscara se aplica a todos los píxeles. Un píxel será considerado borde cuando desde él y en alguna dirección se detecte una transición de valor negativo a positivo.

Apéndice A :

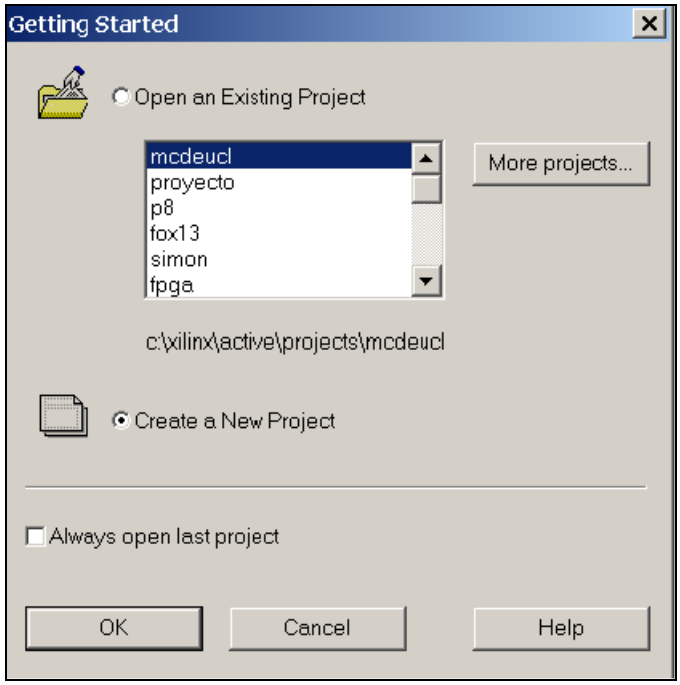
uso de la herramienta Xilinx

Xilinx Foundation Series 3.1i (Build 3.1.179)
Project Manager (Build 6.00.11)

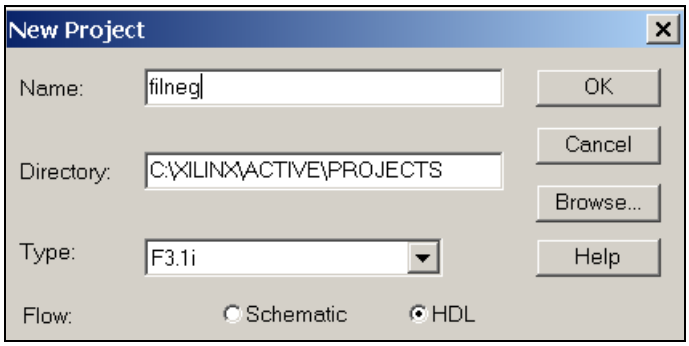
Utilización

Actualmente este software se encuentra disponible en los Laboratorios 7 y 8 de la facultad. En el directorio de arranque Inicio/ Programas/ Electrónica/ Xilinx Foundation Series 3.1i/ Project Manager.

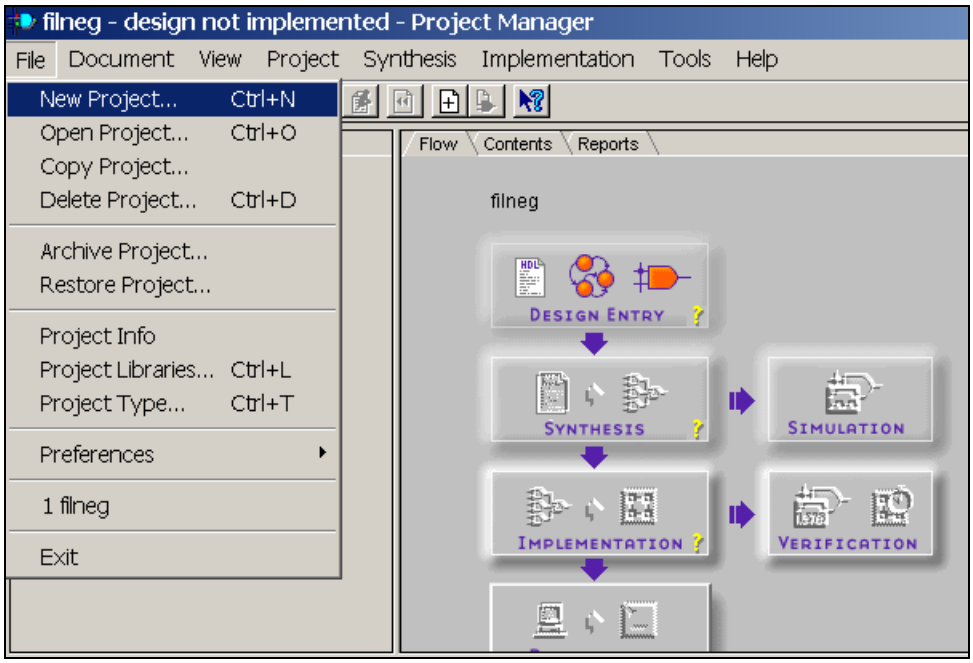
Al arrancar la aplicación salta una pantalla indicando que se seleccione la opción deseada, abrir un proyecto ya existente o crear un nuevo proyecto:



Si se selecciona la opción Create a New Project será necesario rellenar en la siguiente pantalla el nombre del proyecto deseado, además de indicar el tipo de flujo, HDL:

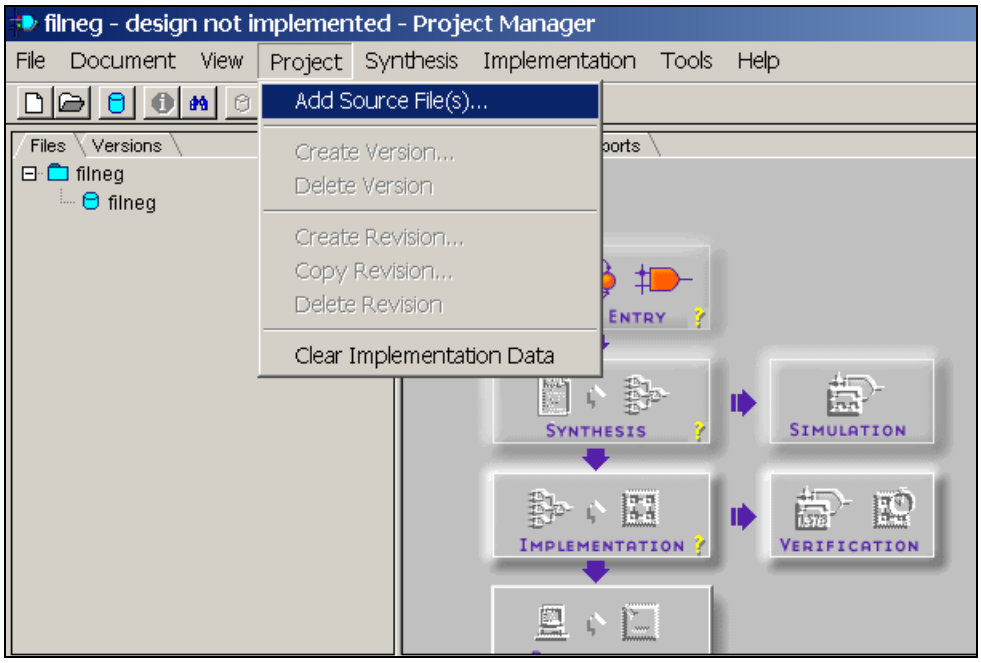


Otra opción para crear un nuevo proyecto sería a través de la barra de herramientas:

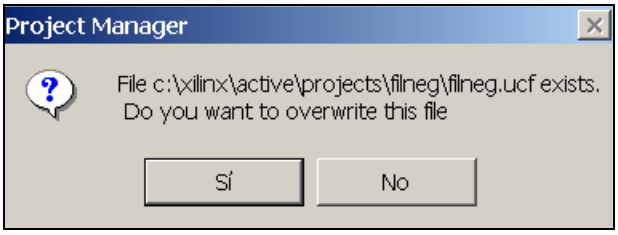


Una vez creado el nuevo proyecto, existe la opción de añadirle ficheros (vhd o ucf[15]) de proyectos anteriores, por si se quiere reutilizar el código ya desarrollado. Para ello seleccionar la opción disponible en el menú superior:

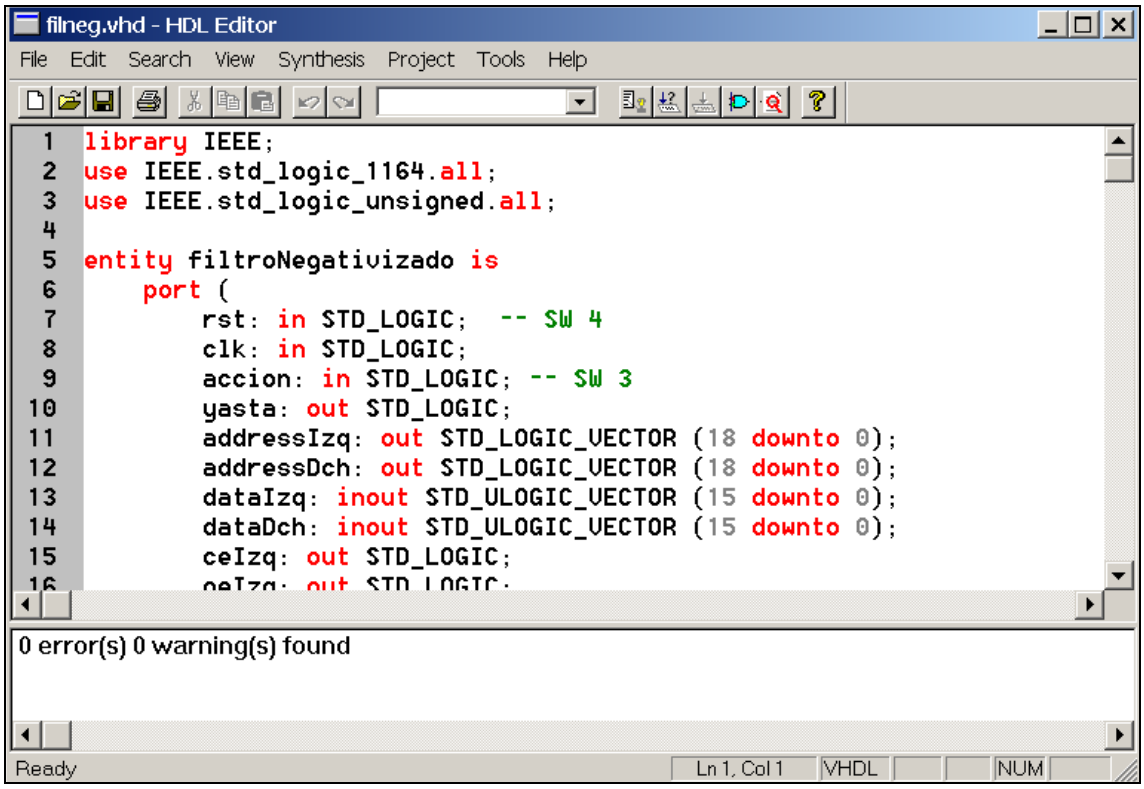
[15] Los ucf son archivos de pines. En estos archivos se asocian las señales de entrada y salida de los vhd con números de pines físicos de la FPGA



Navegar hasta encontrar los ficheros deseados y añadirlos uno por uno. En el momento de añadir el fichero ucf, dado el caso, se pregunta si se desea sobre escribir el ya existente, creado por defecto con el proyecto:



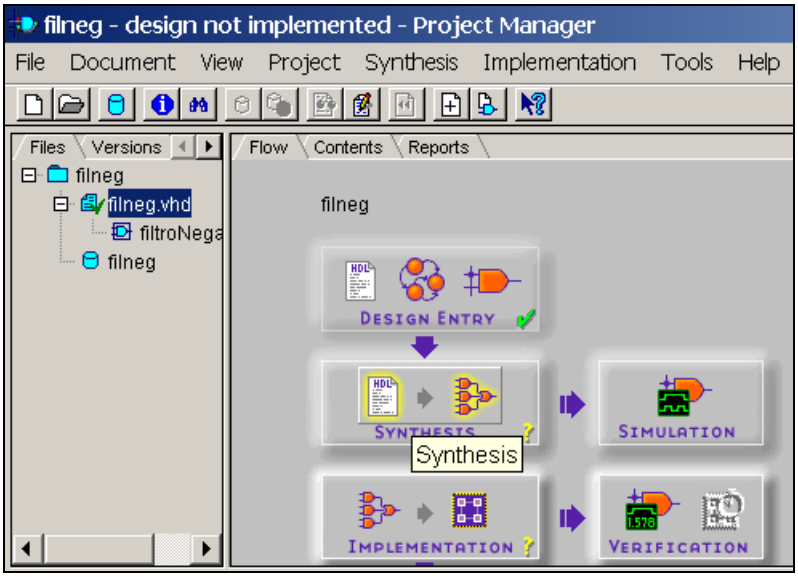
Para desarrollar el código vhdl, y compilarlo, se trabaja con el editor HDL:



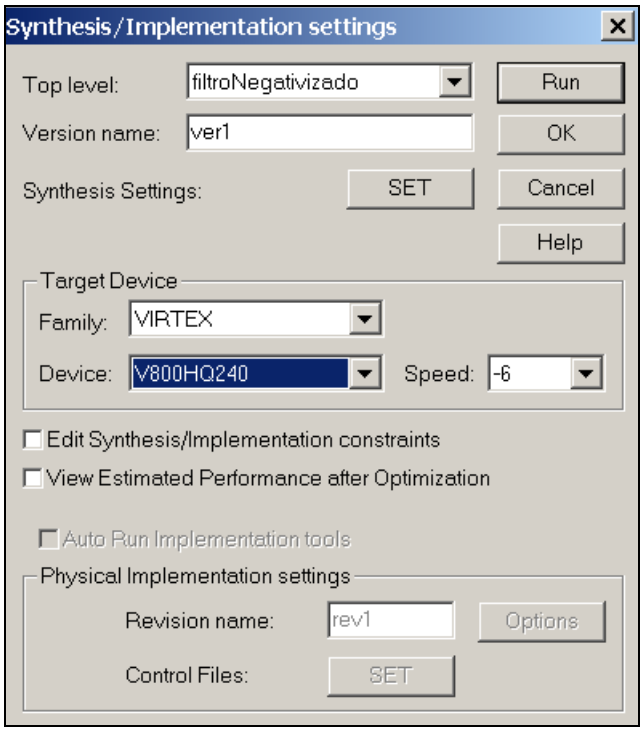
En el que se permite compilar el código, mediante la opción Check Syntax de la barra de herramientas.

Una vez compilado el código sin errores se pasaría a sintetizarlo e implementarlo. El proceso de sintetización está incluído dentro del proceso de implementación, pero, de cualquier manera, en este documento lo veremos por separado.

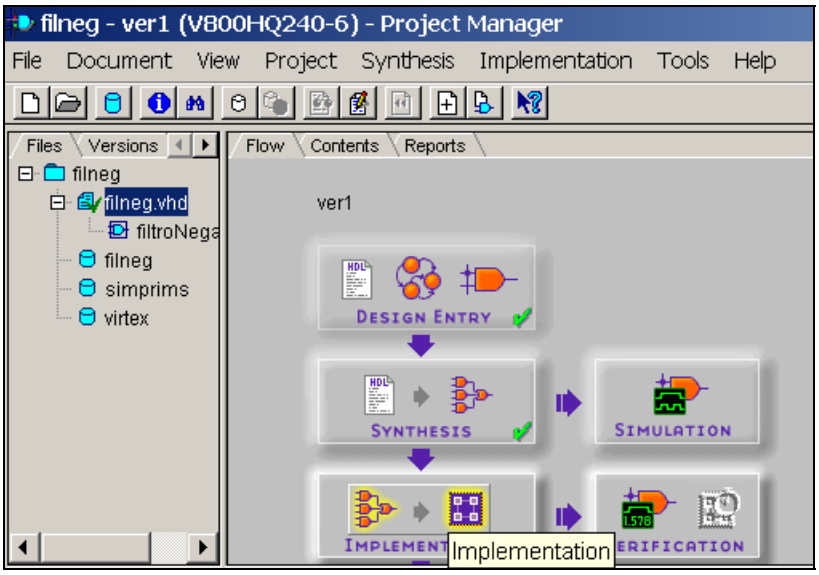
La sintetización se iniciará pulsando el botón creado al efecto en el Project Manager:



En este punto, lo importante es indicar exactamente el tipo de placa en el que se va a ejecutar el código, Virtex V800HQ240 en nuestro caso:

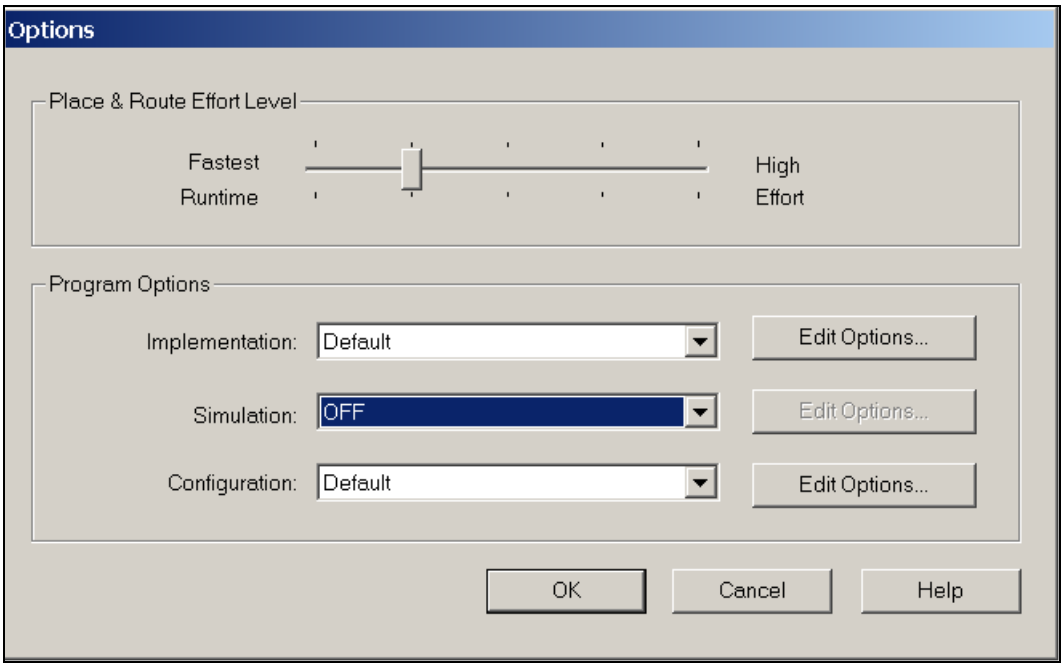


Pulsando el botón Run se iniciaría la síntesis de la primera versión del código. Si este proceso finaliza sin error, se pasaría a la implementación del código, para ello, igual que el caso anterior, pulsar el botón de implementación:



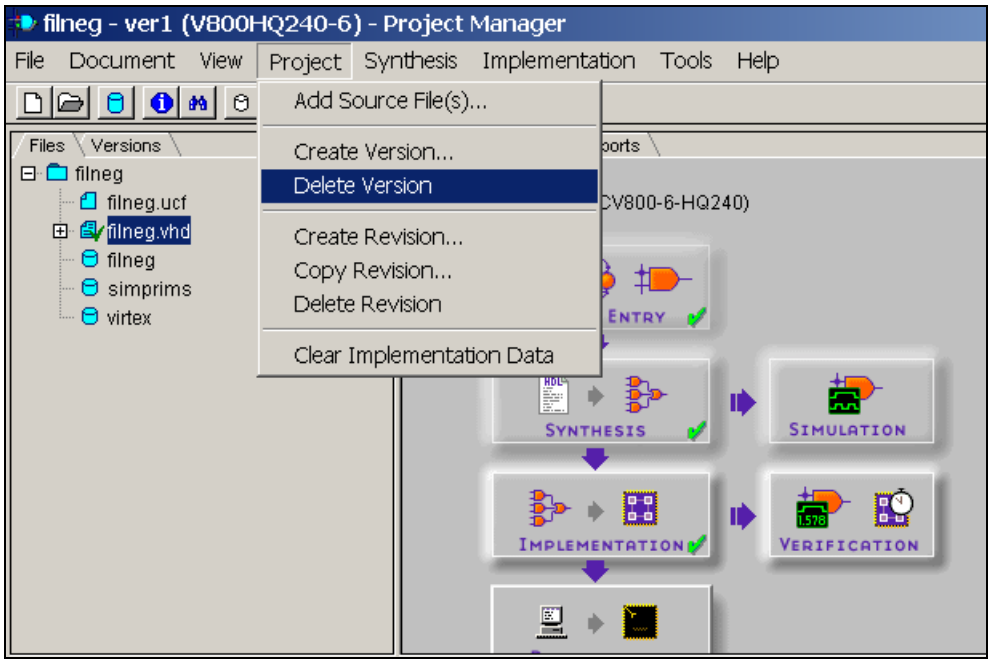
Dependiendo de la máquina en la que se ejecute esta implementación (como los actuales ordenadores de los laboratorios), el tiempo que necesite esta para completarse puede ser realmente importante. Para reducir este tiempo en lo posible, existe la posibilidad de eliminar un paso de la implementación, el Timing. Para

ello seleccionar las opciones de Physical Implementation Settings y fijar a OFF la opción de programa Simulation:



Durante el proceso de implementación se mostrará una pantalla en la que se irá indicando la fase en la que se encuentra en cada momento, Translate, Map, Place&Route y Configure, además de los posibles warnings o errores. Finalmente, si el proceso a terminado sin error, se mostrará la siguiente ventana:

En este punto ya se ha generado el fichero extensión bit que se cargará con el GXSLoad. Si, después de generar dicho fichero bit, se realizan cambios en el código que impliquen nuevas implementaciones, es muy conveniente eliminar la versión generada en este punto, para que nuevas versiones no se solapen con la información generada en un principio. Para eliminar esta versión existe la opción correspondiente en la barra de tareas de la aplicación:



Si se desea transportar el proyecto generado en esta sesión de trabajo será necesario llevarse el fichero <nombre_filtro>.pdf generado, por defecto, en el directorio xilinx/active/projects/filtro.pdf y la carpeta completa con el nombre del filtro. Si sólo se realiza la copia de los ficheros extensión vhd y ucf, se podrá generar un nuevo proyecto, a partir de los mismos, añadiendo dichos ficheros al mismo, tal y como se ha indicado con anterioridad.

Generación de archivos para el CPLD y la Flash

Para realizar la reconfiguración total de la FPGA, es necesario crear mediante la herramienta Xilinx, una serie de archivos cuyos pasos a seguir serán detallados más adelante.

Para guardar una configuración en la FPGA, es necesario configurar el CPLD para que comunique la Flash con el puerto paralelo, guardar una configuración de la FPGA en la Flash y configurar el CPLD con un circuito que cargue la FPGA con la configuración que hayamos almacenado en la memoria Flash.

Para guardar una configuración en la Flash tenemos que:

- Obtener uno o varios archivos bitstream (.bit) que contengan una o más configuraciones válidas de la FPGA.

- Y generar un archivo que permita a la aplicación GXSLOAD cargar esos datos en la Flash con dichos .bit . Este archivo puede ser de tipo .EXO, .MCS, .XES o .HEX

Para generar este archivo se utilizará el siguiente comando:

```
Promgen –u 0 file1.bit –u 100000 file2.bit –p type –s 2048
```

Donde:

- -u NUM indica que el archivo se cargará a partir de la dirección NUM(en hexadecimal)
- -p TYPE indica el tipo de archivo que se generará. En nuestro es un archivo EXO como se explicará a continuación.
- -s SIZE indica el tamaño de la PROM en Kbytes(debe ser potencia de 2).

Esta orden genera un archivo EXO, que se deberá cargar con la aplicación GXSLOAD, y a partir del cual se cargara en la FlashRAM el archivo de configuración file1 a partir de la posición 0 memoria, y el archivo file2 a partir de la posición 1Mbyte.

En nuestro caso se decidió utilizar archivos EXO puesto que pretendíamos cargar al menos dos configuraciones en la FLASH y los otros formatos, como el tipo MCS, dan problemas dado que la versión del PROMGEN incluida en el Xilinx Foundation 3.1, no es capaz de generar archivos de este tipo para tamaños de más de 1Mbyte y como cada archivo de configuración total de la FPGA ocupa 575 Kbytes, sólo es posible cargar un archivo.

No obstante, si se dispone de la versión 4 de Xilinx se puede realizar también la carga de dos archivos con MCS.

Después de cargar la Flash, GXSLoad carga un archivo .svf en el CPLD que automáticamente, al arrancar, permite cargar el primer archivo .bit, existente en la Flash, sobre la fuga. Para poder cargar el resto de configuraciones, se tuvieron que crear nuevos archivos svf, con los que a través del GXSLOAD, se configura el circuito CPLD.

Para generarlos con la herramienta Xilinx Foundation 3.1 se deben realizar los siguientes pasos:

1. Crear un proyecto que incluya los archivos vhdl con el funcionamiento del circuito.
2. Antes de llevar a cabo la fase de síntesis, hay que elegir como modo de codificación de estados el modo binario. Para ello:
    - Pulsar en el menú superior el botón de *Synthesis* y:

      Options: FSM Synthesis: Default encoding : Binary. Ok

    - Forzar el análisis de todos los ficheros fuente:

Síntesis-> Force Analysis of all sources

3. Realizar la síntesis y el análisis eligiendo correctamente la familia y el chip concreto de CPLD. En nuestro caso son:

    - **Family:** XC9500
    - **Device:** 95108TQ100

4. Después de lograr una implementación libre de errores se debe abrir la aplicación JTAG Programmer. Para ello hay que seleccionar en el menú:

    - *Implementation->Tools->JTAG Programmer*

O elegir la opción “Programming” en la ventana principal de la herramienta.

5. Una vez abierta la aplicación hay que realizar los siguientes pasos:
    - En el menú se selecciona: *Output -> Create SVF File…*
    - Elegir la opción “Through Test-Logic-Reset”
    - De nuevo en el menú: *Operations->Program*
      Donde hay que marcar las opciones:
        - *Erase Before Programming*
        - *Verify*

Tras estos pasos, en la carpeta \xproj\ver1\rev1 del proyecto se podrá encontrar el archivo SVF generado.

Este archivo puede ser cargado mediante el GXSLOAD, pero es importante saber que cada vez que se produce la reconfiguración del CPLD, la placa es reiniciada, pasando automáticamente a funcionar el interfaz recién cargado. Este interfaz queda en el CPLD después de apagar la placa, y deberá estar diseñado para cargar una configuración válida de la Flash al arrancar el sistema.

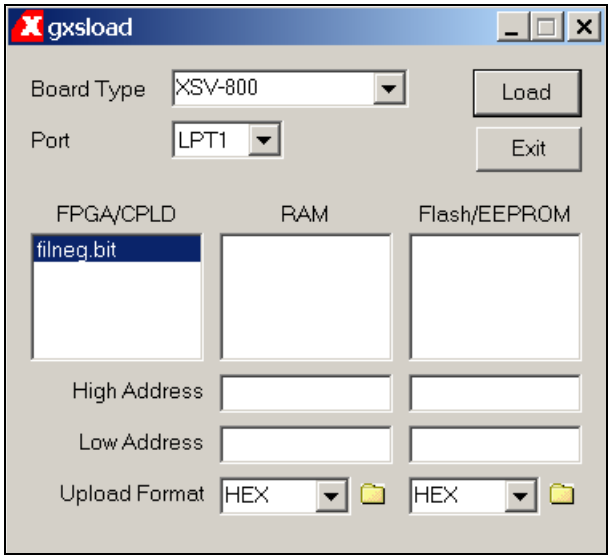
Apéndice B

Uso de la herramienta GXSLOAD

Utilización:

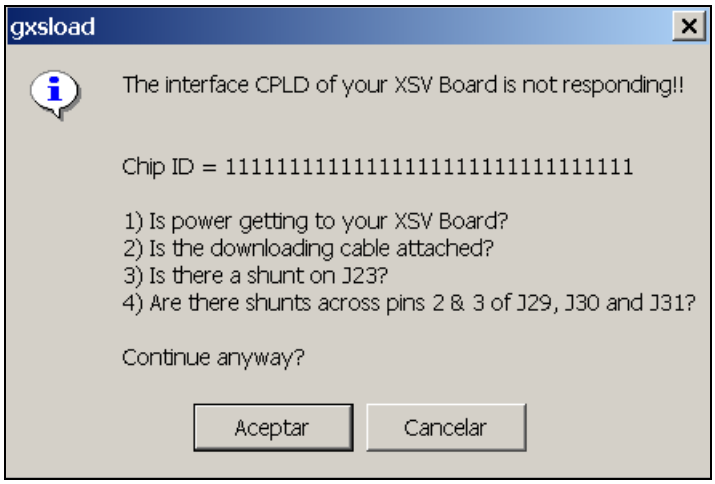
Actualmente este software se encuentra disponible en los Laboratorios 7 y 8 de la facultad. En el directorio de arranque Inicio/ Programas/ Electrónica/ XSTools/ GXSLOAD.

Al iniciar esta aplicación se presentará la siguiente pantalla inicial:



Antes de comenzar a utilizar cualquiera de las opciones de la aplicación es fundamental seleccionar correctamente la placa con la que se está trabajando. En nuestro caso, esta sería la XSV-800. Si no se selecciona correctamente la placa se mostrará un mensaje de error similar a:

Así mismo, si en el momento de carga o recuperación de datos de la placa, esta no está correctamente conectada, o apagada, el mensaje de aviso será:



Esta aplicación tiene tres utilidades principales:

1. Carga de datos en la placa: Para ello se arrastra el fichero, con extensión bit, que se quiera cargar en la FPGA. Con “arrastrar” queremos decir que nos llevaremos el fichero desde su directorio original a la primera ventana por la izquierda de la aplicación. Para cargar el fichero bit se pulsará el botón Load. Los ficheros que se vayan arrastrando a dicha ventana quedan encolados en la misma, de forma que se podrán recargar en cualquier momento. Aunque se haya modificado el fichero origen no será necesario volverlo a arrastrar, ya que la herramienta lo lee cada vez del origen.

2. Recuperación de datos de la memoria Ram: En este caso se trabaja con la ventana central de la aplicación. En Low Address se indica la posición inicial de memoria desde la que queremos recuperar y en High Address la posición final. Para sacar los datos leídos a un fichero plano, con la extensión indicada en Upload Format, se arrastrará el icono de la carpeta al directorio deseado.

3. Carga y recuperación de datos de la FlashRam: En la ventana de la derecha de la aplicación se pueden realizar las dos operaciones mencionadas arriba. Es decir, se pueden cargar datos en la FlashRam y recuperarlos de la misma. Ambos procedimientos son similares a los descritos arriba.

Apéndice C :

Hardware

C.1. Código del módulo q muestra un patrón por pantalla : contador.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity contador is
   port ( clk,rst : in std_logic ;
          filCol : in std_logic;
          hSync,vSync : out std_logic;
          RGB : out std_logic_vector (5 downto 0);
          parpadeo : out std_logic_vector (9 downto 0) );
end contador;

architecture sinRamDac of contador  is
signal contCol,contFil : std_logic_vector (9 downto 0);
signal finCuenta : std_logic;
signal compCol1,compCol2,compCol3 : std_logic;
signal compFil1,compFil2,compFil3 : std_logic;
signal pintura : std_logic_vector (5 downto 0);
begin

   parpadeo <= "0101010101";

   contadorMod794 :
     process( clk,rst )
     begin
        if (rst='0') then
           contCol <= (others=>'0');
        else
           if (contCol = 794) then
              finCuenta<='1';
           else
              finCuenta<='0';
           end if;
           if (clk'event and clk='1') then
             contCol <= contCol + 1;
             if ( finCuenta = '1' ) then
                contCol <= (others=>'0');
             end if;
          end if;
        end if;
     end process;

   contadorMod528 :
     process( clk,rst )
     begin
        if (rst='0') then
           contFil <= (others=>'0');
        else
           if (clk'event and clk='1') then
             if (finCuenta='1') then
                contFil <= contFil + 1;
```

```
                if ( contFil = 528 ) then
                   contFil <= (others=>'0');
                end if;
             end if;
          end if;
       end if;
    end process;

-------------------------------------
-- HACEMOS TODAS LAS COMPARACIONES --
-------------------------------------
   compCol1 <= '0' when (contCol > 628) else '1';
   compCol2 <= '0' when (contCol > 652) else '1';

   --AQUI NO SABEMOS SI ES 746 O 747, HABRA QUE PROBRAR LOS DOS
   compCol3 <= '0' when (contCol < 747) else '1';

   compFil1 <= '0' when (contFil > 479) else '1';
   compFil2 <= '0' when (contFil > 493) else '1';
   compFil3 <= '0' when (contFil < 496) else '1';

------------------------------
-- CALCULAMOS hSync y vSync --
------------------------------

--SABEMOS QUE LAS SEÑALES ESTAN ACTIVAS A ALTA.

   hSync <= '0' when (compCol2 = '0' and compCol3 = '0') else '1';

   vSync <= '0' when (compFil2 = '0' and compFil3 = '0') else '1';

------------------------------
-- AHORA VAMOS CON pintura --
------------------------------

   pintura <= contFil(8 downto 3) when (filCol = '0') else contCol(8
downto 3);

-------------------------------
-- POR ULTIMO TENEMOS EL RGB --
-------------------------------

   RGB <= pintura when (compCol1 = '1' and compFil1 = '1') else
(others=>'0');

end;
```

C.2. Código dl módulo q guarda y lee un patrón de memoria: rayash.vhd

```
                              library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity interfazMem is
    port (
        rst: in STD_LOGIC;
        clk: in STD_LOGIC;
        dir: out STD_LOGIC_VECTOR (18 downto 0);
        datos: out STD_ULOGIC_VECTOR (15 downto 0);
        barrita: out STD_LOGIC_VECTOR (9 downto 0);
        ce: out STD_LOGIC;
        oe: out STD_LOGIC;
        we: out STD_LOGIC;
        hsync,vsync:out STD_LOGIC
    );
end interfazMem;

architecture interfazMem_arch of interfazMem is
signal finPantalla,biOut,tc:STD_LOGIC;
signal contCol,contFil : std_logic_vector (9 downto 0);
begin

   contadorMod794 :
     process( clk,rst )
     begin
      if (contCol = 794) then
               tc<='1';
            else
               tc<='0';
         end if;
        if (rst='0') then
           contCol <= (others=>'0');
        else
          if (clk'event and clk='1') then
            if ( tc = '1' ) then
                contCol <= (others=>'0');
             ELSE
                contCol <= contCol + 1;
             end if;
          end if;
        end if;
     end process;

   contadorMod528 :
     process( clk,rst )
     begin
      if (contFil = 528) then
               finPantalla<='1';
            else
               finPantalla<='0';
      end if;
        if (rst='0') then
           contFil <= (others=>'0');
        else
```

```
           if (clk'event and clk='1') then
            if (tc='1') then
              if ( finPantalla = '1' ) then
                   contFil <= (others=>'0');
             ELSE
                contFil <= contFil + 1;
              end if;
            end if;
          end if;
        end if;
      end process;


   hSync <= '0' when ((contCol > 652) and (contCol < 747)) else '1';

   vSync <= '0' when ((contFil > 493) and (contFil < 496)) else '1';

   --si estamos en la region buena, generamos bien la direccion, y si
no es que estamos
   --en la region de blanking, y entonces miramos en la direccion 0,
que es un 0 y pintamos negro->nada
   dir(9 downto 0) <= contCol when ((ContFil<=479) and (ContCol<=628))
else (others=>'0');
   dir(18 downto 10) <= contFil(8 downto 0) when ((ContFil<=479) and
(ContCol<=628)) else (others=>'0');

-- memoria

   biestable:
   process(clk,rst)
   begin
    if (rst='0') then
       biOut<='0';
    else
       if (clk'event and clk='1') then
          biOut<= biOut or finPantalla;
       end if;
    end if;
   end process;

   we<=biOut;
   oe<=not biOut;
   ce<='0';


 --Metemos en los 6 bits menos significativos de la RAM los bits del 8
al 3 del contador de columnas(que serán los valores a mostrar) en la
primera pasada. Después, cuando el biestable almacene un 1, los
ponemos a 0(que mostrará negro).

  datos(5 downto 0) <= (others=>'0') when (biOut='1') else
contCol(8 downto 3);
  datos(14 downto 6) <= '0';


end interfazMem_arch;
```

C.3. Código del subsistema Cargador:

C.3.1 CargadorMem.vhd

```
--************************************************
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-------------------------------------------------
entity cargadorMem is
      port (
            clk, enable : in std_logic;
            datos : in std_logic_vector(3 downto 0);
            hDat : in std_logic;
            hDir : in std_logic;
            resetP : in std_logic;
            leido : out std_logic;
            ce, oe, we : out std_logic;
            salidaDatos : inout std_logic_vector(15 downto 0);
            salidaDir : out std_logic_vector(18 downto 0);
            enDataOut : out std_logic
 );
end cargadorMem;
-------------------------------------------------
architecture Syn of cargadorMem is

      component CtrlParalelo
            port(
                  clk : in std_logic;

                  -- Conexion con el puerto paralelo
                  datos : in std_logic_vector(3 downto 0);
                  hDat : in std_logic;
                  hDir : in std_logic;
                  resetP : in std_logic;
                  leido : out std_logic;

                  -- Conexion con controlador de memoria
                  escribir: out std_logic;
                  dirs: out std_logic_vector(18 downto 0);
                  dats: out std_logic_vector(15 downto 0);
                  done: in std_logic );
      end component;

      component escribidor
      port (
            clk, reset : in std_logic;
            enable : in std_logic;
            datos : in std_logic_vector(15 downto 0);
            dir : in std_logic_vector(18 downto 0);
            escribir : in std_logic;
            listo : out std_logic;
            ce, oe, we : out std_logic;
            salidaDatos : inout std_logic_vector(15 downto 0);
            salidaDir : out std_logic_vector(18 downto 0);
            enDataOut : out std_logic
      );
      end component;

      signal esc, fin : std_logic;
      signal di : std_logic_vector(18 downto 0);
      signal da : std_logic_vector(15 downto 0);
```

```
begin

ctr:  CtrlParalelo port map (clk, datos, hDat, hDir, resetP, leido,
esc, di, da, fin);
es:   escribidor port map (clk, resetP, enable, da, di, esc, fin, ce,
oe, we, salidaDatos, salidaDir, enDataOut);

end Syn;
-------------------------------------------------
```

C.3.2 CtrlPar.vhd

```
--controla el Puerto paralelo
library ieee;
use ieee.std_logic_1164.all;

entity CtrlParalelo is
      port(
            clk : in std_logic;

            -- Conexion con el puerto paralelo
            datos : in std_logic_vector(3 downto 0);
            hDat : in std_logic;
            hDir : in std_logic;
            resetP : in std_logic;
            leido : out std_logic;

            -- Conexion con controlador de memoria
            escribir: out std_logic;
            dirs: out std_logic_vector(18 downto 0);
            dats: out std_logic_vector(15 downto 0);
            done: in std_logic
      );
end CtrlParalelo;

architecture syn of CtrlParalelo is

      component CONTAD
        generic( N : integer := 8 );
        port(
          CLK     : in     std_logic;
          LOAD    : in     std_logic;
          RESET   : in     std_logic;
          INC     : in     std_logic;
          DEC     : in     std_logic;
          DIN     : in     std_logic_vector (N-1 downto 0);
          DOUT    : buffer std_logic_vector (N-1 downto 0);
          INI     : out    std_logic;
          FIN     : out    std_logic );
      end component;

      component REGGEN
        generic( N : integer := 8 );
        port(
          CLK     : in     std_logic;
          LOAD    : in     std_logic;
          RESET   : in     std_logic;
          DIN     : in     std_logic_vector (N-1 downto 0);
          DOUT    : out    std_logic_vector (N-1 downto 0) );
      end component;
```

```
      component BIESTD_R1
        port(
          CLK     : in     std_logic;
          RESET   : in     std_logic;
          DIN     : in     std_logic;
          DOUT    : out    std_logic
            );
      end component;

      signal CERO, UNO : std_logic;

      signal lLeid, contLeid : std_logic;
      signal entLeid, salLeid : std_logic_vector(2 downto 0);

      signal ldir, contar : std_logic;
      signal entDir, salDir : std_logic_vector(19 downto 0);

      signal ldat : std_logic;
      signal entDat, salDat : std_logic_vector(15 downto 0);

      signal rst : std_logic;

      signal cinco, cuatro : std_logic;
      signal hayDatos, hayDir : std_logic;
      signal hDatSemiSinc, hDirSemiSinc : std_logic;

      type TEstado is (Inicio, LeerDat1, LeerDat2, LeerDat3, LeerDir1,
LeerDir2, LeerDir3, Enviar, AvanzaDir);
      signal est, nest : TEstado;

begin

      CERO <= '0';
      UNO <= '1';
      rst <= resetP;

      -- Contador de bits leidos
      entLeid <= (others=>'0');

leidos:     CONTAD generic map (3)
      port map (clk, lLeid, rst, contLeid, CERO, entLeid, salLeid,
OPEN, OPEN);

      cinco <= '1' when salLeid="101" else '0';
      cuatro <= '1' when salLeid="100" else '0';

      -- Contador de direcciones:
      entDir <= datos & salDir(19 downto 4);

dir:  CONTAD generic map (20)
      port map (clk, ldir, rst, contar, CERO, entDir, salDir, OPEN,
OPEN);

      dirs <= salDir(18 downto 0);

      -- Datos recogidos:
      entDat <= datos & salDat(15 downto 4);

dat:  REGGEN generic map (16)
      port map (clk, ldat, rst, entDat, salDat);
```

```
      dats <= salDat;

      -- Sincronizador:
b1:   BIESTD_R1 port map (clk,rst,hDat,hDatSemiSinc);
b2:   BIESTD_R1 port map (clk,rst,hDatSemiSinc,hayDatos);

b3:   BIESTD_R1 port map (clk,rst,hDir,hDirSemiSinc);
b4:   BIESTD_R1 port map (clk,rst,hDirSemiSinc,hayDir);

      -- Carga del siguiente estado.
      process (clk, rst)
      begin
            if rst = '1' then
                  est <= Inicio;
            elsif clk'event and clk = '1' then
                  est <= nest;
            end if;
      end process;

      -- Maquina de estados de control
      process(est, hayDir, hayDatos, done, cinco, cuatro)
      begin
            ldir <= '0';
            contar <= '0';
            ldat <= '0';
            leido <= '1';
            escribir <= '0';
            lLeid <= '1';
            contLeid<='0';
            nest <= est;
            case est is
                  when Inicio =>
                        if hayDir='0' then
                              nest <= LeerDir1;
                        elsif hayDatos='0' then
                              nest <= LeerDat1;
                        else
                              nest <= Inicio;
                        end if;

                  when LeerDat1 =>
                        lLeid <= '0';
                        contLeid <= '1';
                        ldat <= '1';
                        nest <= LeerDat2;
                  when LeerDat2 =>
                        lLeid <= '0';
                        leido <= '0';
                        if hayDatos='1' then
                              nest <= LeerDat3;
                        else
                              nest <= LeerDat2;
                        end if;
                  when LeerDat3 =>
                        lLeid <= '0';
                        if cuatro='1' then
                              nest <= Enviar;
                        elsif hayDatos='0' then
                              nest <= LeerDat1;
                        else
```

```
                              nest <= LeerDat3;
                        end if;

                  when LeerDir1 =>
                        lLeid <= '0';
                        contLeid <= '1';
                        ldir <= '1';
                        nest <= LeerDir2;
                  when LeerDir2 =>
                        lLeid <= '0';
                        leido <= '0';
                        if hayDir='1' then
                              nest <= LeerDir3;
                        else
                              nest <= LeerDir2;
                        end if;
                  when LeerDir3 =>
                        lLeid <= '0';
                        if cinco='1' then
                              nest <= Inicio;
                        elsif hayDir='0' then
                              nest <= LeerDir1;
                        else
                              nest <= LeerDir3;
                        end if;

                  when Enviar =>
                        escribir <= '1';
                        if done='1' then
                              nest <= AvanzaDir;
                        else
                              nest <= Enviar;
                        end if;

                  when AvanzaDir =>
                        Contar <= '1';
                        nest <= Inicio;

                  when others =>
            end case;
      end process;

end syn;
```

C.3.3 escribidor.vhd

```
--  Módulo que escribe en la memoria. Se latchean las entradas, para
-- que el cliente pueda usarnos como a una memoria síncrona, sin
-- preocuparse de glitches.

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity escribidor is
port (
      clk, reset : in std_logic;
      enable : in std_logic;
      datos : in std_logic_vector(15 downto 0);
      dir : in std_logic_vector(18 downto 0);
      escribir : in std_logic;
      listo : out std_logic;
      ce, oe, we : out std_logic;
```

```
      salidaDatos : out std_logic_vector(15 downto 0);
      salidaDir : out std_logic_vector(18 downto 0);
      enDataOut : out std_logic
);
end escribidor;

architecture rtl of escribidor is

component REGGEN
  generic( N : integer := 8 );
  port(
    CLK     : in     std_logic;
    LOAD    : in     std_logic;
    RESET   : in     std_logic;
    DIN     : in     std_logic_vector (N-1 downto 0);
    DOUT    : out    std_logic_vector (N-1 downto 0) );
end component;

type TEstado is (IDLE, ESCRIBIENDO1, ESCRIBIENDO2, ESCRIBIENDO3);

signal iDatos : std_logic_vector(15 downto 0);
signal iDir : std_logic_vector(18 downto 0);
signal estado, nEstado : TEstado;
signal iwe : std_logic;

begin

-- Triestados para las salidas.
ce <= '0' when enable = '1' else 'Z';
oe <= '1' when enable = '1' else 'Z';
we <= iwe when enable = '1' else 'Z';
salidaDatos <= iDatos when enable = '1' else (OTHERS => 'Z');
salidaDir <= iDir when enable = '1' else (OTHERS => 'Z');

-- Registros para latchear las entradas.
datitos : REGGEN generic map (16)
      port map ( clk, escribir, reset, datos, iDatos);
direccioncillas : REGGEN generic map (19)
      port map (clk, escribir, reset, dir, iDir);

CargaEstado :
process (clk, reset)
begin
      if reset = '1' then
            estado <= IDLE;
      elsif clk'event and clk = '1' then
            estado <= nEstado;
      end if;
end process;

CalculaEstado :
process (estado, escribir)
begin

      listo <= '1';
      iwe <= '1';
      nEstado <= estado;
      enDataOut <= '1';

      case estado is
            when IDLE =>
```

```
                  enDataOut <= '0';

                  if escribir = '1' then
                        nEstado <= ESCRIBIENDO1;
                  end if;

            when ESCRIBIENDO1 =>
                  nEstado <= ESCRIBIENDO2;
                  -- iwe <= '0'; -- Quitado, por si acaso.

            when ESCRIBIENDO2 =>
                  nEstado <= ESCRIBIENDO3;
                  iwe <= '0';

            when ESCRIBIENDO3 =>
                  nEstado <= IDLE;
                  iwe <= '0';
      end case;
end process;

end rtl;
```

C.3.4 contador2.vhd

```
-- CONTADOR GENERICO ASCENDENTE/DESCENDENTE CON SEÑALES
-- DE INCREMENTO Y DECREMENTO SEPARADAS E INDICACIÓN
-- DE INICIO FIN DE LA CUENTA (PRIORIDAD:
-- CARGA>INCREMENTO>DECREMENTO)


library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
--------------------------------------------------
entity CONTAD is
  generic( N : integer := 8 );
  port(
    CLK     : in     std_logic;
    LOAD    : in     std_logic;
    RESET   : in     std_logic;
    INC     : in     std_logic;
    DEC     : in     std_logic;
    DIN     : in     std_logic_vector (N-1 downto 0);
    DOUT    : buffer std_logic_vector (N-1 downto 0);
    INI     : out    std_logic;
    FIN     : out    std_logic );
end CONTAD;
--------------------------------------------------
architecture SYN of CONTAD is

  signal tmp : std_logic_vector (N-1 downto 0);

begin
  process( CLK, LOAD, RESET, INC, DEC, DIN )
  begin
    if RESET='1' then
      DOUT <= (others=>'0');
    elsif CLK'event and CLK='1' then
      if LOAD='1' then
        DOUT <= DIN;
      elsif INC='1' then
```

```
        DOUT <= DOUT + 1;
      elsif DEC='1' then
        DOUT <= DOUT - 1;
      end if;
    end if;
  end process;

  tmp <= (others=>'1');

  FIN <= '1' when (DOUT = tmp) else
         '0';
  INI <= '1' when unsigned(DOUT)=0 else
         '0';

end SYN;
```

C.3.5 biestd_r1.vhd

```
--************************************************
library IEEE;
use IEEE.STD_LOGIC_1164.all;
------------------------------------------------
entity BIESTD_R1 is
  port(
    CLK     : in     std_logic;
    RESET   : in     std_logic;
    DIN     : in     std_logic;
    DOUT    : out    std_logic
      );
end BIESTD_R1;
------------------------------------------------
architecture SYN of BIESTD_R1 is
begin

  process( CLK, RESET, DIN )
  begin
    if RESET='1' then
      DOUT <= '1';
    elsif CLK'event and CLK='1' then
      DOUT <= DIN;
    end if;
  end process;

end SYN;
------------------------------------------------
```

C.3.6 reggen.vhd

```
--************************************************
library IEEE;
use IEEE.STD_LOGIC_1164.all;
------------------------------------------------
entity BIESTD_R1 is
  port(
    CLK     : in     std_logic;
    RESET   : in     std_logic;
    DIN     : in     std_logic;
    DOUT    : out    std_logic
      );
end BIESTD_R1;
------------------------------------------------
architecture SYN of BIESTD_R1 is
begin
```

```
  process( CLK, RESET, DIN )
  begin
    if RESET='1' then
      DOUT <= '1';
    elsif CLK'event and CLK='1' then
      DOUT <= DIN;
    end if;
  end process;

end SYN;
-------------------------------------------------
```

C.4. Código del módulo que lee una imagen de memoria y la muestra:

BmpRamda.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity bmpRamda is
    port (
        clk: in STD_LOGIC;
        rst: in STD_LOGIC;
        hSync: out STD_LOGIC;
        vSync: out STD_LOGIC;
        Dout: out STD_LOGIC_VECTOR (7 downto 0);
        BlankOut: out STD_LOGIC;
        PixelCLKout: out STD_LOGIC;
        RDout: out STD_LOGIC;
        WRout: out STD_LOGIC;
        RSout: out STD_LOGIC_VECTOR (2 downto 0);
        Pout: out STD_LOGIC_VECTOR (7 downto 0);
        WEout: out STD_LOGIC;
        OEout: out STD_LOGIC;
        CEout: out STD_LOGIC;
        DireccOut: out STD_LOGIC_VECTOR (18 downto 0);
        DatosIn: in STD_LOGIC_VECTOR (15 downto 8);
        trste:out STD_LOGIC
    );
end bmpRamda;

architecture bmpRamda_arch of bmpRamda is
signal hSyncAux,vSyncAux : STD_LOGIC_VECTOR (7 downto 0);
signal contCol,contFil : std_logic_vector (9 downto 0);
signal contColAux : std_logic_vector (10 downto 0);
signal divFrec2 : std_logic_vector (1 downto 0);
signal divFrec4 : std_logic_vector (2 downto 0);
signal tc,ini,OEaux,blanking,biOut : std_logic;
signal datosMasSig : std_logic_vector (15 downto 8);
signal clk25:std_logic;
begin
  TRSTE<='1';
--Contador modulo 1578 para llevar la cuenta de las columnas
    contadorMod1578 :
     process( clk,rst )
     begin
      if (contColAux = 1578) then
              tc<='1';
           else
              tc<='0';
         end if;
        if (rst='0') then
           contColAux <= (others=>'0');
        else
          if (clk'event and clk='1') then
            if ( tc = '1' ) then
                contColAux <= (others=>'0');
             ELSE
                contColAux <= contColAux + 1;
             end if;
```

```
          end if;
        end if;
      end process;
  contCol <= contColAux(10 downto 1);

--Contador modulo 528 para llevar la cuenta de las filas
   contadorMod528 :
     process( clk,rst )
     begin
        if (rst='0') then
           contFil <= (others=>'0');
        else
          if (clk'event and clk='1') then
            if (tc='1') then
               if (contFil = 528) then
                  contFil <= (others=>'0');
               else
                  contFil <= contFil + 1;
               end if;
            end if;
          end if;
        end if;
     end process;



--Retrasamos las segnales de hSync y vSync 8 ciclos debido al retraso
que se produce
--porque los pixeles cogen delay al atravesar la DAC.

   hSync <= hSyncAux(7);
   vSync <= vSyncAux(7);

--Proceso que nos permite retrasar las señales mediante una señal
auxiliar
   retrasador:
      process( clk,rst )
      begin
         if (rst='0') then
            hSyncAux <= (others=>'1');
            vSyncAux <= (others=>'1');
         else
            if (clk'event and clk='1') then
               hSyncAux(7 downto 1) <= hSyncAux(6 downto 0);
               if ((contCol > 652) and (contCol < 747)) then
                  hSyncAux(0) <= '0';
               else
                  hSyncAux(0) <= '1';
               end if;

              vSyncAux(7 downto 1) <= vSyncAux(6 downto 0);
               if ((contFil > 493) and (contFil < 496)) then
                  vSyncAux(0) <= '0';
               else
                  vSyncAux(0) <= '1';
               end if;
            end if;
         end if;
      end process;
```

```
   --si estamos en la region buena, generamos bien la direccion, y si
no es que estamos
   --en la region de blanking, y entonces miramos en la direccion 0,
que es un 0 y pintamos negro->nada
--   DireccOut(9 downto 0) <= contCol when ((ContFil<=479) and
(ContCol<=628)) else (others=>'0');
DireccOut(18 downto 10) <= contFil(8 downto 0) when ((ContFil<=479)
and (ContCol<=628)) else (others=>'0');


   DireccOut(9 downto 0) <= contCol;
   DireccOut(18 downto 10) <= contFil(8 downto 0);

   --Cuando escribamos en la RAM tendremos que modificar esto
   WEout <= '1'; --vale siempre 1 (es decir, esta desactivado porque
no escribimos en la RAM)

   CEout <= '0'; --activamos el chip enable


--OEout vale 0 en el primer ciclo (para leer) y 1 en el segundo
ciclo(para no leer)

   OEaux <= contColAux(0) when (ini='1') else '1';
   OEout <= OEaux;

   Pout <= (others=>'Z') when (OEaux='0') else DatosIn;


   ------------------
   --INICIALIZACION--
   ------------------

   Dout <= "10100000";

   blanking <= '1' when ((ContFil<=479) and (ContCol<=628)) else '0';
   BlankOut <= '0' when (ini='0') else blanking;


  PixelCLKout <= clk;

   RDout <='1';

   RSout <= "110";


   --NECESITAMOS QUE WR out ESTE ACTIVO DURANTE TRES CICLOS, COMO
MINIMO, Y NOSOTROS LO DEJAMOS 4 CICLOS
   divisorFrecuencia4:
   process( clk,rst )
     begin
        --REVISAR AQUI SI HACE FALTA CAMBIAR MANUALMENTE DE 111 A 000
        if (rst='0') then
           divFrec4 <= (others=>'0');
        else
          if (clk'event and clk='1') then
            divFrec4 <= divFrec4 + 1;
          end if;
        end if;
     end process;
```

```
   biestable:
   process(clk,rst)
   begin
    if (rst='0') then
       biOut<='0';
   else
       if (clk'event and clk='1') then
          biOut<= biOut or divFrec4(2);
      end if;
    end if;
   end process;

   ini <= biOut;

   WRout <= ini;

end bmpRamda_arch;
```

C.5. Código de la arquitectura (módulo de control de la pantalla, cargador y filtro blanco y negro):

arqImg3.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity arqImag3 is
    port (

        clk: in STD_LOGIC;
        rst: in STD_LOGIC;

        pideByN : in STD_LOGIC;
        pideBinario : in std_logic;

        carga: in STD_LOGIC;

        hSync: out STD_LOGIC;
        vSync: out STD_LOGIC;
        Dout: out STD_LOGIC_VECTOR (7 downto 0);
        BlankOut: out STD_LOGIC;
        PixelCLKout: out STD_LOGIC;
        RDout: out STD_LOGIC;
        WRout: out STD_LOGIC;
        RSout: out STD_LOGIC_VECTOR (2 downto 0);

        trste:out STD_LOGIC;

      datos : in std_logic_vector(3 downto 0);
      hDat : in std_logic;
      hDir : in std_logic;
      resetP : in std_logic;
      leido : out std_logic;
      ce, oe, we : out std_logic;
      salidaDatos : inout std_logic_vector(15 downto 0);
```

```
      salidaDir : out std_logic_vector(18 downto 0);
      enDataOut : out std_logic;

      barra : out std_logic_vector(9 downto 0)
          );
end arqImag3;

architecture uno of arqImag3 is

   component bmpRamda
      port(
        clk: in STD_LOGIC;
        rst: in STD_LOGIC;

        --enable: in STD_LOGIC;

        hSync: out STD_LOGIC;
        vSync: out STD_LOGIC;
        Dout: out STD_LOGIC_VECTOR (7 downto 0);
        BlankOut: out STD_LOGIC;
        PixelCLKout: out STD_LOGIC;
        RDout: out STD_LOGIC;
        WRout: out STD_LOGIC;
        RSout: out STD_LOGIC_VECTOR (2 downto 0);
        Pout: out STD_LOGIC_VECTOR (7 downto 0);
        WEout: out STD_LOGIC;
        OEout: out STD_LOGIC;
        CEout: out STD_LOGIC;
        DireccOut: out STD_LOGIC_VECTOR (18 downto 0);
        DatosIn: in STD_LOGIC_VECTOR (15 downto 8);
        trste:out STD_LOGIC
      );
    end component;

    component cargadorMem
       port(
          clk : in std_logic;
          enable : in std_logic;
        datos : in std_logic_vector(3 downto 0);
        hDat : in std_logic;
        hDir : in std_logic;
        resetP : in std_logic;
        leido : out std_logic;
        ce, oe, we : out std_logic;
        salidaDatos : out std_logic_vector(15 downto 0);
        salidaDir : out std_logic_vector(18 downto 0);
        enDataOut : out std_logic
       );
    end component;

    component rebotes
       port(
         clk: in STD_LOGIC;
         rst: in STD_LOGIC;

         accion : in STD_LOGIC;
         accionOK : out STD_LOGIC
       );
    end component;

    component filtroBlancoNegro
```

```
       port(
          rst: in STD_LOGIC;  -- SW 4
          clk: in STD_LOGIC;
          accion: in STD_LOGIC; -- SW 3
          yasta: out STD_LOGIC;
          addressDch: out STD_LOGIC_VECTOR (18 downto 0);
          dataDchIn: in STD_LOGIC_VECTOR (15 downto 0);
          dataDchOut : out STD_LOGIC_VECTOR (15 downto 0);
          ceDch: out STD_LOGIC;
          oeDch: out STD_LOGIC;
          weDch: out STD_LOGIC
        );
    end component;

    component filtroBinarizar
    port (
        rst: in STD_LOGIC;
        clk: in STD_LOGIC;
        accion: in STD_LOGIC;
        yasta: out STD_LOGIC;
        addressDch: out STD_LOGIC_VECTOR (18 downto 0);
        dataDchIn: in STD_LOGIC_VECTOR (15 downto 0);
        dataDchOut: out STD_LOGIC_VECTOR (15 downto 0);
        ceDch: out STD_LOGIC;
        oeDch: out STD_LOGIC;
        weDch: out STD_LOGIC
        );
    end component;

    constant E : std_logic_vector(9 downto 0) := "0110000000";
    constant S : std_logic_vector(9 downto 0) := "0001100000";
    constant C : std_logic_vector(9 downto 0) := "0000011000";
     constant B : std_logic_vector(9 downto 0) := "0000000110";

    type TEstado is (MOSTRANDO, start1ByN, ByN, startBinario, Binario,
SUBIENDO, CARGANDO, bajandoByN, bajandoBinario);
    signal nEstado,estado : TEstado;
    signal enableCargador,enableParranda,
         doneByN,doneBinario : STD_LOGIC;
    signal enableOK,
         pideByNOK,pideBinarioOK,
         accionByN,accionBinario : STD_LOGIC;
    signal datosOutCargador,
           datosInFiltroByN,
           datosOutFiltroByN,
           datosInFiltroBinario,
           datosOutFiltroBinario : STD_LOGIC_VECTOR(15 downto 0);
    signal datosOutParranda : STD_LOGIC_VECTOR(7 downto 0);
    signal datosInParranda : STD_LOGIC_VECTOR(15 downto 8);
    signal muxSalida : STD_LOGIC_VECTOR(7 downto 0);
    signal ceByN,oeByN,weByN,
           ceP,oeP,weP,
           ceBinario,oeBinario,weBinario,
           ceCarg,oeCarg,weCarg,
           ceAux,oeAux,weAux,
           rstByN,rstBinario : std_logic;
    signal dirFiltroByN,dirParranda,dirCargador,dirFiltroBinario:
STD_LOGIC_VECTOR (18 downto 0);

    signal cargaOK : std_logic;
```

```
begin

CambiaEstados:
PROCESS(clk)
BEGIN
  IF (clk'event AND clk = '1') THEN
     estado <= nEstado;
  END IF;
END PROCESS;

GeneradorDeEstados :
process (enableOK,rst)
begin
     if rst = '0' then
        nEstado <= MOSTRANDO;
     else
      case estado is
            when MOSTRANDO =>
                    if (cargaOK = '0') then
                       nEstado <= CARGANDO;
                     elsif(pideByNOK = '0') then
                           nEstado <= start1ByN;
                        else
                           nEstado <= MOSTRANDO;
                  end if;

            when start1ByN =>
                  nEstado <= ByN;

            when ByN =>
                  if (doneByN = '0') then
                     nEstado <= ByN;
                  else
                     nEstado <= bajandoByN;
                  end if;

            when startBinario =>
                  nEstado <= Binario;

            when Binario =>
                  if (doneBinario = '0') then
                     nEstado <= Binario;
                  else
                     nEstado <= bajandoBinario;
                  end if;

            when CARGANDO =>
                  if (cargaOK = '1') then
                       nEstado <= MOSTRANDO;
                  else
                     nEstado <= CARGANDO;
                  end if;

            when bajandoByN =>
                    if (pideByNOK = '0') then
                     nEstado <= bajandoByN;
                  else
                     nEstado <= MOSTRANDO;
                  end if;

            when bajandoBinario =>
```

```
                    if (pideBinarioOK = '0') then
                     nEstado <= bajandoBinario;
                  else
                     nEstado <= MOSTRANDO;
                  end if;

            when others => nEstado <= MOSTRANDO;
      end case;
    end if;
end process;

SalidaDeEstados :
process (estado)
begin
      case estado is
            when MOSTRANDO =>
                        rstByN <= '0';
                        accionByN <= '1';
                        rstBinario <= '0';
                        accionBinario <= '1';
                        barra <= E;
                        barra(0) <= doneByN or doneBinario;
                        salidaDir <= dirParranda;
                        ceAux <= ceP;
                        oeAux <= oeP;
                        weAux <= weP;
                        enableParranda <= '1';

            when start1ByN =>
                        rstByN <= '1';
                        accionByN <= '1';
                        barra <= S;
                        barra(0) <= doneByN;
                        enableParranda <= '0';

            when startBinario =>
                        rstBinario <= '1';
                        accionBinario <= '1';
                        barra <= S;
                        barra(0) <= doneBinario;
                        enableParranda <= '0';

            when ByN =>
                        rstByN <= '1';
                        accionByN <= '0';
                        barra <= B;
                        barra(0) <= doneByN;
                        salidaDir <= dirFiltroByN;
                        ceAux <= ceByN;
                        oeAux <= oeByN;
                        weAux <= weByN;
                        enableParranda <= '0';

            when Binario =>
                        rstBinario <= '1';
                        accionBinario <= '0';
                        barra <= B;
                        barra(0) <= doneBinario;
                        salidaDir <= dirFiltroBinario;
                        ceAux <= ceBinario;
                        oeAux <= oeBinario;
```

```
                        weAux <= weBinario;
                        enableParranda <= '0';

            when CARGANDO =>
                        barra <= C;
                        salidaDir <= dirCargador;
                        ceAux <= ceCarg;
                        oeAux <= oeCarg;
                        weAux <= weCarg;
                        enableParranda <= '0';

            when bajandoByN =>
                        rstByN <= '0';
                        accionByN <= '1';
                        barra <= E;
                        barra(0) <= doneByN;
                        ceAux <= '1';
                        oeAux <= '1';
                        weAux <= '1';
                        enableParranda <= '0';

            when bajandoBinario =>
                        rstBinario <= '0';
                        accionBinario <= '1';
                        barra <= E;
                        barra(0) <= doneBinario;
                        ceAux <= '1';
                        oeAux <= '1';
                        weAux <= '1';
                        enableParranda <= '0';

            when others => NULL;
      end case;
end process;

--Datos comunes de la RAM
oe <= oeAux;
ce <= ceAux;
we <= weAux;

--Parte baja de los datos de la RAM
muxSalida <= datosOutParranda when (estado=MOSTRANDO) else
             datosOutFiltroByN(7 downto 0) when (estado=ByN) else
             datosOutCargador(7 downto 0) when (estado=CARGANDO) else
(others=>'Z');
salidaDatos(7 downto 0) <= muxSalida when ((estado=MOSTRANDO and
oeAux='1') or
                                (estado=ByN and weAux='0') or
                                (estado=CARGANDO and oeAux='1')) else
(others=>'Z');
datosInFiltroByN(7 downto 0) <= salidaDatos(7 downto 0) when
(estado=ByN and oeAux='0') else ("00110011");
--datosInFiltroBinario(7 downto 0) <= salidaDatos(7 downto 0) when
(estado=Binario and oeAux='0') else ("00110011");

--Parte alta de los datos de la RAM
salidaDatos(15 downto 8) <= datosOutFiltroByN(15 downto 8) when
(estado=ByN and weAux='0') else
                     datosOutCargador(15 downto 8) when
(estado=CARGANDO and oeAux='1') else (others=>'Z');
```

```
datosInParranda <= salidaDatos(15 downto 8) when (estado=MOSTRANDO)
else ("00000111");
datosInFiltroByN(15 downto 8) <= salidaDatos(15 downto 8) when
(estado=ByN and oeAux='0') else ("00110011");
 (estado=Binario and oeAux='0') else ("00110011");

--Submodulos
eliminadorRebotesByN : rebotes port map(clk,rst,pideByN,pideByNOK);
eliminadorRebotesCarga : rebotes port map(clk,rst,carga,cargaOK);
--eliminadorRebotesBinario : rebotes port
map(clk,rst,pideBinario,pideBinarioOK);

parranda : bmpramda port map(
                  clk,rst,
                  hSync,vSync,
                  Dout,BlankOut,PixelCLKout,
                  RDout,WRout,RSout,
                  datosOutParranda,
                  weP,oeP,ceP,
                  dirParranda,
                  datosInParranda,
                  trste
                      );
enableCargador <= not cargaOK;
modulo_cargador : cargadorMem port map(
                   clk,enableCargador,datos,hDat,hDir,resetP,leido,
                   ceCarg,oeCarg,weCarg,
                   datosOutCargador,
                   dirCargador,
                   enDataOut
                        );

filtroByN : filtroBlancoNegro port map(
                  rstByN,clk,
                  accionByN,doneByN,
                  dirFiltroByN,
                  datosInFiltroByN,datosOutFiltroByN,
                  ceByN,oeByN,weByN);


end uno;
```

C.6. Código de los filtros

C.6.1 Filtro de 8 colores : fil8co.vhd

```
                              library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity filtro8Colores is
    port (
        rst: in STD_LOGIC;  -- SW 4
        clk: in STD_LOGIC;
        accion: in STD_LOGIC; -- SW 3
        yasta: out STD_LOGIC;
        addressDch: out STD_LOGIC_VECTOR (18 downto 0);
        dataDch: inout STD_ULOGIC_VECTOR (15 downto 0);
        ceDch: out STD_LOGIC;
        oeDch: out STD_LOGIC;
```

```
        weDch: out STD_LOGIC
        );
end filtro8Colores;

architecture filtro8Colores_arch of filtro8Colores is

signal finMem: std_logic;
signal estadoActual: std_ulogic_vector (2 downto 0);
signal dir, dirAux: std_logic_vector (18 downto 0);
SIGNAL xSync,xDebFallingEdge, xDebRisingEdge, accionOK: std_logic;
SIGNAL startTimer, timerEnd: std_logic;
signal Raux, Gaux, Baux, Rdef, Gdef, Bdef: std_logic_vector (4 downto
0);
constant umbral: std_logic_vector (4 DOWNTO 0) := "01000"; -- 16

begin

      -- Rdef <= (others =>'0') when (Raux < umbral) else  (others
=>'1');
      -- Gdef <= (others =>'0') when (Gaux < umbral) else  (others
=>'1');
      -- Bdef <= (others =>'0') when (Baux < umbral) else  (others
=>'1');

   MaquinaDeEstadosFiltro:
   process (clk, rst, estadoActual)
   begin

   -- Este filtro limita, mediante la utilización de un umbral,
   -- los colores de la imagen original a sólo 8.
   -- Se realiza la comparación de cada componente del pixel con dicho
umbral,
   -- si el valor es mayor se fija al máximo valor del componente, si
es menor
   -- el componente valdrá cero.

     case estadoActual is
        when "000" =>

      -- EStado espera.
      -- Banco derecho de la memoria apagado.
      -- Señal yasta a cero.

          ceDch <= '1';
          yasta <= '0';

        when "001" =>

      -- Leemos los bits de una posición de memoria y cargamos
      -- los correspondientes a cada componente (RGB) en su variable
      -- correspondiente.

          Baux(0) <= dataDch(0);
          Baux(1) <= dataDch(1);
          Baux(2) <= dataDch(2);
          Baux(3) <= dataDch(3);
          Baux(4) <= dataDch(4);
          Gaux(0) <= dataDch(5);
          Gaux(1) <= dataDch(6);
          Gaux(2) <= dataDch(7);
          Gaux(3) <= dataDch(8);
```

```
          Gaux(4) <= dataDch(9);
          Raux(0) <= dataDch(10);
          Raux(1) <= dataDch(11);
          Raux(2) <= dataDch(12);
          Raux(3) <= dataDch(13);
          Raux(4) <= dataDch(14);
          weDch <= '1';
          oeDch <= '0';
          ceDch <= '0';
          dirAux <= dir;
          addressDch <= dir;
          dataDch <= (others =>'Z');

        when "010" =>

      -- Estado Comparación.
      -- En este estado se realiza la comparación de cada uno de los
componentes
      -- del pixel leído y se le asigna el valor correspondiente segun
el resultado.

          ceDch <= '1';

          if (Raux < umbral) then
            Rdef <= (others =>'0');
          else
            Rdef <= (others =>'1');
          end if;

          if (Gaux < umbral) then
            Gdef <= (others =>'0');
          else
            Gdef <= (others =>'1');
          end if;

          if (Baux < umbral) then
            Bdef <= (others =>'0');
          else
            Bdef <= (others =>'1');
          end if;

        when "011" =>

      -- Se escribe en el banco derecho de la memoria el pixel
      -- transformado. Es escribe bit a bit y se va cargando el
      -- valor de cada variable auxiliar, una por componente.

          weDch <= '0';
          oeDch <= '1';
          ceDch <= '0';
          addressDch <= dirAux;
          dataDch(0) <= Bdef(0);
          dataDch(1) <= Bdef(1);
          dataDch(2) <= Bdef(2);
          dataDch(3) <= Bdef(3);
          dataDch(4) <= Bdef(4);
          dataDch(5) <= Gdef(0);
          dataDch(6) <= Gdef(1);
          dataDch(7) <= Gdef(2);
          dataDch(8) <= Gdef(3);
          dataDch(9) <= Gdef(4);
```

```
    dataDch(10) <= Rdef(0);
    dataDch(11) <= Rdef(1);
    dataDch(12) <= Rdef(2);
    dataDch(13) <= Rdef(3);
    dataDch(14) <= Rdef(4);
    dataDch(15) <= '0';
    -- dataDch <= (others =>'1');

   when "100" =>

 -- Estado final.
 -- Se desactiva la memoria y se activa la señal yasta.

    ceDch <= '1';
    yasta <= '1';

   when others =>
    ceDch <= '1';
    yasta <= '0';

end case;

if (rst='0') then
  estadoActual <= "000";

elsif (clk'event and clk='1') then
  if (dir = 524286) then
    finMem <= '1';
  else
    finMem <= '0';
  end if;

  case estadoActual is
    when "000" =>
      if (accionOK = '0') then
        estadoActual <= "001";
        dir <= (others=>'0');
        -- accion = '1';
      else
        estadoActual <= "000";
      end if;

    when "001" =>
      estadoActual <= "010";

    when "010" =>
      estadoActual <= "011";

    when "011" =>
      if (finMem = '1') then
        estadoActual <= "100";
        dir <= (others=>'0');
      else
        estadoActual <= "001";
        dir <= dir + 1;
      end if;

    when "100" =>
      estadoActual <= "100";

    when others =>
```

```
            estadoActual <="000";
        end case;
      end if;
    end process;

  synchronizer:
  PROCESS (rst, clk)
    VARIABLE aux1: std_logic;
  BEGIN
    IF (rst='0') THEN
      aux1 := '1';
      xSync <= '1';
    ELSIF (clk'EVENT AND clk='1') THEN
      xSync <= aux1;
      aux1 := accion;
    END IF;
  END PROCESS synchronizer;

  timer:
  -- espera 50 ms para un reloj a 12.5 MHz
  PROCESS (rst, clk)
    CONSTANT timeOut: std_logic_vector (19 DOWNTO 0) :=
"10011000100101101000";
    VARIABLE count: std_logic_vector (19 DOWNTO 0);
  BEGIN
    IF (count=timeOut) THEN
      timerEnd <= '1';
    ELSE
      timerEnd <= '0';
    END IF;
    IF (rst='0') THEN
      count := timeOut;
    ELSIF (clk'EVENT AND clk='1') THEN
      IF (startTimer='1') THEN
        count := (OTHERS=>'0');
      ELSIF (timerEnd='0') THEN
        count := count + 1;
      END IF;
    END IF;
  END PROCESS timer;

  controller:
  PROCESS (xSync, rst, clk)
    TYPE states IS (waitingPression, pressionDebouncing,
waitingDepression, depressionDebouncing);
    VARIABLE state: states;
  BEGIN
    accionOK <= '1';
    xDebFallingEdge <= '0';
    xDebRisingEdge <= '0';
    startTimer <= '0';
    CASE state IS
      WHEN waitingPression =>
        IF (xSync='0') THEN
          xDebFallingEdge <= '1';
          startTimer <= '1';
        END IF;
      WHEN pressionDebouncing =>
        accionOK <= '0';
      WHEN waitingDepression =>
        accionOK <= '0';
```

```
        IF (xSync='1') THEN
          xDebRisingEdge <= '1';
          startTimer <= '1';
        END IF;
      WHEN depressionDebouncing =>
        NULL;
      END CASE;
    IF (rst='0') THEN
      state := waitingPression;
    ELSIF (clk'EVENT AND clk='1') THEN
      CASE state IS
        WHEN waitingPression =>
          IF (xSync='0') THEN
            state := pressionDebouncing;
          END IF;
        WHEN pressionDebouncing =>
          IF (timerEnd='1') THEN
            state := waitingDepression;
          END IF;
        WHEN waitingDepression =>
          IF (xSync='1') THEN
            state := depressionDebouncing;
          END IF;
        WHEN depressionDebouncing =>
          IF (timerEnd='1') THEN
            state := waitingPression;
          END IF;
      END CASE;
    END IF;
  END PROCESS controller;

end filtro8Colores_arch;
```

C.6.2 Filtro de binarización : filbin.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity filtroBinarizar is
    port (
        rst: in STD_LOGIC;  -- SW 4
        clk: in STD_LOGIC;
        accion: in STD_LOGIC; -- SW 3
        yasta: out STD_LOGIC;
        addressDch: out STD_LOGIC_VECTOR (18 downto 0);
        dataDch: inout STD_ULOGIC_VECTOR (15 downto 0);
        ceDch: out STD_LOGIC;
        oeDch: out STD_LOGIC;
        weDch: out STD_LOGIC
        );
end filtroBinarizar;

architecture filtroBinarizar_arch of filtroBinarizar is

signal finMem: std_logic;
signal estadoActual: std_ulogic_vector (2 downto 0);
signal dir: std_logic_vector (18 downto 0);
SIGNAL xSync,xDebFallingEdge, xDebRisingEdge, accionOK: std_logic;
SIGNAL startTimer, timerEnd: std_logic;
```

```
signal Raux, Gaux, Baux, RGBdef, RGBaux: std_logic_vector (4 downto
0);
constant umbral: std_logic_vector (4 DOWNTO 0) := "01000"; -- 16

begin

   RGBdef <= (others =>'0') when (RGBaux < umbral) else  (others
=>'1');

   MaquinaDeEstadosFiltro:
   process (clk, rst, estadoActual)
   begin

     case estadoActual is
        when "000" =>
          ceDch <= '1';
          yasta <= '0';

        when "001" =>
          Baux(0) <= dataDch(3);
          Baux(1) <= dataDch(4);
          Baux(2) <= '0';
          Baux(3) <= '0';
          Baux(4) <= '0';
          Gaux(0) <= dataDch(6);
          Gaux(1) <= dataDch(7);
          Gaux(2) <= dataDch(8);
          Gaux(3) <= dataDch(9);
          Gaux(4) <= '0';
          Raux(0) <= dataDch(12);
          Raux(1) <= dataDch(13);
          Raux(2) <= dataDch(14);
          Raux(3) <= '0';
          Raux(4) <= '0';
          weDch <= '1';
          oeDch <= '0';
          ceDch <= '0';
          addressDch <= dir;
          dataDch <= (others =>'Z');

        when "010" =>
          ceDch <= '1';
          RGBaux <= Raux + Gaux + Baux;

        when "011" =>
          weDch <= '0';
          oeDch <= '1';
          ceDch <= '0';
          addressDch <= dir;
          dataDch(0) <= RGBdef(0);
          dataDch(1) <= RGBdef(1);
          dataDch(2) <= RGBdef(2);
          dataDch(3) <= RGBdef(3);
          dataDch(4) <= RGBdef(4);
          dataDch(5) <= RGBdef(0);
          dataDch(6) <= RGBdef(1);
          dataDch(7) <= RGBdef(2);
          dataDch(8) <= RGBdef(3);
          dataDch(9) <= RGBdef(4);
          dataDch(10) <= RGBdef(0);
          dataDch(11) <= RGBdef(1);
```

```
        dataDch(12) <= RGBdef(2);
        dataDch(13) <= RGBdef(3);
        dataDch(14) <= RGBdef(4);
        dataDch(15) <= '0';
        -- dataDch <= (others =>'1');

       when "100" =>
        ceDch <= '1';
        yasta <= '1';

       when others =>
        ceDch <= '1';
        yasta <= '0';

    end case;

    if (rst='0') then
      estadoActual <= "000";

    elsif (clk'event and clk='1') then
      if (dir = 524286) then
        finMem <= '1';
      else
        finMem <= '0';
      end if;

      case estadoActual is
        when "000" =>
          if (accionOK = '0') then
            estadoActual <= "001";
            dir <= (others=>'0');
            -- accion = '1';
          else
            estadoActual <= "000";
          end if;

        when "001" =>
          estadoActual <= "010";

        when "010" =>
          estadoActual <= "011";

        when "011" =>
          if (finMem = '1') then
            estadoActual <= "100";
            dir <= (others=>'0');
          else
            estadoActual <= "001";
            dir <= dir + 1;
          end if;

        when "100" =>
          estadoActual <= "100";

        when others =>
          estadoActual <="000";
      end case;
    end if;
  end process;

synchronizer:
```

```
  PROCESS (rst, clk)
    VARIABLE aux1: std_logic;
  BEGIN
    IF (rst='0') THEN
      aux1 := '1';
      xSync <= '1';
    ELSIF (clk'EVENT AND clk='1') THEN
      xSync <= aux1;
      aux1 := accion;
    END IF;
  END PROCESS synchronizer;

  timer:
  -- espera 50 ms para un reloj a 12.5 MHz
  PROCESS (rst, clk)
    CONSTANT timeOut: std_logic_vector (19 DOWNTO 0) :=
"10011000100101101000";
    VARIABLE count: std_logic_vector (19 DOWNTO 0);
  BEGIN
    IF (count=timeOut) THEN
      timerEnd <= '1';
    ELSE
      timerEnd <= '0';
    END IF;
    IF (rst='0') THEN
      count := timeOut;
    ELSIF (clk'EVENT AND clk='1') THEN
      IF (startTimer='1') THEN
        count := (OTHERS=>'0');
      ELSIF (timerEnd='0') THEN
        count := count + 1;
      END IF;
    END IF;
  END PROCESS timer;

  controller:
  PROCESS (xSync, rst, clk)
    TYPE states IS (waitingPression, pressionDebouncing,
waitingDepression, depressionDebouncing);
    VARIABLE state: states;
  BEGIN
    accionOK <= '1';
    xDebFallingEdge <= '0';
    xDebRisingEdge <= '0';
    startTimer <= '0';
    CASE state IS
      WHEN waitingPression =>
        IF (xSync='0') THEN
          xDebFallingEdge <= '1';
          startTimer <= '1';
        END IF;
      WHEN pressionDebouncing =>
        accionOK <= '0';
      WHEN waitingDepression =>
        accionOK <= '0';
        IF (xSync='1') THEN
          xDebRisingEdge <= '1';
          startTimer <= '1';
        END IF;
      WHEN depressionDebouncing =>
        NULL;
```

```
      END CASE;
    IF (rst='0') THEN
      state := waitingPression;
    ELSIF (clk'EVENT AND clk='1') THEN
      CASE state IS
        WHEN waitingPression =>
          IF (xSync='0') THEN
            state := pressionDebouncing;
          END IF;
        WHEN pressionDebouncing =>
          IF (timerEnd='1') THEN
            state := waitingDepression;
          END IF;
        WHEN waitingDepression =>
          IF (xSync='1') THEN
            state := depressionDebouncing;
          END IF;
        WHEN depressionDebouncing =>
          IF (timerEnd='1') THEN
            state := waitingPression;
          END IF;
      END CASE;
    END IF;
  END PROCESS controller;

end filtroBinarizar_arch;
```

C.6.3 Filtro de escala de grises: filbyn.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity filtroBlancoNegro is
    port (
        rst: in STD_LOGIC;  -- SW 4
        clk: in STD_LOGIC;
        accion: in STD_LOGIC; -- SW 3
        yasta: out STD_LOGIC;
        addressDch: out STD_LOGIC_VECTOR (18 downto 0);
        dataDch: inout STD_ULOGIC_VECTOR (15 downto 0);
        ceDch: out STD_LOGIC;
        oeDch: out STD_LOGIC;
        weDch: out STD_LOGIC
        );
end filtroBlancoNegro;

architecture filtroBlancoNegro_arch of filtroBlancoNegro is

signal finMem: std_logic;
signal estadoActual: std_ulogic_vector (2 downto 0);
signal dir: std_logic_vector (18 downto 0);
SIGNAL xSync,xDebFallingEdge, xDebRisingEdge, accionOK: std_logic;
SIGNAL startTimer, timerEnd: std_logic;
signal Raux, Gaux, Baux, RGBdef: std_logic_vector (4 downto 0);

begin

-- Salida = 0,299 * Rojo + 0,587 *  Verde + 0,114 * Azul
-- 299->0100101011
```

```
-- 587->1001001011
-- 114->0001110010
--1000->1111101000

-- Salida = 0,256 * Rojo + 0,512 *  Verde + 0,128 * Azul
-- 512->1000000000 -> desplazar 1 a la dch
-- 256->0100000000 -> desplazar 2 a la dch
-- 128->0010000000 -> desplazar 3 a la dch

   MaquinaDeEstadosFiltro:
   process (clk, rst, estadoActual)
   begin

     case estadoActual is

      -- Espera.
      -- Ambas memorias apagadas e inicialización de la señal yasta.

        when "000" =>
          ceDch <= '1';
          yasta <= '0';

        when "001" =>

      -- Se lee la información del banco derecho de la memoria y se
carga cada
      -- componente del pixel leído en la señal auxiliar
correspondiente.
      -- Los bits se cargan ya desplazadas las posiciones necesarias.

          Baux(0) <= dataDch(3);
          Baux(1) <= dataDch(4);
          Baux(2) <= '0';
          Baux(3) <= '0';
          Baux(4) <= '0';
          Gaux(0) <= dataDch(6);
          Gaux(1) <= dataDch(7);
          Gaux(2) <= dataDch(8);
          Gaux(3) <= dataDch(9);
          Gaux(4) <= '0';
          Raux(0) <= dataDch(12);
          Raux(1) <= dataDch(13);
          Raux(2) <= dataDch(14);
          Raux(3) <= '0';
          Raux(4) <= '0';
          weDch <= '1';
          oeDch <= '0';
          ceDch <= '0';
          addressDch <= dir;
          dataDch <= (others =>'Z');

        when "010" =>

        -- Estado operación.
        -- En este estado se realiza la operación de suma de los
componentes auxiliares
        -- del estado anterior. Para obtener el pixel definitivo.

          ceDch <= '1';
          RGBdef <= Raux + Gaux + Baux;
```

```
        when "011" =>

      -- Se carga en el banco derecho de la memoria el valor de la
señal RGBdef.
      -- Esta señal contiene el valor del pixel una vez ejecutado
sobre el mismo la
      -- fórmula de paso a Blanco y Negro.
      -- Este pixel debe ser cargado en la misma posición de la que
      -- fue leído en el estado 001. Esta posición está guardada en
dir.

          weDch <= '0';
          oeDch <= '1';
          ceDch <= '0';
          addressDch <= dir;
          dataDch(0) <= RGBdef(0);
          dataDch(1) <= RGBdef(1);
          dataDch(2) <= RGBdef(2);
          dataDch(3) <= RGBdef(3);
          dataDch(4) <= RGBdef(4);
          dataDch(5) <= RGBdef(0);
          dataDch(6) <= RGBdef(1);
          dataDch(7) <= RGBdef(2);
          dataDch(8) <= RGBdef(3);
          dataDch(9) <= RGBdef(4);
          dataDch(10) <= RGBdef(0);
          dataDch(11) <= RGBdef(1);
          dataDch(12) <= RGBdef(2);
          dataDch(13) <= RGBdef(3);
          dataDch(14) <= RGBdef(4);
          dataDch(15) <= '0';

         when "100" =>

        -- Estado final.
        -- Se deshabilita la memoria utilizada, el banco derecho.
        -- Y se activa la señal yasta.

          ceDch <= '1';
          yasta <= '1';

         when others =>
          ceDch <= '1';
          yasta <= '0';

      end case;

      if (rst='0') then
        estadoActual <= "000";

      elsif (clk'event and clk='1') then
        if (dir = 524286) then
          finMem <= '1';
        else
          finMem <= '0';
        end if;

        case estadoActual is
          when "000" =>
            if (accionOK = '0') then
              estadoActual <= "001";
```

```
              dir <= (others=>'0');
              -- accion = '1';
            else
              estadoActual <= "000";
            end if;

          when "001" =>
            estadoActual <= "010";

          when "010" =>
            estadoActual <= "011";

          when "011" =>
            if (finMem = '1') then
              estadoActual <= "100";
              dir <= (others=>'0');
            else
              estadoActual <= "001";
              dir <= dir + 1;
            end if;

          when "100" =>
            estadoActual <= "100";

          when others =>
            estadoActual <="000";
        end case;
      end if;
    end process;

  synchronizer:
  PROCESS (rst, clk)
    VARIABLE aux1: std_logic;
  BEGIN
    IF (rst='0') THEN
      aux1 := '1';
      xSync <= '1';
    ELSIF (clk'EVENT AND clk='1') THEN
      xSync <= aux1;
      aux1 := accion;
    END IF;
  END PROCESS synchronizer;

  timer:
  -- espera 50 ms para un reloj a 12.5 MHz
  PROCESS (rst, clk)
    CONSTANT timeOut: std_logic_vector (19 DOWNTO 0) :=
"10011000100101101000";
    VARIABLE count: std_logic_vector (19 DOWNTO 0);
  BEGIN
    IF (count=timeOut) THEN
      timerEnd <= '1';
    ELSE
      timerEnd <= '0';
    END IF;
    IF (rst='0') THEN
      count := timeOut;
    ELSIF (clk'EVENT AND clk='1') THEN
      IF (startTimer='1') THEN
        count := (OTHERS=>'0');
      ELSIF (timerEnd='0') THEN
```

```
        count := count + 1;
      END IF;
    END IF;
  END PROCESS timer;

  controller:
  PROCESS (xSync, rst, clk)
    TYPE states IS (waitingPression, pressionDebouncing,
waitingDepression, depressionDebouncing);
    VARIABLE state: states;
  BEGIN
    accionOK <= '1';
    xDebFallingEdge <= '0';
    xDebRisingEdge <= '0';
    startTimer <= '0';
    CASE state IS
      WHEN waitingPression =>
        IF (xSync='0') THEN
          xDebFallingEdge <= '1';
          startTimer <= '1';
        END IF;
      WHEN pressionDebouncing =>
        accionOK <= '0';
      WHEN waitingDepression =>
        accionOK <= '0';
        IF (xSync='1') THEN
          xDebRisingEdge <= '1';
          startTimer <= '1';
        END IF;
      WHEN depressionDebouncing =>
        NULL;
      END CASE;
    IF (rst='0') THEN
      state := waitingPression;
    ELSIF (clk'EVENT AND clk='1') THEN
      CASE state IS
        WHEN waitingPression =>
          IF (xSync='0') THEN
            state := pressionDebouncing;
          END IF;
        WHEN pressionDebouncing =>
          IF (timerEnd='1') THEN
            state := waitingDepression;
          END IF;
        WHEN waitingDepression =>
          IF (xSync='1') THEN
            state := depressionDebouncing;
          END IF;
        WHEN depressionDebouncing =>
          IF (timerEnd='1') THEN
            state := waitingPression;
          END IF;
      END CASE;
    END IF;
  END PROCESS controller;

end filtroBlancoNegro_arch;
```

C.6.4 Filtro de negativización: filneg.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity filtroNegativizado is
    port (
        rst: in STD_LOGIC;  -- SW 4
        clk: in STD_LOGIC;
        accion: in STD_LOGIC;
        yasta: out STD_LOGIC;
        addressIzq: out STD_LOGIC_VECTOR (18 downto 0);
        addressDch: out STD_LOGIC_VECTOR (18 downto 0);
        dataIzq: inout STD_ULOGIC_VECTOR (15 downto 0);
        dataDch: inout STD_ULOGIC_VECTOR (15 downto 0);
        ceIzq: out STD_LOGIC;
        oeIzq: out STD_LOGIC;
        weIzq: out STD_LOGIC;
        ceDch: out STD_LOGIC;
        oeDch: out STD_LOGIC;
        weDch: out STD_LOGIC
        );
end filtroNegativizado;

architecture filtroNegativizado_arch of filtroNegativizado is

signal finMem: std_logic;
signal estadoActual: std_ulogic_vector (1 downto 0);
signal dir: std_logic_vector (18 downto 0);
SIGNAL xSync,xDebFallingEdge, xDebRisingEdge, accionOK: std_logic;
SIGNAL startTimer, timerEnd: std_logic;

begin

   MaquinaDeEstadosFiltro:
   process (clk, rst, estadoActual)
   begin

     case estadoActual is

        when "00" =>

        -- Espera.
        -- Ambas memorias apagadas ya que en este estado no se
utilizan.
        -- yasta es la señal que indica el final en la ejecución del
filtro.

          ceIzq <= '1';
          ceDch <= '1';
          yasta <= '0';


        when "01" =>

        -- Lectura del banco derecho y negación de los valores.
        -- Carga en el banco izquierdo.
        -- Negamos bit a bit del valor leido de la memoria, aunque se
puede
        -- hacer sin ningún problema con los 15 bits directamente
(operación comentada).
```

```
          weIzq <= '0';
          oeIzq <= '1';
          ceIzq <= '0';
          addressIzq <= dir;
          -- dataIzq <= not dataDch;
          dataIzq(0) <= not dataDch(0);
          dataIzq(1) <= not dataDch(1);
          dataIzq(2) <= not dataDch(2);
          dataIzq(3) <= not dataDch(3);
          dataIzq(4) <= not dataDch(4);
          dataIzq(5) <= not dataDch(5);
          dataIzq(6) <= not dataDch(6);
          dataIzq(7) <= not dataDch(7);
          dataIzq(8) <= not dataDch(8);
          dataIzq(9) <= not dataDch(9);
          dataIzq(10) <= not dataDch(10);
          dataIzq(11) <= not dataDch(11);
          dataIzq(12) <= not dataDch(12);
          dataIzq(13) <= not dataDch(13);
          dataIzq(14) <= not dataDch(14);
          dataIzq(15) <= not dataDch(15);
          weDch <= '1';
          oeDch <= '0';
          ceDch <= '0';
          addressDch <= dir;
          dataDch <= (others =>'Z');

        when "10" =>

        -- Lectura del banco izquierdo y carga en el banco derecho.

          weIzq <= '1';
          oeIzq <= '0';
          ceIzq <= '0';
          addressIzq <= dir;
          dataIzq <= (others =>'Z');
          weDch <= '0';
          oeDch <= '1';
          ceDch <= '0';
          addressDch <= dir;
          dataDch <= dataIzq;

         when "11" =>

       -- Estado final.
         -- Se apagan las memorias, que ya no se van a utilizar.
       -- Se activa la señal yasta para indicar que el filtro ha
terminado.

          ceIzq <= '1';
          ceDch <= '1';
          yasta <= '1';

         when others =>
          ceIzq <= '1';
          ceDch <= '1';
          yasta <= '0';

      end case;
```

```
    if (rst='0') then
      estadoActual <= "00";

    elsif (clk'event and clk='1') then
      if (dir = 524286) then
        finMem <= '1';
      else
        finMem <= '0';
      end if;

      case estadoActual is
        when "00" =>
          if (accionOK = '0') then
            estadoActual <= "01";
          else
            estadoActual <= "00";
          end if;

        when "01" =>
          if (finMem = '1') then
            estadoActual <= "10";
            dir <= (others=>'0');
          else
            estadoActual <= "01";
            dir <= dir + 1;
          end if;

        when "10" =>
          if (finMem = '1') then
            estadoActual <= "11";
            dir <= (others=>'0');
          else
            estadoActual <= "10";
            dir <= dir + 1;
          end if;

        when "11" =>
          estadoActual <= "11";

        when others =>
          estadoActual <="00";
      end case;
    end if;
  end process;

synchronizer:
PROCESS (rst, clk)
  VARIABLE aux1: std_logic;
BEGIN
  IF (rst='0') THEN
    aux1 := '1';
    xSync <= '1';
  ELSIF (clk'EVENT AND clk='1') THEN
    xSync <= aux1;
    aux1 := accion;
  END IF;
END PROCESS synchronizer;

timer:
-- espera 50 ms para un reloj a 12.5 MHz
PROCESS (rst, clk)
```

```
    CONSTANT timeOut: std_logic_vector (19 DOWNTO 0) :=
"10011000100101101000";
    VARIABLE count: std_logic_vector (19 DOWNTO 0);
  BEGIN
    IF (count=timeOut) THEN
      timerEnd <= '1';
    ELSE
      timerEnd <= '0';
    END IF;
    IF (rst='0') THEN
      count := timeOut;
    ELSIF (clk'EVENT AND clk='1') THEN
      IF (startTimer='1') THEN
        count := (OTHERS=>'0');
      ELSIF (timerEnd='0') THEN
        count := count + 1;
      END IF;
    END IF;
  END PROCESS timer;

  controller:
  PROCESS (xSync, rst, clk)
    TYPE states IS (waitingPression, pressionDebouncing,
waitingDepression, depressionDebouncing);
    VARIABLE state: states;
  BEGIN
    accionOK <= '1';
    xDebFallingEdge <= '0';
    xDebRisingEdge <= '0';
    startTimer <= '0';
    CASE state IS
      WHEN waitingPression =>
        IF (xSync='0') THEN
          xDebFallingEdge <= '1';
          startTimer <= '1';
        END IF;
      WHEN pressionDebouncing =>
        accionOK <= '0';
      WHEN waitingDepression =>
        accionOK <= '0';
        IF (xSync='1') THEN
          xDebRisingEdge <= '1';
          startTimer <= '1';
        END IF;
      WHEN depressionDebouncing =>
        NULL;
      END CASE;
    IF (rst='0') THEN
      state := waitingPression;
    ELSIF (clk'EVENT AND clk='1') THEN
      CASE state IS
        WHEN waitingPression =>
          IF (xSync='0') THEN
            state := pressionDebouncing;
          END IF;
        WHEN pressionDebouncing =>
          IF (timerEnd='1') THEN
            state := waitingDepression;
          END IF;
        WHEN waitingDepression =>
          IF (xSync='1') THEN
```

```
            state := depressionDebouncing;
          END IF;
        WHEN depressionDebouncing =>
          IF (timerEnd='1') THEN
            state := waitingPression;
          END IF;
      END CASE;
    END IF;
  END PROCESS controller;

end filtroNegativizado_arch;
```

C.6.5 Filtro de suavizado: filsua.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity filtroSuavizado is
    port (
        -- Pelea personal, los 7 segmentos:
        -- segmentos: STD_ULOGIC_VECTOR (6 downto 0);
        rst: in STD_LOGIC;  -- SW 4
        clk: in STD_LOGIC;
        accion: in STD_LOGIC; -- SW 3
        yasta: out STD_LOGIC;
        addressIzq: out STD_LOGIC_VECTOR (18 downto 0);
        addressDch: out STD_LOGIC_VECTOR (18 downto 0);
        dataIzq: inout STD_LOGIC_VECTOR (15 downto 0);
        dataDch: inout STD_LOGIC_VECTOR (15 downto 0);
        ceIzq: out STD_LOGIC;
        oeIzq: out STD_LOGIC;
        weIzq: out STD_LOGIC;
        ceDch: out STD_LOGIC;
        oeDch: out STD_LOGIC;
        weDch: out STD_LOGIC
        );
end filtroSuavizado;

architecture filtroSuavizado_arch of filtroSuavizado is

signal finMem, finLin: std_logic;
signal estadoActual: std_ulogic_vector (3 downto 0);
signal numLin: std_logic_vector (9 downto 0);
signal dir: std_logic_vector (18 downto 0);
signal aux11, aux12, aux13, aux21, aux22, aux23, aux31, aux32, aux33:
std_logic_vector (15 downto 0);
signal Raux, Gaux, Baux: std_logic_vector (4 downto 0);
signal Raux1, Gaux1, Baux1: std_logic_vector (4 downto 0);
signal Raux2, Gaux2, Baux2: std_logic_vector (4 downto 0);
signal Raux3, Gaux3, Baux3: std_logic_vector (4 downto 0);
--signal Raux11, Raux12, Raux13, Raux21, Raux22, Raux23, Raux31,
Raux32, Raux33: std_logic_vector (4 downto 0);
--signal Gaux11, Gaux12, Gaux13, Gaux21, Gaux22, Gaux23, Gaux31,
Gaux32, Gaux33: std_logic_vector (4 downto 0);
--signal Baux11, Baux12, Baux13, Baux21, Baux22, Baux23, Baux31,
Baux32, Baux33: std_logic_vector (4 downto 0);
signal Rdef11, Rdef12, Rdef13, Rdef21, Rdef22, Rdef23, Rdef31, Rdef32,
Rdef33: std_logic_vector (4 downto 0);
```

```
signal Gdef11, Gdef12, Gdef13, Gdef21, Gdef22, Gdef23, Gdef31, Gdef32,
Gdef33: std_logic_vector (4 downto 0);
signal Bdef11, Bdef12, Bdef13, Bdef21, Bdef22, Bdef23, Bdef31, Bdef32,
Bdef33: std_logic_vector (4 downto 0);
SIGNAL xSync,xDebFallingEdge, xDebRisingEdge, accionOK: std_logic;
SIGNAL startTimer, timerEnd: std_logic;

begin

-- Visualizar el estado:

--   segmentos <= '0011011' when (estadoActual = 0); -- fin => F
--   segmentos <= '1010010' when (estadoActual = 1); -- leyendo => L
--   segmentos <= '1011011' when (estadoActual = 2); -- escribiendo =>
E

--    Raux <= aux11(14 downto 10) + aux12(14 downto 10) + aux13(14
downto 10) + aux21(14 downto 10) + aux22(14 downto 10) + aux23(14
downto 10) + aux31(14 downto 10) + aux32(14 downto 10) + aux33(14
downto 10);
--    Gaux <= aux11(9 downto 5) + aux12(9 downto 5) + aux13(9 downto
5) + aux21(9 downto 5) + aux22(9 downto 5) + aux23(9 downto 5) +
aux31(9 downto 5) + aux32(9 downto 5) + aux33(9 downto 5);
--    Baux <= aux11(4 downto 0) + aux12(4 downto 0) + aux13(4 downto
0) + aux21(4 downto 0) + aux22(4 downto 0) + aux23(4 downto 0) +
aux31(4 downto 0) + aux32(4 downto 0) + aux33(4 downto 0);

   MaquinaDeEstadosFiltro:
   process (clk, rst, estadoActual) -- accion? finMem?
   begin
     case estadoActual is
        when "0000" =>

      -- Estado de espera.
      -- Ambos bancos de memoria apagados y la señal yasta a cero.

          ceIzq <= '1';
          ceDch <= '1';
          yasta <= '0';

        when "0001" =>

      -- Inicializacion 11
      -- Banco Izquierdo de la memoria apagado, no se usa.
      -- Pixel arriba a la izq del todo

          ceIzq <= '1';
          weDch <= '1';
          oeDch <= '0';
          ceDch <= '0';
          addressDch <= (others =>'0');
          dataDch <= (others =>'Z');
          aux12 <= dataDch;
          --Raux12 <= dataDch (14 downto 10);
          --Gaux12 <= dataDch (9 downto 5);
          --Baux12 <= dataDch (4 downto 0);

        when "0010" =>

      -- Inicializacion 12
      -- Mem izq apagada, no se usa.
```

```
-- Leemos el pixel arriba del centro del todo, de la dirección 1
-- de memoria.

    ceIzq <= '1';
    weDch <= '1';
    oeDch <= '0';
    ceDch <= '0';
    addressDch <= ("00000000000000000001");
    dataDch <= (others =>'Z');
    aux13 <= dataDch;
    --Raux13 <= dataDch (14 downto 10);
    --Gaux13 <= dataDch (9 downto 5);
    --Baux13 <= dataDch (4 downto 0);

  when "0011" =>

-- Inicializacion 21
-- Mem izq apagada, no se usa.
-- 1024 (Num de Columnas) (Pixel medio a la izq del todo)

    ceIzq <= '1';
    weDch <= '1';
    oeDch <= '0';
    ceDch <= '0';
    addressDch <= ("00000000010000000000");
    dataDch <= (others =>'Z');
    aux22 <= dataDch;
    --Raux22 <= dataDch (14 downto 10);
    --Gaux22 <= dataDch (9 downto 5);
    --Baux22 <= dataDch (4 downto 0);

  when "0100" =>

-- Inicializacion 22
-- Mem izq apagada, no se usa.
-- 1025 (Num de Columnas +1) (Pixel medio del centro del todo)

    ceIzq <= '1';
    weDch <= '1';
    oeDch <= '0';
    ceDch <= '0';
    addressDch <= ("00000000010000000001");
    dataDch <= (others =>'Z');
    aux23 <= dataDch;
    --Raux23 <= dataDch (14 downto 10);
    --Gaux23 <= dataDch (9 downto 5);
    --Baux23 <= dataDch (4 downto 0);

  when "0101" =>

-- Inicializacion 31
-- Mem izq apagada, no se usa.
-- 2048 (Num de Columnas*2)  (Pixel abajo a la izq del todo)

    ceIzq <= '1';
    weDch <= '1';
    oeDch <= '0';
    ceDch <= '0';
    addressDch <= ("00000001000000000000");
    dataDch <= (others =>'Z');
    aux32 <= dataDch;
```

```
          --Raux32 <= dataDch (14 downto 10);
          --Gaux32 <= dataDch (9 downto 5);
          --Baux32 <= dataDch (4 downto 0);

        when "0110" =>

      -- Inicializacion 32
      -- Mem izq apagada, no se usa.
      -- 2049 (Num de Columnas*2) +1  (Pixel abajo del centro del
todo)

          ceIzq <= '1';
          weDch <= '1';
          oeDch <= '0';
          ceDch <= '0';
          addressDch <= ("00000001000000000001");
          dataDch <= (others =>'Z');
          aux33 <= dataDch;
          --Raux33 <= dataDch (14 downto 10);
          --Gaux33 <= dataDch (9 downto 5);
          --Baux33 <= dataDch (4 downto 0);

        when "0111" =>

      -- Desplaza y actualiza 1
      -- El banco izquierdo de la memoria se mantiene apagado.
      -- Desplazamos los valores de la primera fila de la matriz y
cargamos
      -- el valor de la posición a13

          ceIzq <= '1';
          weDch <= '1';
          oeDch <= '0';
          ceDch <= '0';
          dataDch <= (others =>'Z');
          addressDch <= dir - 1023;
          aux11 <= aux12;
          aux12 <= aux13;
          aux13 <= dataDch;
          --Raux11 <= Raux12;
          --Gaux11 <= Gaux12;
          --Baux11 <= Baux12;
          --Raux12 <= Raux13;
          --Gaux12 <= Gaux13;
          --Baux12 <= Baux13;
          --Raux13 <= dataDch (14 downto 10);
          --Gaux13 <= dataDch (9 downto 5);
          --Baux13 <= dataDch (4 downto 0);

        when "1000" =>

      -- Desplaza y actualiza 2
      -- El banco izquierdo de la memoria se mantiene apagado.
      -- Desplazamos los valores de la segunda fila de la matriz y
cargamos
      -- el valor de la posición a23

          ceIzq <= '1';
          weDch <= '1';
          oeDch <= '0';
          ceDch <= '0';
```

```
            dataDch <= (others =>'Z');
            addressDch <= dir + 1;
            aux21 <= aux22;
            aux22 <= aux23;
            aux23 <= dataDch;
            --Raux21 <= Raux22;
            --Gaux21 <= Gaux22;
            --Baux21 <= Baux22;
            --Raux22 <= Raux23;
            --Gaux22 <= Gaux23;
            --Baux22 <= Baux23;
            --Raux23 <= dataDch (14 downto 10);
            --Gaux23 <= dataDch (9 downto 5);
            --Baux23 <= dataDch (4 downto 0);


          when "1001" =>

        -- Desplaza y actualiza 3
        -- La memoria izquierda continua apagada.
        -- Desplazamos los valores de la tercera fila de la matriz y
cargamos
        -- el valor de la posición a33

            ceIzq <= '1'; -- Sigue apagada
            weDch <= '1';
            oeDch <= '0';
            ceDch <= '0';
            dataDch <= (others =>'Z');
            addressDch <= dir + 1025;
            aux31 <= aux32;
            aux32 <= aux33;
            aux33 <= dataDch;
            --Raux31 <= Raux32;
            --Gaux31 <= Gaux32;
            --Baux31 <= Baux32;
            --Raux32 <= Raux33;
            --Gaux32 <= Gaux33;
            --Baux32 <= Baux33;
            --Raux33 <= dataDch (14 downto 10);
            --Gaux33 <= dataDch (9 downto 5);
            --Baux33 <= dataDch (4 downto 0);

          when "1010" =>

        -- Desplaza componentes tres posiciones a la derecha (divide por
8).
        -- Desplazamos cada componente de cada píxel de la matriz por
        -- separado.

            Rdef11 (0) <= aux11 (13);
            Rdef11 (1) <= aux11 (14);
            Rdef11 (2) <= '0';
            Rdef11 (3) <= '0';
            Rdef11 (4) <= '0';
            Rdef12 (0) <= aux12 (13);
            Rdef12 (1) <= aux12 (14);
            Rdef12 (2) <= '0';
            Rdef12 (3) <= '0';
            Rdef12 (4) <= '0';
            Rdef13 (0) <= aux13 (13);
```

```
Rdef13 (1) <= aux13 (14);
Rdef13 (2) <= '0';
Rdef13 (3) <= '0';
Rdef13 (4) <= '0';
Rdef21 (0) <= aux21 (13);
Rdef21 (1) <= aux21 (14);
Rdef21 (2) <= '0';
Rdef21 (3) <= '0';
Rdef21 (4) <= '0';
Rdef22 (0) <= aux22 (13);
Rdef22 (1) <= aux22 (14);
Rdef22 (2) <= '0';
Rdef22 (3) <= '0';
Rdef22 (4) <= '0';
Rdef23 (0) <= aux23 (13);
Rdef23 (1) <= aux23 (14);
Rdef23 (2) <= '0';
Rdef23 (3) <= '0';
Rdef23 (4) <= '0';
Rdef31 (0) <= aux31 (13);
Rdef31 (1) <= aux31 (14);
Rdef31 (2) <= '0';
Rdef31 (3) <= '0';
Rdef31 (4) <= '0';
Rdef32 (0) <= aux32 (13);
Rdef32 (1) <= aux32 (14);
Rdef32 (2) <= '0';
Rdef32 (3) <= '0';
Rdef32 (4) <= '0';
Rdef33 (0) <= aux33 (13);
Rdef33 (1) <= aux33 (14);
Rdef33 (2) <= '0';
Rdef33 (3) <= '0';
Rdef33 (4) <= '0';

Gdef11 (0) <= aux11 (8);
Gdef11 (1) <= aux11 (9);
Gdef11 (2) <= '0';
Gdef11 (3) <= '0';
Gdef11 (4) <= '0';
Gdef12 (0) <= aux12 (8);
Gdef12 (1) <= aux12 (9);
Gdef12 (2) <= '0';
Gdef12 (3) <= '0';
Gdef12 (4) <= '0';
Gdef13 (0) <= aux13 (8);
Gdef13 (1) <= aux13 (9);
Gdef13 (2) <= '0';
Gdef13 (3) <= '0';
Gdef13 (4) <= '0';
Gdef21 (0) <= aux21 (8);
Gdef21 (1) <= aux21 (9);
Gdef21 (2) <= '0';
Gdef21 (3) <= '0';
Gdef21 (4) <= '0';
Gdef22 (0) <= aux22 (8);
Gdef22 (1) <= aux22 (9);
Gdef22 (2) <= '0';
Gdef22 (3) <= '0';
Gdef22 (4) <= '0';
Gdef23 (0) <= aux23 (8);
```

```
Gdef23 (1) <= aux23 (9);
Gdef23 (2) <= '0';
Gdef23 (3) <= '0';
Gdef23 (4) <= '0';
Gdef31 (0) <= aux31 (8);
Gdef31 (1) <= aux31 (9);
Gdef31 (2) <= '0';
Gdef31 (3) <= '0';
Gdef31 (4) <= '0';
Gdef32 (0) <= aux32 (8);
Gdef32 (1) <= aux32 (9);
Gdef32 (2) <= '0';
Gdef32 (3) <= '0';
Gdef32 (4) <= '0';
Gdef33 (0) <= aux33 (8);
Gdef33 (1) <= aux33 (9);
Gdef33 (2) <= '0';
Gdef33 (3) <= '0';
Gdef33 (4) <= '0';

Bdef11 (0) <= aux11 (3);
Bdef11 (1) <= aux11 (4);
Bdef11 (2) <= '0';
Bdef11 (3) <= '0';
Bdef11 (4) <= '0';
Bdef12 (0) <= aux12 (3);
Bdef12 (1) <= aux12 (4);
Bdef12 (2) <= '0';
Bdef12 (3) <= '0';
Bdef12 (4) <= '0';
Bdef13 (0) <= aux13 (3);
Bdef13 (1) <= aux13 (4);
Bdef13 (2) <= '0';
Bdef13 (3) <= '0';
Bdef13 (4) <= '0';
Bdef21 (0) <= aux21 (3);
Bdef21 (1) <= aux21 (4);
Bdef21 (2) <= '0';
Bdef21 (3) <= '0';
Bdef21 (4) <= '0';
Bdef22 (0) <= aux22 (3);
Bdef22 (1) <= aux22 (4);
Bdef22 (2) <= '0';
Bdef22 (3) <= '0';
Bdef22 (4) <= '0';
Bdef23 (0) <= aux23 (3);
Bdef23 (1) <= aux23 (4);
Bdef23 (2) <= '0';
Bdef23 (3) <= '0';
Bdef23 (4) <= '0';
Bdef31 (0) <= aux31 (3);
Bdef31 (1) <= aux31 (4);
Bdef31 (2) <= '0';
Bdef31 (3) <= '0';
Bdef31 (4) <= '0';
Bdef32 (0) <= aux32 (3);
Bdef32 (1) <= aux32 (4);
Bdef32 (2) <= '0';
Bdef32 (3) <= '0';
Bdef32 (4) <= '0';
Bdef33 (0) <= aux33 (3);
```

```
            Bdef33 (1) <= aux33 (4);
            Bdef33 (2) <= '0';
            Bdef33 (3) <= '0';
            Bdef33 (4) <= '0';

          when "1011" =>

        -- Opera 1.
        -- Sumamos los componentes de cada uno de los píxeles de la
matriz.
        -- Sumamos los tres componentes de los píxeles de cada fila.

            Raux1 <= Rdef11 + Rdef12 + Rdef13;
            Gaux1 <= Gdef11 + Gdef12 + Gdef13;
            Baux1 <= Bdef11 + Bdef12 + Bdef13;

            Raux2 <= Rdef21 + Rdef22 + Rdef23;
            Gaux2 <= Gdef21 + Gdef22 + Gdef23;
            Baux2 <= Bdef21 + Bdef22 + Bdef23;

            Raux3 <= Rdef31 + Rdef32 + Rdef33;
            Gaux3 <= Gdef31 + Gdef32 + Gdef33;
            Baux3 <= Bdef31 + Bdef32 + Bdef33;

          when "1100" =>

        -- Opera 2.
        -- Sumamos los tres componentes de cada fila para
        -- dejar una sóla señal por componente.
        -- Estas señales se asignarán directamente a la señal
        -- de datos de la memoria.

            Raux <=  Raux1 + Raux2 + Raux3;
            Gaux <=  Gaux1 + Gaux2 + Gaux3;
            Baux <=  Baux1 + Baux2 + Baux3;

          when "1101" =>

        -- Escritura en el banco izquierdo.
        -- Cargamos bit a bit cada componente del píxel resultante.
        -- Este píxel es cargado en la posición de memoria "dir", que
        -- se corresponde con el píxel cuya convolución estamos
calculando.
        -- El banco derecho de la memoria está apagado, no lo
utilizamos.

            weIzq <= '0';
            oeIzq <= '1';
            ceIzq <= '0';
            dataIzq (0) <= Baux (0);
            dataIzq (1) <= Baux (1);
            dataIzq (2) <= Baux (2);
            dataIzq (3) <= Baux (3);
            dataIzq (4) <= Baux (4);
            dataIzq (5) <= Gaux (0);
            dataIzq (6) <= Gaux (1);
            dataIzq (7) <= Gaux (2);
            dataIzq (8) <= Gaux (3);
            dataIzq (9) <= Gaux (4);
            dataIzq (10) <= Raux (0);
            dataIzq (11) <= Raux (1);
```

```
          dataIzq (12) <= Raux (2);
          dataIzq (13) <= Raux (3);
          dataIzq (14) <= Raux (4);
          dataIzq (15) <= '0';
          -- dataIzq <= not aux22; -- Para probar, se supone que
invertira...
          addressIzq <= dir;
          ceDch <= '1';

        when "1110" =>

      -- Volcamos el contenido del banco izq en el dcho.

          weIzq <= '1';
          oeIzq <= '0';
          ceIzq <= '0';
          addressIzq <= dir;
          dataIzq <= (others =>'Z');
          weDch <= '0';
          oeDch <= '1';
          ceDch <= '0';
          addressDch <= dir;
          dataDch <= dataIzq;

        when "1111" => -- Apagado
          ceIzq <= '1';
          ceDch <= '1';
          yasta <= '1';

        when others =>
          ceIzq <= '1';
          ceDch <= '1';

      end case;

      if (rst='0') then
        estadoActual <= "0000";

      elsif (clk'event and clk='1') then

        if (dir = 524286) then
          finMem <= '1';
        else
          finMem <= '0';
        end if;

        if (dir = 1023) then
          finLin <= '1';
          numLin <= numLin + 1;
        end if;

        case estadoActual is
          when "0000" => -- Apagado
            if (accionOK = '0') then
              estadoActual <= "0001";
              dir <= (others =>'0');
            else
              estadoActual <= "0000";
            end if;

          when "0001" => -- Inicializacion 11
```

```
  estadoActual <= "0010";

when "0010" => -- Inicializacion 12
  estadoActual <= "0011";

when "0011" => -- Inicializacion 21
  estadoActual <= "0100";

when "0100" => -- Inicializacion 22
  estadoActual <= "0101";

when "0101" => -- Inicializacion 31
  estadoActual <= "0110";

when "0110" => -- Inicializacion 32
  estadoActual <= "0111";
  dir <= ("00000000010000000000");

when "0111" => -- Desplaza y actualiza 1
  estadoActual <= "1000";

when "1000" => -- Desplaza y actualiza 2
  estadoActual <= "1001";

when "1001" => -- Desplaza y actualiza 3
  estadoActual <= "1010";

when "1010" => -- Desplaza operandos
  estadoActual <= "1011";

when "1011" => -- Opera
  estadoActual <= "1100";

when "1100" => -- Opera  mas
  estadoActual <= "1101";

when "1101" => -- Guarda en el banco izq
  if (finMem = '1') then
    estadoActual <= "1110";
    dir <= (others =>'0');
    finMem <= '0';
    finLin <= '0';
    numLin <= (others =>'0');
  else
    estadoActual <= "0111";
    dir <= dir + 1;
  end if;

when "1110" => -- Volcamos de Izq a Dch
  if (finMem = '1') then
    estadoActual <= "1111";
    dir <= (others =>'0');
  else
    estadoActual <= "1110";
    dir <= dir + 1;
  end if;

when "1111" => -- Fin
   estadoActual <= "1111";

when others =>
```

```
            estadoActual <="0000";
        end case;
      end if;
    end process;

  synchronizer:
  PROCESS (rst, clk)
    VARIABLE aux1: std_logic;
  BEGIN
    IF (rst='0') THEN
      aux1 := '1';
      xSync <= '1';
    ELSIF (clk'EVENT AND clk='1') THEN
      xSync <= aux1;
      aux1 := accion;
    END IF;
  END PROCESS synchronizer;

  timer:
  -- espera 50 ms para un reloj a 12.5 MHz
  PROCESS (rst, clk)
    CONSTANT timeOut: std_logic_vector (19 DOWNTO 0) :=
"10011000100101101000";
    VARIABLE count: std_logic_vector (19 DOWNTO 0);
  BEGIN
    IF (count=timeOut) THEN
      timerEnd <= '1';
    ELSE
      timerEnd <= '0';
    END IF;
    IF (rst='0') THEN
      count := timeOut;
    ELSIF (clk'EVENT AND clk='1') THEN
      IF (startTimer='1') THEN
        count := (OTHERS=>'0');
      ELSIF (timerEnd='0') THEN
        count := count + 1;
      END IF;
    END IF;
  END PROCESS timer;

  controller:
  PROCESS (xSync, rst, clk)
    TYPE states IS (waitingPression, pressionDebouncing,
waitingDepression, depressionDebouncing);
    VARIABLE state: states;
  BEGIN
    accionOK <= '1';
    xDebFallingEdge <= '0';
    xDebRisingEdge <= '0';
    startTimer <= '0';
    CASE state IS
      WHEN waitingPression =>
        IF (xSync='0') THEN
          xDebFallingEdge <= '1';
          startTimer <= '1';
        END IF;
      WHEN pressionDebouncing =>
        accionOK <= '0';
      WHEN waitingDepression =>
        accionOK <= '0';
```

```
        IF (xSync='1') THEN
          xDebRisingEdge <= '1';
          startTimer <= '1';
        END IF;
      WHEN depressionDebouncing =>
        NULL;
      END CASE;
    IF (rst='0') THEN
      state := waitingPression;
    ELSIF (clk'EVENT AND clk='1') THEN
      CASE state IS
        WHEN waitingPression =>
          IF (xSync='0') THEN
            state := pressionDebouncing;
          END IF;
        WHEN pressionDebouncing =>
          IF (timerEnd='1') THEN
            state := waitingDepression;
          END IF;
        WHEN waitingDepression =>
          IF (xSync='1') THEN
            state := depressionDebouncing;
          END IF;
        WHEN depressionDebouncing =>
          IF (timerEnd='1') THEN
            state := waitingPression;
          END IF;
      END CASE;
    END IF;
  END PROCESS controller;

end filtroSuavizado_arch;
```

C.6.6 Filtro de obtención de bordes: filbor.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity filtroBordes is
    port (
        -- Pelea personal, los 7 segmentos:
        -- segmentos: STD_ULOGIC_VECTOR (6 downto 0);
        rst: in STD_LOGIC;  -- SW 4
        clk: in STD_LOGIC;
        accion: in STD_LOGIC; -- SW 3
        yasta: out STD_LOGIC;
        addressIzq: out STD_LOGIC_VECTOR (18 downto 0);
        addressDch: out STD_LOGIC_VECTOR (18 downto 0);
        dataIzq: inout STD_LOGIC_VECTOR (15 downto 0);
        dataDch: inout STD_LOGIC_VECTOR (15 downto 0);
        ceIzq: out STD_LOGIC;
        oeIzq: out STD_LOGIC;
        weIzq: out STD_LOGIC;
        ceDch: out STD_LOGIC;
        oeDch: out STD_LOGIC;
        weDch: out STD_LOGIC
        );
end filtroBordes;
```

```
architecture filtroBordes_arch of filtroBordes is

signal finMem, finLin: std_logic;
signal estadoActual: std_ulogic_vector (3 downto 0);
signal numLin: std_logic_vector (9 downto 0);
signal dir: std_logic_vector (18 downto 0);
--signal aux11, aux12, aux13, aux21, aux22, aux23, aux31, aux32,
aux33: std_logic_vector (15 downto 0);
signal Raux, Gaux, Baux: std_logic_vector (4 downto 0);
signal Raux1, Gaux1, Baux1: std_logic_vector (4 downto 0);
signal Raux2, Gaux2, Baux2: std_logic_vector (4 downto 0);
signal Raux3, Gaux3, Baux3: std_logic_vector (4 downto 0);
signal Raux11, Raux12, Raux13, Raux21, Raux22, Raux23, Raux31, Raux32,
Raux33: std_logic_vector (4 downto 0);
signal Gaux11, Gaux12, Gaux13, Gaux21, Gaux22, Gaux23, Gaux31, Gaux32,
Gaux33: std_logic_vector (4 downto 0);
signal Baux11, Baux12, Baux13, Baux21, Baux22, Baux23, Baux31, Baux32,
Baux33: std_logic_vector (4 downto 0);
signal Rdef11, Rdef12, Rdef13, Rdef21, Rdef22, Rdef23, Rdef31, Rdef32,
Rdef33: std_logic_vector (4 downto 0);
signal Gdef11, Gdef12, Gdef13, Gdef21, Gdef22, Gdef23, Gdef31, Gdef32,
Gdef33: std_logic_vector (4 downto 0);
signal Bdef11, Bdef12, Bdef13, Bdef21, Bdef22, Bdef23, Bdef31, Bdef32,
Bdef33: std_logic_vector (4 downto 0);
SIGNAL xSync,xDebFallingEdge, xDebRisingEdge, accionOK: std_logic;
SIGNAL startTimer, timerEnd: std_logic;

begin

-- Visualizar el estado:

--   segmentos <= '0011011' when (estadoActual = 0); -- fin => F
--   segmentos <= '1010010' when (estadoActual = 1); -- leyendo => L
--   segmentos <= '1011011' when (estadoActual = 2); -- escribiendo =>
E

--    Raux <= aux11(14 downto 10) + aux12(14 downto 10) + aux13(14
downto 10) + aux21(14 downto 10) + aux22(14 downto 10) + aux23(14
downto 10) + aux31(14 downto 10) + aux32(14 downto 10) + aux33(14
downto 10);
--    Gaux <= aux11(9 downto 5) + aux12(9 downto 5) + aux13(9 downto
5) + aux21(9 downto 5) + aux22(9 downto 5) + aux23(9 downto 5) +
aux31(9 downto 5) + aux32(9 downto 5) + aux33(9 downto 5);
--    Baux <= aux11(4 downto 0) + aux12(4 downto 0) + aux13(4 downto
0) + aux21(4 downto 0) + aux22(4 downto 0) + aux23(4 downto 0) +
aux31(4 downto 0) + aux32(4 downto 0) + aux33(4 downto 0);

   MaquinaDeEstadosFiltro:
   process (clk, rst, estadoActual) -- accion? finMem?
   begin
     case estadoActual is
        when "0000" =>

      -- Estado de espera.
      -- Ambos bancos de memoria apagados y la señal yasta a cero.

          ceIzq <= '1';
          ceDch <= '1';
          yasta <= '0';

        when "0001" =>
```

```
-- Inicializacion 11
-- Banco Izquierdo de la memoria apagado, no se usa.
-- Pixel arriba a la izq del todo

    ceIzq <= '1';
    weDch <= '1';
    oeDch <= '0';
    ceDch <= '0';
    addressDch <= (others =>'0');
    dataDch <= (others =>'Z');
    --aux12 <= dataDch;
    Raux12 <= dataDch (14 downto 10);
    Gaux12 <= dataDch (9 downto 5);
    Baux12 <= dataDch (4 downto 0);

  when "0010" =>

-- Inicializacion 12
-- Mem izq apagada, no se usa.
-- Leemos el pixel arriba del centro del todo, de la dirección 1
-- de memoria.

    ceIzq <= '1';
    weDch <= '1';
    oeDch <= '0';
    ceDch <= '0';
    addressDch <= ("00000000000000000001");
    dataDch <= (others =>'Z');
    --aux13 <= dataDch;
    Raux13 <= dataDch (14 downto 10);
    Gaux13 <= dataDch (9 downto 5);
    Baux13 <= dataDch (4 downto 0);

  when "0011" =>

-- Inicializacion 21
-- Mem izq apagada, no se usa.
-- 1024 (Num de Columnas) (Pixel medio a la izq del todo)

    ceIzq <= '1';
    weDch <= '1';
    oeDch <= '0';
    ceDch <= '0';
    addressDch <= ("00000000100000000000");
    dataDch <= (others =>'Z');
    -- aux22 <= dataDch;
    Raux22 <= dataDch (14 downto 10);
    Gaux22 <= dataDch (9 downto 5);
    Baux22 <= dataDch (4 downto 0);

  when "0100" =>

-- Inicializacion 22
-- Mem izq apagada, no se usa.
-- 1025 (Num de Columnas +1) (Pixel medio del centro del todo)

    ceIzq <= '1';
    weDch <= '1';
    oeDch <= '0';
    ceDch <= '0';
```

```
          addressDch <= ("00000000100000000001");
          dataDch <= (others =>'Z');
          -- aux23 <= dataDch;
          Raux23 <= dataDch (14 downto 10);
          Gaux23 <= dataDch (9 downto 5);
          Baux23 <= dataDch (4 downto 0);

        when "0101" =>

      -- Inicializacion 31
      -- Mem izq apagada, no se usa.
      -- 2048 (Num de Columnas*2)  (Pixel abajo a la izq del todo)

          ceIzq <= '1';
          weDch <= '1';
          oeDch <= '0';
          ceDch <= '0';
          addressDch <= ("00000001000000000000");
          dataDch <= (others =>'Z');
          -- aux32 <= dataDch;
          Raux32 <= dataDch (14 downto 10);
          Gaux32 <= dataDch (9 downto 5);
          Baux32 <= dataDch (4 downto 0);

        when "0110" =>

      -- Inicializacion 32
      -- Mem izq apagada, no se usa.
      -- 2049 (Num de Columnas*2) +1  (Pixel abajo del centro del
todo)

          ceIzq <= '1';
          weDch <= '1';
          oeDch <= '0';
          ceDch <= '0';
          addressDch <= ("00000001000000000001");
          dataDch <= (others =>'Z');
          -- aux33 <= dataDch;
          Raux33 <= dataDch (14 downto 10);
          Gaux33 <= dataDch (9 downto 5);
          Baux33 <= dataDch (4 downto 0);

        when "0111" =>


      -- Desplaza y actualiza 1
      -- El banco izquierdo de la memoria se mantiene apagado.
      -- Desplazamos los valores de la primera fila de la matriz y
cargamos
      -- el valor de la posición a13

          ceIzq <= '1';
          weDch <= '1';
          oeDch <= '0';
          ceDch <= '0';
          dataDch <= (others =>'Z');
          addressDch <= dir - 1023;
          --aux11 <= aux12;
          --aux12 <= aux13;
          --aux13 <= dataDch;
          Raux11 <= Raux12;
```

```
          Gaux11 <= Gaux12;
          Baux11 <= Baux12;
          Raux12 <= Raux13;
          Gaux12 <= Gaux13;
          Baux12 <= Baux13;
          Raux13 <= dataDch (14 downto 10);
          Gaux13 <= dataDch (9 downto 5);
          Baux13 <= dataDch (4 downto 0);

        when "1000" =>

      -- Desplaza y actualiza 2
      -- El banco izquierdo de la memoria se mantiene apagado.
      -- Desplazamos los valores de la segunda fila de la matriz y
cargamos
      -- el valor de la posición a23

          ceIzq <= '1';
          weDch <= '1';
          oeDch <= '0';
          ceDch <= '0';
          dataDch <= (others =>'Z');
          addressDch <= dir + 1;
          --aux21 <= aux22;
          --aux22 <= aux23;
          --aux23 <= dataDch;
          Raux21 <= Raux22;
          Gaux21 <= Gaux22;
          Baux21 <= Baux22;
          Raux22 <= Raux23;
          Gaux22 <= Gaux23;
          Baux22 <= Baux23;
          Raux23 <= dataDch (14 downto 10);
          Gaux23 <= dataDch (9 downto 5);
          Baux23 <= dataDch (4 downto 0);


        when "1001" =>


      -- Desplaza y actualiza 3
      -- La memoria izquierda continua apagada.
      -- Desplazamos los valores de la tercera fila de la matriz y
cargamos
      -- el valor de la posición a33

          ceIzq <= '1';
          weDch <= '1';
          oeDch <= '0';
          ceDch <= '0';
          dataDch <= (others =>'Z');
          addressDch <= dir + 1025;
          --aux31 <= aux32;
          --aux32 <= aux33;
          --aux33 <= dataDch;
          Raux31 <= Raux32;
          Gaux31 <= Gaux32;
          Baux31 <= Baux32;
          Raux32 <= Raux33;
          Gaux32 <= Gaux33;
          Baux32 <= Baux33;
```

```
          Raux33 <= dataDch (14 downto 10);
          Gaux33 <= dataDch (9 downto 5);
          Baux33 <= dataDch (4 downto 0);

        when "1010" =>

      -- Desplazamos tres posiciones a la derecha (multiplicar por
ocho)
      -- los tres componentes del pixel central de la matriz.


          Rdef22(0) <= '0';
          Rdef22(1) <= '0';
          Rdef22(2) <= '0';
          Rdef22(3) <= Raux22(0);
          Rdef22(4) <= Raux22(1);

          Gdef22(0) <= '0';
          Gdef22(1) <= '0';
          Gdef22(2) <= '0';
          Gdef22(3) <= Gaux22(0);
          Gdef22(4) <= Gaux22(1);

          Bdef22(0) <= '0';
          Bdef22(1) <= '0';
          Bdef22(2) <= '0';
          Bdef22(3) <= Baux22(0);
          Bdef22(4) <= Baux22(1);

        when "1011" =>

      -- Sumamos para luego restar
      -- Sumamos los tres componentes de todos los pixeles de la
matriz
      -- por separado.

          Raux1 <= Raux11 + Raux12 + Raux13;
          Gaux1 <= Gaux11 + Gaux12 + Gaux13;
          Baux1 <= Baux11 + Baux12 + Baux13;

          Raux2 <= Raux21 + Raux23;
          Gaux2 <= Gaux21 + Gaux23;
          Baux2 <= Baux21 + Baux23;

          Raux3 <= Raux31 + Raux32 + Raux33;
          Gaux3 <= Gaux31 + Gaux32 + Gaux33;
          Baux3 <= Baux31 + Baux32 + Baux33;

        when "1100" =>

      -- Restamos a cada componente del píxel central, ya
multiplicado,
      -- los resultados de sumar los componentes del resto de los
píxeles.

          Raux <=  Rdef22 - Raux1 - Raux2 - Raux3;
          Gaux <=  Gdef22 - Gaux1 - Gaux2 - Gaux3;
          Baux <=  Bdef22 - Baux1 - Baux2 - Baux3;

        when "1101" =>
```

```
      -- Escribimos en el banco izquierdo de la memoria
      -- el resultado de la convolución del pixel.

          weIzq <= '0';
          oeIzq <= '1';
          ceIzq <= '0';
          dataIzq (0) <= Baux (0);
          dataIzq (1) <= Baux (1);
          dataIzq (2) <= Baux (2);
          dataIzq (3) <= Baux (3);
          dataIzq (4) <= Baux (4);
          dataIzq (5) <= Gaux (0);
          dataIzq (6) <= Gaux (1);
          dataIzq (7) <= Gaux (2);
          dataIzq (8) <= Gaux (3);
          dataIzq (9) <= Gaux (4);
          dataIzq (10) <= Raux (0);
          dataIzq (11) <= Raux (1);
          dataIzq (12) <= Raux (2);
          dataIzq (13) <= Raux (3);
          dataIzq (14) <= Raux (4);
          dataIzq (15) <= '0';
          -- dataIzq <= not aux22; -- Para probar, se supone que
invertira...
          addressIzq <= dir;
          ceDch <= '1'; -- El banco dcho no lo utilizamos

        when "1110" => -- Volcamos el contenido del banco izq en el
dcho.
          weIzq <= '1';
          oeIzq <= '0';
          ceIzq <= '0';
          addressIzq <= dir;
          dataIzq <= (others =>'Z');
          weDch <= '0';
          oeDch <= '1';
          ceDch <= '0';
          addressDch <= dir;
          dataDch <= dataIzq;

        when "1111" => -- Apagado
          ceIzq <= '1';
          ceDch <= '1';
          yasta <= '1';

        when others =>
          ceIzq <= '1';
          ceDch <= '1';

      end case;

      if (rst='0') then
        estadoActual <= "0000";

      elsif (clk'event and clk='1') then

        if (dir = 524286) then
          finMem <= '1';
        else
          finMem <= '0';
        end if;
```

```
if (dir = 1023) then
  finLin <= '1';
  numLin <= numLin + 1;
end if;

case estadoActual is
  when "0000" => -- Apagado
    if (accionOK = '0') then
      estadoActual <= "0001";
      dir <= (others =>'0');
    else
      estadoActual <= "0000";
    end if;

  when "0001" => -- Inicializacion 11
    estadoActual <= "0010";

  when "0010" => -- Inicializacion 12
    estadoActual <= "0011";

  when "0011" => -- Inicializacion 21
    estadoActual <= "0100";

  when "0100" => -- Inicializacion 22
    estadoActual <= "0101";

  when "0101" => -- Inicializacion 31
    estadoActual <= "0110";

  when "0110" => -- Inicializacion 32
    estadoActual <= "0111";
    dir <= ("00000000100000000000");

  when "0111" => -- Desplaza y actualiza 1
    estadoActual <= "1000";

  when "1000" => -- Desplaza y actualiza 2
    estadoActual <= "1001";

  when "1001" => -- Desplaza y actualiza 3
    estadoActual <= "1010";

  when "1010" => -- Desplaza operandos
    estadoActual <= "1011";

  when "1011" => -- Opera
    estadoActual <= "1100";

  when "1100" => -- Opera  mas
    estadoActual <= "1101";

  when "1101" => -- Guarda en el banco izq
    if (finMem = '1') then
      estadoActual <= "1110";
      dir <= (others =>'0');
      finMem <= '0';
      finLin <= '0';
      numLin <= (others =>'0');
    else
      estadoActual <= "0111";
```

```
              dir <= dir + 1;
            end if;

          when "1110" => -- Volcamos de Izq a Dch
            if (finMem = '1') then
              estadoActual <= "1111";
              dir <= (others =>'0');
            else
              estadoActual <= "1110";
              dir <= dir + 1;
            end if;

          when "1111" => -- Fin
             estadoActual <= "1111";

          when others =>
            estadoActual <="0000";
        end case;
      end if;
    end process;

  synchronizer:
  PROCESS (rst, clk)
    VARIABLE aux1: std_logic;
  BEGIN
    IF (rst='0') THEN
      aux1 := '1';
      xSync <= '1';
    ELSIF (clk'EVENT AND clk='1') THEN
      xSync <= aux1;
      aux1 := accion;
    END IF;
  END PROCESS synchronizer;

  timer:
  -- espera 50 ms para un reloj a 12.5 MHz
  PROCESS (rst, clk)
    CONSTANT timeOut: std_logic_vector (19 DOWNTO 0) :=
"10011000100101101000";
    VARIABLE count: std_logic_vector (19 DOWNTO 0);
  BEGIN
    IF (count=timeOut) THEN
      timerEnd <= '1';
    ELSE
      timerEnd <= '0';
    END IF;
    IF (rst='0') THEN
      count := timeOut;
    ELSIF (clk'EVENT AND clk='1') THEN
      IF (startTimer='1') THEN
        count := (OTHERS=>'0');
      ELSIF (timerEnd='0') THEN
        count := count + 1;
      END IF;
    END IF;
  END PROCESS timer;

  controller:
  PROCESS (xSync, rst, clk)
    TYPE states IS (waitingPression, pressionDebouncing,
waitingDepression, depressionDebouncing);
```

```
    VARIABLE state: states;
  BEGIN
    accionOK <= '1';
    xDebFallingEdge <= '0';
    xDebRisingEdge <= '0';
    startTimer <= '0';
    CASE state IS
      WHEN waitingPression =>
        IF (xSync='0') THEN
          xDebFallingEdge <= '1';
          startTimer <= '1';
        END IF;
      WHEN pressionDebouncing =>
        accionOK <= '0';
      WHEN waitingDepression =>
        accionOK <= '0';
        IF (xSync='1') THEN
          xDebRisingEdge <= '1';
          startTimer <= '1';
        END IF;
      WHEN depressionDebouncing =>
        NULL;
      END CASE;
    IF (rst='0') THEN
      state := waitingPression;
    ELSIF (clk'EVENT AND clk='1') THEN
      CASE state IS
        WHEN waitingPression =>
          IF (xSync='0') THEN
            state := pressionDebouncing;
          END IF;
        WHEN pressionDebouncing =>
          IF (timerEnd='1') THEN
            state := waitingDepression;
          END IF;
        WHEN waitingDepression =>
          IF (xSync='1') THEN
            state := depressionDebouncing;
          END IF;
        WHEN depressionDebouncing =>
          IF (timerEnd='1') THEN
            state := waitingPression;
          END IF;
      END CASE;
    END IF;
  END PROCESS controller;

end filtroBordes_arch;
```

C.7. Primer archivo de reconfiguración de la FPGA:

flashCon.vhd

```
--En este VHD se carga el segundo de los archivos, el de la direcc
1000.....
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
```

```
entity flashCon is
    generic (
       ADDR_LEN:positive:=21
    );
    port (
        clk: in STD_LOGIC;

        a: inout STD_LOGIC_VECTOR (ADDR_LEN-1 downto 0);
        ceb : out STD_LOGIC;
        oeb : out STD_LOGIC;
        web : out STD_LOGIC;
        resetb : out STD_LOGIC;

        led1 : out STD_LOGIC_VECTOR (6 downto 0);
        led2 : out STD_LOGIC;

        V_progb : out STD_LOGIC;
        V_cclk : out STD_LOGIC;
        V_csb : out STD_LOGIC;
        V_wrb : out STD_LOGIC;
        V_initB : in STD_LOGIC;
        V_dout : in STD_LOGIC;
        V_done : in STD_LOGIC;
        V_m : out STD_LOGIC_VECTOR (2 downto 0)
    );
end flashCon;

architecture flashCon_arch of flashCon is
   constant LO : std_logic := '0';
   constant HI : std_logic := '1';
   constant CERO : std_logic_vector(6 downto 0) := "1110111";
   constant UNO  : std_logic_vector(6 downto 0) := "0010010";
   constant DOS  : std_logic_vector(6 downto 0) := "1011101";
   constant TRES : std_logic_vector(6 downto 0) := "1011011";
   constant Eli_Reb  : std_logic_vector(6 downto 0) := "0001100";
   constant Esp_Baj  : std_logic_vector(6 downto 0) := "0101111";
   constant OTRO  : std_logic_vector(6 downto 0) := "1110111";

   signal clk_cnt : std_logic_vector(3 downto 0);
   signal cclk,programb,cs : std_logic;
   signal addr, next_addr : std_logic_vector (ADDR_LEN-1 downto 0);
   signal poweron_reset : std_logic;
   signal poweron_cnt : std_logic_vector(19 downto 0);
   signal V_busy,button_b : std_logic;

   type TEstado is (S0, S1, S2, S3, EliminaRebotes1, BajaDipSwitch,
EliminaRebotes2);
   signal estado,sig_estado : TEstado;
   signal comienza,reconf,sel : std_logic;

   signal cuenta300ns : std_logic_vector(4 downto 0);
   signal cuenta100ms : std_logic_vector(19 downto 0);
   signal reset_espera_300ns, reset_espera_100ms : std_logic;
   signal fin_espera_300ns, fin_espera_100ms : std_logic;

begin

   V_busy <= V_dout;

   V_m <= "110";
```

```
--oeb <= LO when (V_done=LO) else 'Z';
--ceb <= LO when (V_done=LO) else 'Z';
--web <= HI when (V_done=LO) else 'Z';
oeb <= LO when (reconf=HI) else 'Z';
ceb <= LO when (reconf=HI) else 'Z';
web <= HI when (reconf=HI) else 'Z';
resetb <= HI ;

process (clk, clk_cnt)
begin
   if (clk'event and clk=HI) then
      clk_cnt <= clk_cnt+1;
   end if;
end process;
cclk <= clk_cnt(3);
V_cclk <= cclk;

process (poweron_cnt, cclk)
begin
   if (cclk'event and cclk=HI) then
      if (poweron_cnt = 0) then
         poweron_reset <= LO;
      else
         poweron_cnt <= poweron_cnt - 1;
         poweron_reset <= HI;
      end if;
   end if;
end process;
led2 <= not(poweron_reset);

--programb <= not(poweron_reset);
--programb <= not(reconf);
V_progb <= programb;

process(V_initb, cclk, programb)
begin
   if (programb=LO) then
      cs <= LO;
   elsif (cclk'event and cclk=HI) then
      cs <= V_initb;
   end if;
end process;

--V_csb <= not (cs) when (V_done=LO) else 'Z';
--V_wrb <= not (cs) when (V_done=LO) else 'Z';
V_csb <= not (cs) when (reconf=HI) else 'Z';
V_wrb <= not (cs) when (reconf=HI) else 'Z';

process(addr, cs, V_initb, V_busy, cclk)
begin
   if (cclk'event and cclk=HI) then
      if ((cs=HI) and (V_initb=HI) and (V_busy=LO)) then
         addr <= addr + 1;
      elsif (programb=LO) then
            addr(19 downto 0) <= (others=>LO) ;
            addr(20) <= HI;
      end if;
   end if;
end process;

comienza <= a(13);
```

```
   --carga del siguiente estado
   --AQUI HAY UN ERROR, NO SIEMPRE QUE BAJE EL DIPSWITCH TENGO QUE
EMPEZAR EN ESTADO 0
   process( cclk,comienza )
   begin
      if cclk'event and cclk = '1' then
         estado <= sig_estado;
      end if;
   end process;

   --maquina de estados que controla todo el comportamiento
   process( estado, poweron_reset, comienza, V_done )
   begin
      --reconf <= LO;
      --sig_estado <= estado;
      case estado is
        --estado inicial
        --sigo en este estado hasta que me llegue una subida del
dipswitch 1
        when S0 =>
            reconf <= LO;
            led1 <= CERO;
            programb <= LO;
            if (poweron_reset = LO) and (comienza = LO) then
               sig_estado <= EliminaRebotes1;
               reset_espera_100ms <= '0';  --comienzo a contar para
llegar a los 300 ns que necesito
                                           --para eliminar rebotes
            else
               sig_estado <= S0;
               reset_espera_100ms <= '1';
            end if;
        --me espero un poco (300ns) para eliminar los rebotes del
dipswitch
        when EliminaRebotes1 =>
            reconf <= LO;
            led1 <= Eli_Reb;
            programb <= LO;
            if ( fin_espera_100ms = HI) then
               sig_estado <= BajaDipSwitch;
            else
               sig_estado <= EliminaRebotes1;
            end if;
        --me quedo en este estado hasta que baje el dipswitch
        when BajaDipSwitch =>
            reconf <= LO;
            led1 <= Esp_Baj;
            programb <= LO;
            if ( comienza = HI) then
               sig_estado <= EliminaRebotes2;
               reset_espera_100ms <= '1';  --comienzo a contar para
llegar a los 300 ns que necesito
                                           --para eliminar rebotes
            else
               sig_estado <= BajaDipSwitch;
            end if;
        --me espero un poco (300ns) para eliminar los rebotes del
dipswitch
        when EliminaRebotes2 =>
            reconf <= LO;
            led1 <= Eli_Reb;
```

```
            programb <= LO;
            reset_espera_100ms <= '0';
            if ( fin_espera_100ms = HI) then
               sig_estado <= S1;
            else
               sig_estado <= EliminaRebotes2;
            end if;
        --ya me ha llegado la segnal de que empiece a reconfigurar,
asi que
        --pido a la FPGA que comience su reconfiguracion, borrandose
        when S1 =>
            reconf <= LO;
            led1 <= UNO;
            programb <= HI;  --se borra
            reset_espera_300ns <= '1';  --reseteo el contador
            sig_estado <= S2;  --paso automaticamente al siguiente
estado
        --espero hasta que este la FPGA ya este borrada y lista para
reconfigurar
        when S2 =>
            reconf <= LO;
            led1 <= UNO;
            programb <= HI;  --mantengo el pulso durante 300 ns para
que se borre
            reset_espera_300ns <= '0';
            if (fin_espera_300ns = HI) then
               sig_estado <= S3;
            else
               sig_estado <= S2;
            end if;
        --ya he recibido la confirmacion de que puedo reconfigurar,
asi que empiezo a hacerlo
        --y ya bajo la segnal que obliga a la FPGA a borrarse. Sigo en
este estado hasta que la FPGA
        --me avise de que ya ha terminado de reconfigurarse
        when S3 =>
            reconf <= HI;
            led1 <= TRES;
            programb <= LO;
            if (V_done = HI) then
               sig_estado <= S0;  --cuando termino vuelvo a la
normalidad y permito nuevas reconfiguraciones
            else
               sig_estado <= S3;
            end if;
         when others =>
            led1 <= OTRO;
      end case;
   end process;

   --a <= addr when (V_done=LO) else (others=>'Z');
   a <= addr when (reconf=HI) else (others=>'Z');

   --para que espere 300ns a 50MHz serian 15 pulsos, le voy a dar 32
pulsos
--   process(cclk, reset_espera_300ns)
--   begin
--      if (reset_espera_300ns = '1') then
--         cuenta300ns <= (others=>'1');
--         fin_espera_300ns <= '0';
--      elsif (cclk'event and cclk=HI) then
```

```
--         if (cuenta300ns = 0) then
--            fin_espera_300ns <= '1';
--         else
--            cuenta300ns <= cuenta300ns - 1;
--            fin_espera_300ns <= '0';
--         end if;
--      end if;
--   end process;

   --para que espere 100ms a 50MHz serian 5.000.000 pulsos, 24 bits
   process(clk, reset_espera_100ms)
   begin
      if (reset_espera_100ms = '1') then
         cuenta100ms <= (others=>'0');
         fin_espera_100ms <= '0';
         --led1 <= Eli_Reb;
      elsif (clk'event and clk=HI) then
         if (cuenta100ms = 1000000) then
            fin_espera_100ms <= HI;
            --led1 <= CERO;
         else
            cuenta100ms <= cuenta100ms + 1;
            fin_espera_100ms <= LO;
            --led1 <= cuenta100ms(16 downto 10);
         end if;
      end if;
   end process;

end flashCon_arch;
```

C.8. Módulo que elimina rebotes de una señal:

rebotes.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity rebotes is
    port (
        clk: in STD_LOGIC;
        rst: in STD_LOGIC;

        accion : in STD_LOGIC;
        accionOK : out STD_LOGIC
          );
end rebotes;

architecture uno of rebotes is

SIGNAL xSync, xDebFallingEdge, xDebRisingEdge, timerEnd, startTimer:
std_logic;

begin

  synchronizer:
  PROCESS (rst, clk)
    VARIABLE aux1: std_logic;
  BEGIN
    IF (rst='0') THEN
      aux1 := '1';
```

```
      xSync <= '1';
    ELSIF (clk'EVENT AND clk='1') THEN
      xSync <= aux1;
      aux1 := accion;
    END IF;
  END PROCESS synchronizer;

  timer:
  -- espera 50 ms para un reloj a 12.5 MHz
  PROCESS (rst, clk)
    CONSTANT timeOut: std_logic_vector (19 DOWNTO 0) :=
"10011000100101101000";
    VARIABLE count: std_logic_vector (19 DOWNTO 0);
  BEGIN
    IF (count=timeOut) THEN
      timerEnd <= '1';
    ELSE
      timerEnd <= '0';
    END IF;
    IF (rst='0') THEN
      count := timeOut;
    ELSIF (clk'EVENT AND clk='1') THEN
      IF (startTimer='1') THEN
        count := (OTHERS=>'0');
      ELSIF (timerEnd='0') THEN
        count := count + 1;
      END IF;
    END IF;
  END PROCESS timer;

  controller:
  PROCESS (xSync, rst, clk)
    TYPE states IS (waitingPression, pressionDebouncing,
waitingDepression, depressionDebouncing);
    VARIABLE state: states;
  BEGIN
    accionOK <= '1';
    xDebFallingEdge <= '0';
    xDebRisingEdge <= '0';
    startTimer <= '0';
    CASE state IS
      WHEN waitingPression =>
        IF (xSync='0') THEN
          xDebFallingEdge <= '1';
          startTimer <= '1';
        END IF;
      WHEN pressionDebouncing =>
        accionOK <= '0';
      WHEN waitingDepression =>
        accionOK <= '0';
        IF (xSync='1') THEN
          xDebRisingEdge <= '1';
          startTimer <= '1';
        END IF;
      WHEN depressionDebouncing =>
        NULL;
      END CASE;
    IF (rst='0') THEN
      state := waitingPression;
    ELSIF (clk'EVENT AND clk='1') THEN
      CASE state IS
```

```
        WHEN waitingPression =>
          IF (xSync='0') THEN
            state := pressionDebouncing;
          END IF;
        WHEN pressionDebouncing =>
          IF (timerEnd='1') THEN
            state := waitingDepression;
          END IF;
        WHEN waitingDepression =>
          IF (xSync='1') THEN
            state := depressionDebouncing;
          END IF;
        WHEN depressionDebouncing =>
          IF (timerEnd='1') THEN
            state := waitingPression;
          END IF;
      END CASE;
    END IF;
  END PROCESS controller;

end uno;
```

C.9. BMP2PACA

C.9.1 matrizRGB.h

```
//---------------------------------------------------------------------------

#ifndef RGBColorH
#define RGBColorH
class matrizRGB
{
        typedef unsigned char uchar;

        public:
        uchar r,g,b;
        matrizRGB();
        matrizRGB(matrizRGB &color);
};
//---------------------------------------------------------------------------
#endif
```

C.9.2 matrizRGB.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "matrizRGB.h"
//---------------------------------------------------------------------------
matrizRGB::matrizRGB()
{
        r = g = b = 0;
}
//---------------------------------------------------------------------------
matrizRGB::matrizRGB(matrizRGB &color)
{
        r = color.r;
        g = color.g;
        b = color.b;
}
//---------------------------------------------------------------------------

#pragma package(smart_init)
```

C.9.3 RGBPixmap.h

```
//---------------------------------------------------------------------
-------

#ifndef RGBPixmapH
#define RGBPixmapH

#include "matrizRGB.h"

class RGBPixmap
{

   typedef unsigned char uchar; // 8 bits
   typedef unsigned short int ushortint; // 16 bits
   typedef unsigned int uint; // 32 bits

   private:
   matrizRGB *pixel;
   bool correctos(int x ,int y);
   int columDeDibujo,    //628
       filasDeDibujo,    //479
       columEnPantalla,  //1023
       filasEnPantalla;  //528
   int despR,   //10
       despG,   //5
       despB;   //0

   public:
   void setColumDeDibujo(int a) {columDeDibujo=a;}
   void setFilasDeDibujo(int a) { filasDeDibujo=a;}
   void setColumEnPantalla(int a) { columEnPantalla=a;}
   void setFilasEnPantalla(int a) { filasEnPantalla=a;}
   void setDespR(int a) {despR=a;}
   void setDespG(int a) {despG=a;}
   void setDespB(int a) {despB=a;}
   int nCols,nRows;
   RGBPixmap();
   RGBPixmap(int rows,int cols);
   void setPixel(int x,int y,matrizRGB color);
   matrizRGB getPixel(int x,int y);
   void draw();
   bool readBMPFile(AnsiString fname);
   bool writeBMPFile(AnsiString fname);
   bool writePruebaHEX();
   bool write2HEX(AnsiString fname);
   bool writePruebaXES();
   bool write2XESS24(AnsiString fname);
   bool write2paca(AnsiString fname);
   ~RGBPixmap();

};
//---------------------------------------------------------------------
-------
#endif
```

C.9.4 RGBPixmap.cpp

```
//---------------------------------------------------------------------
-------
```

```
#include <vcl.h>
#pragma hdrstop

#include "RGBPixmap.h"
#include <gl\gl.h>
#include <gl\glu.h>
#include <stdio.h>
#include <math.h>
#include <memory.h>
#include <string.h>
#include <fstream.h>
#include <iomanip.h>
#define DEBUG 1

//---------------------------------------------------------------------
-------
RGBPixmap::RGBPixmap()
{
        nRows = nCols = 0;
        pixel = NULL;

        columDeDibujo = 628;    //628
        filasDeDibujo = 479;    //479
        columEnPantalla = 1023;  //794
        filasEnPantalla = 512;  //528

        despR = 10;
        despG = 5;
        despB = 0;
}
//---------------------------------------------------------------------
-------
RGBPixmap::RGBPixmap(int rows,int cols)
{
        /*
        nRows = rows;
        nCols = cols;
        */
        nRows = cols;
        nCols = rows;
        pixel = new matrizRGB[nCols*nRows];

        columDeDibujo = 628;    //628
        filasDeDibujo = 479;    //479
        columEnPantalla = 1023;  //794
        filasEnPantalla = 512;  //528

        despR = 10;
        despG = 5;
        despB = 0;
}
//---------------------------------------------------------------------
-------
void RGBPixmap::setPixel(int x,int y,matrizRGB color)
{
        if(correctos(x,y))
                pixel[nCols*y + x] = color;
}
//---------------------------------------------------------------------
-------
```

```
matrizRGB RGBPixmap::getPixel(int x,int y)
{
        if(correctos(x,y))
                return pixel[nCols*y + x];
        else
        {
                matrizRGB c;
                return c;
        }
}
//-------------------------------------------------------------------
-------
void RGBPixmap::draw()
{
  if (nRows==0 || nCols==0)
    return;
  glPixelStorei(GL_UNPACK_ALIGNMENT,1);
  glDrawPixels(nCols,nRows,GL_RGB,GL_UNSIGNED_BYTE,pixel);
}
//-------------------------------------------------------------------
-------
bool RGBPixmap::correctos(int x,int y)
{
        bool retVal = false;
        if((x>=0) && (x<nCols) && (y>=0) && (y<nRows))
                retVal = true;
        return retVal;
}
//-------------------------------------------------------------------
-------
bool RGBPixmap::readBMPFile(AnsiString fname)
{
  fstream inf;
  inf.open(fname.c_str(),ios::in|ios::binary);
  if (!inf) return false;
  //leemos la cabecera
  BITMAPFILEHEADER bmfh;
  inf.read((char*)&bmfh,sizeof(BITMAPFILEHEADER));
  //leemos la información del bitmap
  BITMAPINFOHEADER bmih;
  inf.read((char*)&bmih,sizeof(BITMAPINFOHEADER));
  //nos aseguramos que la profundidad de color es de 24
  if (bmih.biBitCount != 24)
  {
    inf.close();
    return false;
  }
  //calculamos el número de bytes sobrantes por fila
  int numPadbytes=(4-((bmih.biWidth*3)%4))%4;
  // número de columnas y filas del pixmap

  nCols=bmih.biWidth;
  nRows=bmih.biHeight;

  // generamos el pixmap
  delete []pixel;
  pixel=new matrizRGB[nCols*nRows];
  if (!pixel)
  {
     inf.close();
     pixel=NULL;
```

```
     return false;
  }
  //saltar los bytes referentes a tablas de colores
  char dump;
  for (int k=0; k < bmfh.bfOffBits-54; k++)
    inf.get(dump);
  long count=0;
  for (int row=0; row < nRows; row++)
  {
    for (int col=0; col < nCols; col++)
    {
      char r,g,b;
      inf.get(b);
      inf.get(g);
      inf.get(r);
      pixel[count].r=r;
      pixel[count].g=g;
      pixel[count].b=b;
      count++;
    }
    for (int k=0; k < numPadbytes; k++)
      inf.get(dump);
  }
  inf.close();
  return true;
}
//---------------------------------------------------------------------------
-------
bool RGBPixmap::writeBMPFile(AnsiString fname)
{
        fstream inf;
        inf.open(fname.c_str(),ios::out|ios::binary);
        if (!inf) return false;
        //cargamos la cabecera
        BITMAPFILEHEADER bmfh;
        bmfh.bfType=19778;
        bmfh.bfSize=0;
        bmfh.bfReserved1=0;
        bmfh.bfReserved2=0;
        bmfh.bfOffBits=54;
        inf.write((char*)&bmfh,sizeof(BITMAPFILEHEADER));
        //cargamos la configuración de la imagen
        BITMAPINFOHEADER bmih;
        bmih.biSize=40;
        bmih.biWidth=nCols;
        bmih.biHeight=nRows;
        bmih.biPlanes=1;
        bmih.biBitCount=24;
        bmih.biCompression=0;
        int numPadbytes=(4-((nCols*3)%4))%4;
        bmih.biSizeImage=nCols*nRows*3+nRows*numPadbytes;
        bmih.biXPelsPerMeter=0;
        bmih.biYPelsPerMeter=0;
        bmih.biClrUsed=0;
        bmih.biClrImportant=0;
        inf.write((char*)&bmih,sizeof(BITMAPINFOHEADER));
        long count=0;
        char dump='0';
        //cargamos el pixmap
        for (int row=0; row < nRows; row++)
        {
```

```
                for (int col=0; col < nCols; col++)
                {
                        char r,g,b;
                        r=pixel[count].r;
                        g=pixel[count].g;
                        b=pixel[count].b;
                        inf.put(b);
                        inf.put(g);
                        inf.put(r);
                        count++;
                }
                // añadimos para completar la escritura en bloques de
4
                for (int k=0; k < numPadbytes; k++)
                {
                        inf.put(dump);
                }
        }
        inf.close();
        return true;
}
//---------------------------------------------------------------------
-------
bool RGBPixmap::write2HEX(String fname)
{
        ofstream outf;
        const char* nombre = fname.c_str();
        outf.open(nombre);
        if (!outf) return false;
        long cont=0;
        ushortint dir=0;
        ushortint CRC;
        //Metemos negro en la posicion cero
        outf<<":020000000000FE";
        dir+=2;
        outf<<'\n';
        //Aki se recorre el pixMap y se va guardando un pixel en cada
linea
        for (int row=0; row < nRows; row++)
        {
          for (int col=0; col < nCols; col++)
          {
            if ((col <= columDeDibujo) && (row <= filasDeDibujo))
            {
              outf<<":02";
              CRC= 2;
              outf<<setbase(16)<<setw(4)<<setfill('0')<< dir;
              CRC= CRC + (dir + ( dir>>8));
              dir+=2;
              outf<<"00";
              ushortint rBIN=pixel[cont].r;
              ushortint gBIN=pixel[cont].g;
              ushortint bBIN=pixel[cont].b;
              //Quito los 3 bits menos significativos
              rBIN=rBIN>>3;
              gBIN=gBIN>>3;
              bBIN=bBIN>>3;
              //Los desplazo para colocarlos en su sitio
              rBIN=rBIN<<10;
              gBIN=gBIN<<5;
              //Los sumamos para tenerlos todos en uno
```

```
            ushortint colorBIN= rBIN + gBIN + bBIN;
            outf<<setbase(16)<<setw(4)<<setfill('0')<< colorBIN;
            CRC= CRC + (colorBIN + (colorBIN>>8));
            CRC= (~CRC +1) & 0x00FF;
            outf<<setbase(16)<<setw(2)<<setfill('0')<< CRC;
            outf<<'\n';
          }
          cont++;
        }//cols
        for (int fill=nCols; fill<629; fill++)
        //Si el grafico es mas pequeño que la pantalla, rellenamos
de negro
        //Rellenamos las columnas que faltan
        {
          outf<<":02";
          CRC= 2;
          outf<<setbase(16)<<setw(4)<<setfill('0')<< dir;
          CRC= CRC + (dir + ( dir>>8));
          dir+=2;
          outf<<"00";
          outf<<"0000";
          CRC= (~CRC +1) & 0x00FF;
          outf<<setbase(16)<<setw(2)<<setfill('0')<< CRC;
          outf<<'\n';
        }
        for (int blank=629; blank<=746; blank++)
        //Los espacios de blanking
        {
          outf<<":02";
          CRC= 2;
          outf<<setbase(16)<<setw(4)<<setfill('0')<< dir;
          CRC= CRC + (dir + ( dir>>8));
          dir+=2;
          outf<<"00";
          outf<<"0000";
          CRC= (~CRC +1) & 0x00FF;
          outf<<setbase(16)<<setw(2)<<setfill('0')<< CRC;
          outf<<'\n';
        }
      }//rows
      for (int fill=nRows; fill<480; fill++)
      //Si el grafico es mas pequeño que la pantalla, rellenamos de
negro
      //Rellenamos las filas que faltan
      {
        for (int c=0; c<=746; c++)
        {
          outf<<":02";
          CRC= 2;
          outf<<setbase(16)<<setw(4)<<setfill('0')<< dir;
          CRC= CRC + (dir + ( dir>>8));
          dir+=2;
          outf<<"00";
          outf<<"0000";
          CRC= (~CRC +1) & 0x00FF;
          outf<<setbase(16)<<setw(2)<<setfill('0')<< CRC;
          outf<<'\n';
        }
      }
      for (int blank=480; blank<=496; blank++)
      //Los espacios de blanking
```

```
        {
          outf<<":02";
          CRC= 2;
          outf<<setbase(16)<<setw(4)<<setfill('0')<< dir;
          CRC= CRC + (dir + ( dir>>8));
          dir+=2;
          outf<<"00";
          outf<<"0000";
          CRC= (~CRC +1) & 0x00FF;
          outf<<setbase(16)<<setw(2)<<setfill('0')<< CRC;
          outf<<'\n';
        }
        //Esta es la ultima linea que indica el final del HEX
        outf<<":00000001FF";
        outf<<'\n';
        outf.close();
        return true;
}
//-------------------------------------------------------------------------
-------
bool RGBPixmap::writePruebaHEX()
{
        ofstream outf;
        const char* nombre = "prueba.hex";
        outf.open(nombre);
        if (!outf) return false;
        ushortint datosBIN;
        ushortint CRC;
        ushortint dir=0;
        //Metemos negro en la posicion cero
        //outf<<":0200000000FE";
        //outf<<'\n';
        //Vamos a generar la parrilla:
        for (uint fil=0; fil < filasEnPantalla; fil++)
        {
          for (uint col=0; col < columEnPantalla; col++)
          {
            if ((col <= columDeDibujo) && (fil <= filasDeDibujo))
            {
              outf<<":02";
              CRC= 2;
              ushortint colBIN= col;
              ushortint filBIN= fil;
              //Generamos dir
              filBIN= filBIN<<10;
              dir= filBIN + colBIN;
              outf<<hex<<setw(4)<<setfill('0')<< dir;
              CRC= CRC + (dir + ( dir>>8));
              outf<<"0000";
              //Generamos datos
              datosBIN= colBIN>>3;
              datosBIN= datosBIN & 0x3F;
              outf<<hex<<setw(2)<<setfill('0')<< datosBIN;
              CRC= CRC + datosBIN;
              CRC= (~CRC +1) & 0x00FF;
              outf<<hex<<setw(2)<<setfill('0')<< CRC;
              outf<<'\n';
            }//if
            else
            //Los espacios de blanking
            {
```

```
                outf<<":02";
                CRC= 2;
                ushortint colBIN= col;
                ushortint filBIN= fil;
                //Generamos dir
                filBIN= filBIN<<10;
                dir= filBIN + colBIN;
                outf<<hex<<setw(4)<<setfill('0')<< dir;
                CRC= CRC + (dir + ( dir>>8));
                outf<<"00";
                outf<<"00"; //Ponemos negro
                CRC= (~CRC +1) & 0x00FF;
                outf<<hex<<setw(2)<<setfill('0')<< CRC;
                outf<<'\n';
              }
            }//cols
          }//filas
          //Esta es la ultima linea que indica el final del HEX
          outf<<":00000001FF";
          outf<<'\n';
          outf.close();
          return true;
}
//---------------------------------------------------------------------
-------
bool RGBPixmap::write2XESS24(String fname)
{
          ofstream outf;
          const char* nombre = fname.c_str();
          outf.open(nombre);
          if (!outf) return false;
          long cont=0;
          uint dir=0;
          //Metemos negro en la posicion cero
          outf<<"= 02 000000 00 00";
          dir+=2;
          outf<<'\n';
          //Aki se recorre el pixMap y se va guardando un pixel en cada
linea
          for (int row=0; row < nRows; row++)
          {
            for (int col=0; col < nCols; col++)
            {
              if ((col <= columDeDibujo) && (row <= filasDeDibujo))
              {
                outf<<"= 02 ";
                outf<<setbase(16)<<setw(6)<<setfill('0')<< dir;
                dir+=2;
                outf<<' ';
                //Los 00 de control
                //outf<<"00 ";
                uint rBIN=pixel[cont].r;
                uint gBIN=pixel[cont].g;
                uint bBIN=pixel[cont].b;
                //Quito los 3 bits menos significativos
                rBIN=rBIN>>3;
                gBIN=gBIN>>3;
                bBIN=bBIN>>3;
                //Los desplazo para colocarlos en su sitio
                rBIN=rBIN<<10;
                gBIN=gBIN<<5;
```

```
              //Los sumamos para tenerlos todos en uno
              uint colorBIN= rBIN + gBIN + bBIN;
              uint colorBINdesp= colorBIN>>8;
              outf<<setbase(16)<<setw(2)<<setfill('0')<< colorBINdesp;
              outf<<' ';
              outf<<setbase(16)<<setw(2)<<setfill('0')<< (colorBIN &
0x00FF);
              outf<<'\n';
            }
            cont++;
          }//cols
          for (int fill=nCols; fill<629; fill++)
          //Si el grafico es mas pequeño que la pantalla, rellenamos
de negro
          //Rellenamos las columnas que faltan
          {
            outf<<"= 02 ";
            outf<<setbase(16)<<setw(6)<<setfill('0')<< dir;
            dir+=2;
            //outf<<"00 ";
            outf<<" 00 00";
            outf<<'\n';
          }
          for (int blank=629; blank<=746; blank++)
          //Los espacios de blanking
          {
            outf<<"= 02 ";
            outf<<setbase(16)<<setw(6)<<setfill('0')<< dir;
            dir+=2;
            //outf<<"00 ";
            outf<<" 00 00";
            outf<<'\n';
          }
        }//rows
        for (int fill=nRows; fill<480; fill++)
        //Si el grafico es mas pequeño que la pantalla, rellenamos de
negro
        //Rellenamos las filas que faltan
        {
          for (int c=0; c<=746; c++)
          {
            outf<<"= 02 ";
            outf<<setbase(16)<<setw(6)<<setfill('0')<< dir;
            dir+=2;
            //outf<<" 00";
            outf<<" 00 00";
            outf<<'\n';
          }
        }
        for (int blank=480; blank<=496; blank++)
        //Los espacios de blanking
        {
          outf<<"= 02 ";
          outf<<setbase(16)<<setw(6)<<setfill('0')<< dir;
          dir+=2;
          //outf<<" 00";
          outf<<" 00 00";
          outf<<'\n';
        }
        //Esta es la ultima linea que indica el final del HEX
        outf<<"= 00 000000 01";
```

```
        outf<<'\n';
        outf.close();
        return true;
}
//-----------------------------------------------------------------------
-------
bool RGBPixmap::writePruebaXES()
{
        ofstream outf;
        const char* nombre = "prueba.xes";
        outf.open(nombre);
        if (!outf) return false;
        ushortint datosBIN;
        ushortint dir=0;
        //Metemos negro en la posicion cero
        //outf<<"= 02 000000 00 00";
        //outf<<'\n';
        //Vamos a generar la parrilla:
        for (uint fil=0; fil < filasEnPantalla; fil++)
        {
          for (uint col=0; col < columEnPantalla; col++)
          {
            if ((col <= columDeDibujo) && (fil <= filasDeDibujo))
            {
              outf<<"= 02 ";
              ushortint colBIN= col;
              ushortint filBIN= fil;
              //Generamos dir
              filBIN= filBIN<<10;
              dir= filBIN + colBIN;
              outf<<hex<<setw(6)<<setfill('0')<< dir;
              outf<<" 00 ";
              //Generamos datos
              datosBIN= colBIN>>3;
              datosBIN= datosBIN & 0x3F;
              outf<<hex<<setw(2)<<setfill('0')<< datosBIN;
              outf<<'\n';
            }//if
            else
            //Los espacios de blanking
            {
              outf<<"= 02 ";
              ushortint colBIN= col;
              ushortint filBIN= fil;
              //Generamos dir
              filBIN= filBIN<<10;
              dir= filBIN + colBIN;
              outf<<hex<<setw(6)<<setfill('0')<< dir;
              outf<<" 00 00 ";
              outf<<"00"; //Ponemos negro
              outf<<'\n';
            }
          }//cols
        }//filas
        //Esta es la ultima linea que indica el final del HEX
        outf<<"= 00 000000 01";
        outf<<'\n';
        outf.close();
        return true;
}
```

```
//-------------------------------------------------------------------------
-------

void itob(char *c, int num)
{
int i;
for(i=0xF; i>=0; i--)
{
c[i]=(num & 0x0001)+'0';
num >>= 1;
}
c[0x10]='\0';
}
//-------------------------------------------------------------------------
-------
bool RGBPixmap::write2paca(String fname)
{
     ofstream outf;
     const char* nombre = fname.c_str();
     outf.open(nombre);
     if (!outf) return false;
     long cont;
     //long cont = 0;
     //Empezamos a escribir en la posicion 0
     outf<<":0";
     outf<<'\n';
     //Aki se recorre el pixMap y se va guardando un pixel en cada
linea
     //for (int row=0; row < nRows; row++)
     for (int row=(nRows-1); row >= 0 ; row--)
     {
        if (row <= filasDeDibujo)
        {
          for (int col=0; col < nCols; col++)
          {
            if (col <= columDeDibujo)
            {
              cont = col + (row*nCols);
              uint rBIN=(pixel[cont].r);
              uint gBIN=(pixel[cont].g);
              uint bBIN=(pixel[cont].b) ;
              //Quito los 3 bits menos significativos

              rBIN=rBIN>>3;
              gBIN=gBIN>>3;
              bBIN=bBIN>>3;

              //Los desplazo para colocarlos en su sitio
              rBIN=rBIN<<despR;
              gBIN=gBIN<<despG;
              bBIN=bBIN<<despB;
              //Los sumamos para tenerlos todos en uno
              uint colorBIN= rBIN + gBIN + bBIN;
              char bin[16];
              itob (bin,colorBIN);
              outf<<bin;
              //outf<<aBin(colorBIN);
              //outf<<setbase(2)<<setw(16)<<setfill('0')<<colorBIN;
              outf<<'\n';
            }
//            cont++;
```

```
          }//cols
          for (int fill=nCols; fill<(columDeDibujo + 1); fill++)
          //Si el grafico es mas pequeño que la pantalla, rellenamos
de negro
          //Rellenamos las columnas que faltan
          {
            outf<<"0000000000000000";
            outf<<'\n';
          }
//          for (int blank=629; blank<=746; blank++)
          for (int blank=(columDeDibujo + 1); blank<=columEnPantalla;
blank++)
          //Los espacios de blanking
          {
            outf<<"0000000000000000";
            outf<<'\n';
          }
        }//del if
     }//rows
     //for (int fill=nRows; fill<480; fill++)
     for (int fill=nRows; fill<filasDeDibujo+1; fill++)
     //Si el grafico es mas pequeño que la pantalla, rellenamos de
negro
     //Rellenamos las filas que faltan
      {
//        for (int c=0; c<=746; c++)
        for (int c=0; c<=columEnPantalla; c++)
        {
          outf<<"0000000000000000";
          outf<<'\n';
        }
      }
//      for (int blank=480; blank<=496; blank++)
      for (int blank=(filasDeDibujo + 1); blank<=filasEnPantalla;
blank++)
      //Los espacios de blanking
      {
//         for (int blank=0; blank<=746; blank++) {
         for (int blank=0; blank<=columEnPantalla; blank++) {
          outf<<"0000000000000000";
          outf<<'\n';
         }
      }
      //No hay ultima linea
      outf.close();
      return true;
}
//---------------------------------------------------------------------
-------
RGBPixmap::~RGBPixmap()
{
   nRows = nCols = 0;
   delete[] pixel;
}
#pragma package(smart_init)
```

C.9.5 GlSkel.h

```
//---------------------------------------------------------------------
-------
#ifndef GlSkelH
```

```
#define GlSkelH
//---------------------------------------------------------------------------
-------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Dialogs.hpp>
#include <ExtDlgs.hpp>
#include <Menus.hpp>
#include <gl\gl.h>
#include <gl\glu.h>

#include "RGBPixmap.h"
#include "param.h"

//---------------------------------------------------------------------------
-------
class TGLForm2D : public TForm
{
__published:      // IDE-managed Components
        TMainMenu *MainMenu1;
        TMenuItem *Archivo1;
        TMenuItem *Abrir1;
        TMenuItem *Salir1;
        TMenuItem *Transformar1;
        TMenuItem *aHEX1;
        TOpenDialog *DialogoAbrir;
        TSaveDialog *Dialogo2HEX;
        TMenuItem *PruebaHEX;
        TMenuItem *aXESS241;
        TSaveDialog *Dialogo2XES;
        TMenuItem *PruebaXESS241;
        TMenuItem *Pruebas1;
        TMenuItem *apaca1;
        TSaveDialog *Dialogo2paca;
        TMenuItem *Parmetros1;
    void __fastcall FormResize(TObject *Sender);
    void __fastcall FormPaint(TObject *Sender);
    void __fastcall FormDestroy(TObject *Sender);
    void __fastcall FormCreate(TObject *Sender);
        void __fastcall FormKeyDown(TObject *Sender, WORD &Key,
          TShiftState Shift);
        void __fastcall Abrir1Click(TObject *Sender);
        void __fastcall aHEX1Click(TObject *Sender);
        void __fastcall Salir1Click(TObject *Sender);
        void __fastcall PruebaHEXClick(TObject *Sender);
        void __fastcall aXESS241Click(TObject *Sender);
        void __fastcall PruebaXESS241Click(TObject *Sender);
        void __fastcall apaca1Click(TObject *Sender);
        void __fastcall Parmetros1Click(TObject *Sender);
private:    // User declarations
    HDC hdc;
    HGLRC hrc;
    GLfloat w, h;
    GLint x, y;
    GLfloat nRange;
    GLdouble left,right,bottom,top;
    void SetPixelFormatDescriptor();
    void __fastcall GLScene();
    void __fastcall DrawEjes();
```

```
public:           // User declarations
    __fastcall TGLForm2D(TComponent* Owner);

    RGBPixmap *pixMap;
    bool hayBMP;

};
//---------------------------------------------------------------------
-------
extern PACKAGE TGLForm2D *GLForm2D;
//---------------------------------------------------------------------
-------
#endif
```

C.9.5 GlSkel.cpp

```
//---------------------------------------------------------------------
-------
#include <vcl.h>
#pragma hdrstop

#include "GlSkel.h"
//---------------------------------------------------------------------
-------
#pragma package(smart_init)    +
#pragma resource "*.dfm"
TGLForm2D *GLForm2D;
//---------------------------------------------------------------------
-------
__fastcall TGLForm2D::TGLForm2D(TComponent* Owner)
    : TForm(Owner)
{
  hayBMP= false;
}
//---------------------------------------------------------------------
-------
void __fastcall TGLForm2D::FormCreate(TObject *Sender)
{
  hdc = GetDC(Handle);
  SetPixelFormatDescriptor();
  hrc = wglCreateContext(hdc);
  if (hrc == NULL)
     ShowMessage("Error CreateContex");
  if (wglMakeCurrent(hdc, hrc) == false)
     ShowMessage("Error MakeCurrent");
  ClientWidth = 628;
  w = 628;
  h = 478;
  ClientHeight = 478;
  /*nRange = ;   // w/2 == h/2
  left = -nRange;
  right = nRange;
  bottom = -nRange;
  top = nRange;
  */
  left = -314;
  right = 314;
  bottom = -239;
  top = 239;
  x = 10;
```

```
  y = 10;
  // podemos definir otros colores
  glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
  // crear objetos
  pixMap = new RGBPixmap();
}
//---------------------------------------------------------------------
-------
void TGLForm2D::SetPixelFormatDescriptor()
{
  PIXELFORMATDESCRIPTOR pfd = {
     sizeof(PIXELFORMATDESCRIPTOR),
     1,
     PFD_DRAW_TO_WINDOW | PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER,
     PFD_TYPE_RGBA,
     24,
     0,0,0,0,0,0,
     0,0,
     0,0,0,0,0,
     32,
     0,
     0,
     PFD_MAIN_PLANE,
     0,
     0,0,0 };
  int PixelFormat = ChoosePixelFormat(hdc, &pfd);
  SetPixelFormat(hdc, PixelFormat, &pfd);
}
//---------------------------------------------------------------------
-------
void __fastcall TGLForm2D::FormResize(TObject *Sender)
{
  w = ClientWidth;
  h = ClientHeight;
  if (h==0) h=1.0;
  if (w==0) w=1.0;

  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  if (w <= h)
     {
     left = -nRange;
     right = nRange;
     bottom = -nRange*h/w;
     top = nRange*h/w;
     }
  else
     {
     left = -nRange*w/h;
     right = nRange*w/h;
     bottom = -nRange;
     top = nRange;
     }
  gluOrtho2D (left, right, bottom, top);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  GLScene();
}
//---------------------------------------------------------------------
-------
```

```
void __fastcall TGLForm2D::GLScene()
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  // instrucciones para dibujar la escena
  // DrawEjes();
  if (hayBMP)
     pixMap->draw();
  glFlush();
  SwapBuffers(hdc);
}
//---------------------------------------------------------------------
-------
void __fastcall TGLForm2D::DrawEjes()
{
  glBegin(GL_LINES);
    glColor3f(1.0,0.0,0.0);
    glVertex2i(-x,0);
    glVertex2i(x,0);
    glColor3f(0.0,1.0,0.0);
    glVertex2i(0,-y);
    glVertex2i(0,y);
  glEnd();
}
//---------------------------------------------------------------------
-------
void __fastcall TGLForm2D::FormPaint(TObject *Sender)
{
  GLScene();
}
//---------------------------------------------------------------------
-------
void __fastcall TGLForm2D::FormDestroy(TObject *Sender)
{
  // liberar  objetos
  ReleaseDC(Handle,hdc);
  wglMakeCurrent(NULL, NULL);
  wglDeleteContext(hrc);
  delete pixMap;
}
//---------------------------------------------------------------------
--------
void __fastcall TGLForm2D::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
if (Key == VK_UP)
   {
   x = (x + 10) % (GLint)right;
   y = (y + 10) % (GLint)top;
   }
if (Key == VK_DOWN)
   {
   x = (x - 10) % (GLint)left;
   y = (y - 10) % (GLint)bottom;
   }
GLScene();
}
//---------------------------------------------------------------------
-------

void __fastcall TGLForm2D::Abrir1Click(TObject *Sender)
{
```

```
  if (DialogoAbrir->Execute())
  {
     hayBMP= pixMap->readBMPFile(DialogoAbrir->FileName);
     GLScene();
  }
}
//---------------------------------------------------------------------
-------

void __fastcall TGLForm2D::aHEX1Click(TObject *Sender)
{
  if (hayBMP && Dialogo2HEX->Execute())
  {
    bool ok= pixMap->write2HEX(Dialogo2HEX->FileName);
    if (ok)
      ShowMessage ("Perfecto, ya se ha creado el .HEX");
    else
      ShowMessage ("Vaya, algo ha salido mal");
  }
  else
    ShowMessage ("Carga un .BMP primero, vamos, digo yo...");
}
//---------------------------------------------------------------------
-------

void __fastcall TGLForm2D::Salir1Click(TObject *Sender)
{
 if (MessageDlg("¿No quieres pasar nada mas a HEX?, ¿seguro?",
                mtConfirmation,TMsgDlgButtons() << mbYes << mbNo,0)
== mrYes)
   Application->Terminate();
}
//---------------------------------------------------------------------
-------

void __fastcall TGLForm2D::PruebaHEXClick(TObject *Sender)
{
  bool ok= pixMap->writePruebaHEX();
  if (ok)
    ShowMessage ("Perfecto, ya esta el prueba.hex");
  else
    ShowMessage ("Vaya, algo ha salido mal");
}
//---------------------------------------------------------------------
-------

void __fastcall TGLForm2D::aXESS241Click(TObject *Sender)
{
     if (hayBMP && Dialogo2XES->Execute())
  {
    bool ok= pixMap->write2XESS24(Dialogo2XES->FileName);
    if (ok)
      ShowMessage ("Perfecto, ya se ha creado el .XES");
    else
      ShowMessage ("Vaya, algo ha salido mal");
  }
  else
    ShowMessage ("Carga un .BMP primero, vamos, digo yo...");
}
//---------------------------------------------------------------------
-------
```

```
void __fastcall TGLForm2D::PruebaXESS241Click(TObject *Sender)
{
  bool ok= pixMap->writePruebaXES();
  if (ok)
    ShowMessage ("Perfecto, ya esta el prueba.xes");
  else
    ShowMessage ("Vaya, algo ha salido mal");

}
//---------------------------------------------------------------------
-------

void __fastcall TGLForm2D::apaca1Click(TObject *Sender)
{
  if (hayBMP && Dialogo2paca->Execute())
  {
    bool ok= pixMap->write2paca(Dialogo2paca->FileName);
    if (ok)
      ShowMessage ("Perfecto, ya se ha creado el .paca");
    else
      ShowMessage ("Vaya, algo ha salido mal");
  }
  else
    ShowMessage ("Carga un .BMP primero, vamos, digo yo...");
}
//---------------------------------------------------------------------
-------



void __fastcall TGLForm2D::Parmetros1Click(TObject *Sender)
{
   FormParam->ShowModal();

   if (FormParam->ModalResult== mrOk){
      pixMap->setColumDeDibujo(FormParam->colDib->Value);
      pixMap->setFilasDeDibujo(FormParam->filDib->Value);
      pixMap->setColumEnPantalla(FormParam->colPant->Value);
      pixMap->setFilasEnPantalla(FormParam->filPant->Value);

      pixMap->setDespR(FormParam->despRojo->Value);
      pixMap->setDespG(FormParam->despVerde->Value);
      pixMap->setDespB(FormParam->despAzul->Value);
   }
}
//---------------------------------------------------------------------
-------
```

C.10 DIBUBIF

C.10.1 Pixel.h

```
#ifndef ClasePunto
#define ClasePunto
//---------------------------------------------------------------------
-------
```

```
#include <gl\gl.h>
#include <gl\glu.h>
#include <Math.h>

class pixel {

private:
        float r,g,b;
        long unsigned int x;
public:
        pixel() {;}
        pixel(long unsigned int _x,GLfloat _r, GLfloat _g, GLfloat _b)
                {x=_x; r=_r; g=_g; b=_b;}
        void set(long unsigned int _x,GLfloat _r, GLfloat _g, GLfloat
_b)
                {x=_x; r=_r; g=_g; b=_b;}
        void set(GLfloat _r, GLfloat _g, GLfloat _b)
                {r=_r; g=_g; b=_b;}
        long unsigned int X() {return x;}
        GLfloat R() {return r;}
        GLfloat G() {return g;}
        GLfloat B() {return b;}
};
#endif
```

C.10.2 GlSkel.h

```
//---------------------------------------------------------------------
-------
#ifndef GlSkelH
#define GlSkelH
//---------------------------------------------------------------------
-------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Menus.hpp>
#include <Dialogs.hpp>
#include <vector.h>
#include "CGAUGES.h"
#include <ComCtrls.hpp>
#include <gl\gl.h>
#include <gl\glu.h>
#include "pixel.h"
//---------------------------------------------------------------------
-------
class TFormPrincipal : public TForm
{
__published:      // IDE-managed Components
        TMainMenu *MainMenu1;
        TMenuItem *MenuArchivo;
        TMenuItem *MenuCargar;
        TMenuItem *MenuSalir;
        TOpenDialog *OpenDialog;
        TMenuItem *Parmetros1;
        TMenuItem *Filtros1;
        TMenuItem *Negativizar1;
```

```
        TMenuItem *Escaladegrises1;
        TMenuItem *Binarizar1;
        TMenuItem *N8colores1;
        TMenuItem *Suavizado1;
        TMenuItem *Bordes1;
        TStatusBar *StatusBar;
    void __fastcall FormResize(TObject *Sender);
    void __fastcall FormPaint(TObject *Sender);
    void __fastcall FormDestroy(TObject *Sender);
    void __fastcall FormCreate(TObject *Sender);
        void __fastcall MenuCargarClick(TObject *Sender);
        void __fastcall MenuSalirClick(TObject *Sender);
        void __fastcall Parmetros1Click(TObject *Sender);
        void __fastcall Escaladegrises1Click(TObject *Sender);
        void __fastcall Negativizar1Click(TObject *Sender);
        void __fastcall Binarizar1Click(TObject *Sender);
        void __fastcall N8colores1Click(TObject *Sender);
        void __fastcall Suavizado1Click(TObject *Sender);
        void __fastcall Bordes1Click(TObject *Sender);


private:    // User declarations
    HDC hdc;
    HGLRC hrc;
    GLfloat w, h;
    GLint x, y;
    GLfloat nRange;
    vector<pixel> pixmap;
    GLint DespR,DespG,DespB;
    AnsiString nombre_archivo;
    GLint ColumDeDibujo,FilasDeDibujo,ColumEnPantalla,FilasEnPantalla;
    GLdouble _left,_right,_bottom,_top;
    void SetPixelFormatDescriptor();
    void __fastcall GLScene();
    GLfloat traduceX(int x);
    GLfloat traduceY(int y);
    void draw(long unsigned int x, GLfloat r,GLfloat g,GLfloat b);
public:           // User declarations
    __fastcall TFormPrincipal(TComponent* Owner);
};
//---------------------------------------------------------------------
-------
extern PACKAGE TFormPrincipal *FormPrincipal;
//---------------------------------------------------------------------
-------
#endif
```

C.10.3 GlSkel.cpp

```
//---------------------------------------------------------------------
-------
#include <vcl.h>
#include <fstream.h>
#include <excpt.h>
#include <fstream.h>
#pragma hdrstop

#include "GlSkel.h"
#include "param.h"
```

```
//---------------------------------------------------------------------------
-------
#pragma package(smart_init)    +
#pragma link "CGAUGES"
#pragma resource "*.dfm"

TFormPrincipal *FormPrincipal;
//---------------------------------------------------------------------------
-------
__fastcall TFormPrincipal::TFormPrincipal(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
-------
void __fastcall TFormPrincipal::FormCreate(TObject *Sender)
{
  hdc = GetDC(Handle);
  SetPixelFormatDescriptor();
  hrc = wglCreateContext(hdc);
  if (hrc == NULL)
     ShowMessage("Error CreateContex");
  if (wglMakeCurrent(hdc, hrc) == false)
     ShowMessage("Error MakeCurrent");
  glEnable(GL_LINE_STIPPLE);

  ColumDeDibujo=628;
  FilasDeDibujo=479;
  ColumEnPantalla=1024;
  FilasEnPantalla=528;

  ClientWidth = ColumDeDibujo;
  ClientHeight = FilasDeDibujo;
  //ClientWidth = 790;
  w = ClientWidth;
  h = ClientHeight;
  //ClientHeight = 528;
  nRange = 200;   // w/2 == h/2

  _left = -nRange;
  _right = nRange;
  _bottom = -nRange;
  _top = nRange;
  // podemos definir otros colores
  glClearColor(0.0f, 0.0f, 0.0f, 1.0f);


  FormResize(Sender);
  // crear objetos

  DespR=1;
  DespG=6;
  DespB=11;
}
//---------------------------------------------------------------------------
-------
void TFormPrincipal::SetPixelFormatDescriptor()
{
  PIXELFORMATDESCRIPTOR pfd = {
     sizeof(PIXELFORMATDESCRIPTOR),
```

```
     1,
     PFD_DRAW_TO_WINDOW | PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER,
     PFD_TYPE_RGBA,
     24,
     0,0,0,0,0,0,
     0,0,
     0,0,0,0,0,
     32,
     0,
     0,
     PFD_MAIN_PLANE,
     0,
     0,0,0 };
  int PixelFormat = ChoosePixelFormat(hdc, &pfd);
  SetPixelFormat(hdc, PixelFormat, &pfd);
}
//---------------------------------------------------------------------
-------
void __fastcall TFormPrincipal::FormResize(TObject *Sender)
{
  w = ClientWidth;
  h = ClientHeight;
  if (h==0) h=1.0;
  if (w==0) w=1.0;

  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  if (w <= h)
     {
    _left = -nRange;
     _right = nRange;
     _bottom = -nRange*h/w;
     _top = nRange*h/w;
     }
  else
     {
     _left = -nRange*w/h;
     _right = nRange*w/h;
     _bottom = -nRange;
     _top = nRange;
     }
  gluOrtho2D (_left, _right, _bottom, _top);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  GLScene();

}
//---------------------------------------------------------------------
-------
unsigned int bin2int(string cad)
  {
    unsigned int num = 0;
    unsigned int potencia = 1;   //2^0
    for (int i = (cad.size()-1);  i>=0; i-- ){
      if (cad[i]=='1') {
         num += potencia;
      }
      potencia *= 2;
    }
    return num;
```

```
  }


unsigned int sacaColor(string cad, int ini, int num){
   return bin2int(cad.substr(ini,num));
}

void DrawEjes()
{
  glBegin(GL_LINES);
    glColor3f(1.0,0.0,0.0);
    glVertex2i(-10,0);
    glVertex2i(10,0);
    glColor3f(0.0,1.0,0.0);
    glVertex2i(0,-10);
    glVertex2i(0,10);
  glEnd();
}

//---------------------------------------------------------------------
-------
GLfloat TFormPrincipal::traduceX(int x){
   GLfloat X_escalada;
   //Ahora la X e Y están en el rango correcto, -200 - 200
   //x -= 397;
   x -= (int)(ColumDeDibujo/2);
   //La X excalada esta en el rango 0 - ClientWidth
   //Ahora la X y la Y escaladas estan en el rango 0 - 400
   X_escalada = (x / (float)ClientWidth ) * (_right - _left);
   return X_escalada;
}

GLfloat TFormPrincipal::traduceY(int y){
   GLfloat Y_escalada;
   //La Y corregida es la Y escalada pero con el sentido de OpenGL
   GLfloat Y_corregida = -y;
   //Y_corregida -= 264;
   Y_corregida += (int)(FilasDeDibujo/2);
   //Ahora la X y la Y escaladas estan en el rango 0 - 400
   Y_escalada = (Y_corregida / (float)ClientHeight) * (_top -
_bottom);
   //Ahora la X e Y están en el rango correcto, -200 - 200
   //La Y tiene que tener en cuenta lo que ocupa el menú superior y el
titulo
   //y la X lo que ocupan los bordes de la ventana
   return Y_escalada;
}

int div(long unsigned int a, int b) {
   float re = floor(a/b);
   return (int)re;
}

void TFormPrincipal::draw(long unsigned int x, GLfloat r,GLfloat
g,GLfloat b){
   glBegin(GL_POINTS);
      glColor3f(r,g,b);
      //glVertex2f( traduceX(x % 795),traduceY(div(x,795)) );
      glVertex2f( traduceX(x %
ColumEnPantalla),traduceY(div(x,ColumEnPantalla)) );
   glEnd();
```

```
}
void __fastcall TFormPrincipal::GLScene()
{
   glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
   //DrawEjes();
   for (unsigned int i=0; i<pixmap.size(); i++) {

draw(pixmap.at(i).X(),pixmap.at(i).R(),pixmap.at(i).G(),pixmap.at(i).B
());
   }
   SwapBuffers(hdc);
}
//---------------------------------------------------------------------
-------

void __fastcall TFormPrincipal::FormPaint(TObject *Sender)
{
  GLScene();
}
//---------------------------------------------------------------------
-------

void __fastcall TFormPrincipal::FormDestroy(TObject *Sender)
{
  // liberar  objetos
  pixmap.clear();
  ReleaseDC(Handle,hdc);
  wglMakeCurrent(NULL, NULL);
  wglDeleteContext(hrc);
}
//---------------------------------------------------------------------
--------

void __fastcall TFormPrincipal::MenuCargarClick(TObject *Sender)
{
   double porcentaje = 0.0;
   int pos = 0;

   if(OpenDialog->Execute()){
    nombre_archivo = OpenDialog->FileName;
   };

   ifstream archivo(nombre_archivo.c_str());
   pixmap.clear();
   //leo la dirección inicial
   string cadena;
   archivo >> cadena;
   unsigned long int cont = 0;
   //int* valores = new int[32];
   //for(int g = 0; g<32; g++)
   //   valores[g] = 0;
   //entro en un bucle en el que voy guardando los pixeles
   while (archivo >> cadena) {
      pos++;
      if (pos%10000==0)
          porcentaje += 1000000.0/524288.0;  //numero de pixeles de
una imagen
      StatusBar->SimpleText = "abriendo " + nombre_archivo + " : " +
(int)porcentaje +"%";
      unsigned int r = sacaColor(cadena,DespR,5);
      unsigned int g = sacaColor(cadena,DespG,5);
```

```
      unsigned int b = sacaColor(cadena,DespB,5);
      //valores[b]++;
      pixel p;
      float r_IN = ((float)r)/32;
      float g_IN = ((float)g)/32;
      float b_IN = ((float)b)/32;
      /*
      if (r_IN>0.5) { r_IN = 1.0; }
      else { r_IN=0.0; }
      if (g_IN>0.5) { g_IN = 1.0; }
      else { g_IN=0.0; }
      if (b_IN>0.5) { b_IN = 1.0; }
      else { b_IN=0.0; }
      */

      p.set(cont,r_IN,g_IN,b_IN);
      pixmap.push_back(p);
      cont++;
   }
   archivo.close();
   //ofstream archivo2("valoresAzul.txt");
   //for(int g = 0; g<32; g++)
   //   archivo2 << valores[g] << "\n";
   //archivo2.close();
   //delete[] valores;
   GLScene();
   StatusBar->SimpleText = nombre_archivo;
}
//---------------------------------------------------------------------
-------

void __fastcall TFormPrincipal::MenuSalirClick(TObject *Sender)
{
   if (MessageDlg("¿Seguro que desea
salir?",mtConfirmation,TMsgDlgButtons() << mbYes << mbNo,0) == mrYes)
{
        ShowMessage("Tú te lo pierdes, insensato");
        Close();
        pixmap.clear();
   }
}
//---------------------------------------------------------------------
-------

void __fastcall TFormPrincipal::Parmetros1Click(TObject *Sender)
{
   FormParam->ShowModal();

   if (FormParam->ModalResult== mrOk){
      ColumDeDibujo=FormParam->colDib->Value;
      FilasDeDibujo=FormParam->filDib->Value;
      ColumEnPantalla=FormParam->colPant->Value;
      FilasEnPantalla=FormParam->filPant->Value;

      DespR=FormParam->despRojo->Value;
      DespG=FormParam->despVerde->Value;
      DespB=FormParam->despAzul->Value;

      ClientWidth = ColumDeDibujo;
      ClientHeight = FilasDeDibujo;
   }
```

```
}
//---------------------------------------------------------------------------
-------



void __fastcall TFormPrincipal::Escaladegrises1Click(TObject *Sender)
{
   //R*0.3+G*0.59+B*0.11
   float gris;
   float r,g,b;
   unsigned int tam = pixmap.size();
   for (unsigned int i=0; i<tam; i++) {
      StatusBar->SimpleText = "filtrando " + nombre_archivo + " a
escala de grises : " + (100*i)/tam +"%";
      r = pixmap.at(i).R();
      g = pixmap.at(i).G();
      b = pixmap.at(i).B();
      gris = r*0.25+g*0.5+b*0.125;
      pixmap.at(i).set(gris,gris,gris);
   }
   StatusBar->SimpleText = nombre_archivo;
   GLScene();
}
//---------------------------------------------------------------------------
-------

void __fastcall TFormPrincipal::Negativizar1Click(TObject *Sender)
{
   //R*0.3+G*0.59+B*0.11
   float r,g,b;
   unsigned int tam = pixmap.size();
   for (unsigned int i=0; i<tam; i++) {
      StatusBar->SimpleText = "filtrando " + nombre_archivo + " a
negativo : " + (100*i)/tam +"%";
      r = pixmap.at(i).R();
      g = pixmap.at(i).G();
      b = pixmap.at(i).B();
      pixmap.at(i).set(1.0-r,1.0-g,1.0-b);
   }
   StatusBar->SimpleText = nombre_archivo;
   GLScene();
}
//---------------------------------------------------------------------------
-------

void __fastcall TFormPrincipal::Binarizar1Click(TObject *Sender)
{
   float gris;
   float r,g,b;
   float nivel;
   unsigned int tam = pixmap.size();
   for (unsigned int i=0; i<tam; i++) {
      StatusBar->SimpleText = "filtrando " + nombre_archivo + " a
binario : " + (100*i)/tam +"%";
      r = pixmap.at(i).R();
      g = pixmap.at(i).G();
      b = pixmap.at(i).B();
      gris = r*0.25+g*0.5+b*0.125;
      if (gris>=0.5) {nivel=1.0;}
      else {nivel=0.0;}
```

```
      pixmap.at(i).set(nivel,nivel,nivel);
   }
   StatusBar->SimpleText = nombre_archivo;
   GLScene();
}
//---------------------------------------------------------------------
-------

void __fastcall TFormPrincipal::N8colores1Click(TObject *Sender)
{
   float r,g,b;
   float nivel;
   unsigned int tam = pixmap.size();
   for (unsigned int i=0; i<tam; i++) {
      StatusBar->SimpleText = "filtrando " + nombre_archivo + " a 8
colores : " + (100*i)/tam +"%";
      r = pixmap.at(i).R();
      g = pixmap.at(i).G();
      b = pixmap.at(i).B();
      if (r>=0.5) {r=1.0;}
      else {r=0.0;}
      if (g>=0.5) {g=1.0;}
      else {g=0.0;}
      if (b>=0.5) {b=1.0;}
      else {b=0.0;}
      pixmap.at(i).set(r,g,b);
   }
   StatusBar->SimpleText = nombre_archivo;
   GLScene();
}
//---------------------------------------------------------------------
-------

void __fastcall TFormPrincipal::Suavizado1Click(TObject *Sender)
{
   vector<pixel> pixmapAux;
   pixel** matriz;
   matriz = new pixel*[3];
   for (int j=0; j<3; j++) {
      matriz[j] = new pixel[3];
   }

   unsigned int i=0;
   unsigned int pos;
   unsigned int tam = pixmap.size();

   while (i<1025) {
      pixmapAux.push_back(pixmap.at(i));
      i+=1;
      StatusBar->SimpleText = "suavizando " + nombre_archivo + " : " +
(100*i)/(2*tam) +"%";
   }
   while (i<tam-1024) {
      pos = i-1025;
      matriz[0][0] = pixmap.at(pos);
      pos = i-1024;
      matriz[0][1] = pixmap.at(pos);
      pos = i-1023;
      matriz[0][2] = pixmap.at(pos);
      pos = i-1;
      matriz[1][0] = pixmap.at(pos);
```

```
      pos = i;
      matriz[1][1] = pixmap.at(pos);
      pos = i+1;
      matriz[1][2] = pixmap.at(pos);
      pos = i+1023;
      matriz[2][0] = pixmap.at(pos);
      pos = i+1024;
      matriz[2][1] = pixmap.at(pos);
      pos = i+1025;
      matriz[2][2] = pixmap.at(pos);

      float ROJO = 0.0;
      for (int a=0; a<3; a++) {
         for (int b=0; b<3; b++) {
            ROJO += matriz[a][b].R();
         }
      }
      ROJO /= 8;
      float VERDE = 0.0;
      for (int a=0; a<3; a++) {
         for (int b=0; b<3; b++) {
            VERDE += matriz[a][b].G();
         }
      }
      VERDE /= 8;
      float AZUL = 0.0;
      for (int a=0; a<3; a++) {
         for (int b=0; b<3; b++) {
            AZUL += matriz[a][b].B();
         }
      }
      AZUL /= 8;

      pixel aux(i,ROJO,VERDE,AZUL);
      pixmapAux.push_back(aux);
      i++;
      if ((i%1024) == 1022) {
         pixmapAux.push_back(aux);
         i+=1;
         pixmapAux.push_back(pixmap.at(i));
         i+=1;
         pixmapAux.push_back(aux);
         i+=1;
      }
      StatusBar->SimpleText = "suavizando " + nombre_archivo + " : " +
(100*i)/(2*tam) +"%";
   }

   pixmap.clear();
   for (unsigned int j=0; j<pixmapAux.size(); j++) {
      pixmap.push_back(pixmapAux.at(j));
      StatusBar->SimpleText = "suavizando " + nombre_archivo + " : " +
(100*(tam+j))/(2*tam) +"%";
   }

   for (int j=0; j<3; j++) {
      delete[] matriz[j];
   }
   delete[] matriz;
   StatusBar->SimpleText = nombre_archivo;
   GLScene();
```

```
}
//---------------------------------------------------------------------------
-------

void __fastcall TFormPrincipal::Bordes1Click(TObject *Sender)
{
   vector<pixel> pixmapAux;
   float** matriz;
   matriz = new float*[3];
   for (int j=0; j<3; j++) {
      matriz[j] = new float[3];
   }

   //ofstream archivo("pixel.txt");

   unsigned int i=0;
   int pos = 0;
   unsigned int tam = pixmap.size();

   while (i<1025) {
      pixmapAux.push_back(pixmap.at(i));
      i+=1;
      StatusBar->SimpleText = "obteniendo bordes de " + nombre_archivo
+ " : " + (100*i)/(2*tam) +"%";
   }
   while (i<tam-1024) {

      /*
      archivo << i << "  ,  "
              << pixmap.at(i).R() << " , "
              << pixmap.at(i).G() << " , "
              << pixmap.at(i).B() << " , " << "\n";
      */

      pos = i-1025;
      matriz[0][0] = (pixmap.at(pos).R() * 0.25)  +
                     (pixmap.at(pos).G() * 0.5)  +
                     (pixmap.at(pos).B() * 0.125);
      pos = i-1024;
      matriz[0][1] = (pixmap.at(pos).R() * 0.25)  +
                     (pixmap.at(pos).G() * 0.5)  +
                     (pixmap.at(pos).B() * 0.125);
      pos = i-1023;
      matriz[0][2] = (pixmap.at(pos).R() * 0.25)  +
                     (pixmap.at(pos).G() * 0.5)  +
                     (pixmap.at(pos).B() * 0.125);
      pos = i-1;
      matriz[1][0] = (pixmap.at(pos).R() * 0.25)  +
                     (pixmap.at(pos).G() * 0.5)  +
                     (pixmap.at(pos).B() * 0.125);
      pos = i;
      matriz[1][1] = (pixmap.at(pos).R() * 0.25)  +
                     (pixmap.at(pos).G() * 0.5)  +
                     (pixmap.at(pos).B() * 0.125);
      pos = i+1;
      matriz[1][2] = (pixmap.at(pos).R() * 0.25)  +
                     (pixmap.at(pos).G() * 0.5)  +
                     (pixmap.at(pos).B() * 0.125);
      pos = i+1023;
      matriz[2][0] = (pixmap.at(pos).R() * 0.25)  +
                     (pixmap.at(pos).G() * 0.5)  +
```

```
                    (pixmap.at(pos).B() * 0.125);
      pos = i+1024;
      matriz[2][1] = (pixmap.at(pos).R() * 0.25)  +
                     (pixmap.at(pos).G() * 0.5)  +
                     (pixmap.at(pos).B() * 0.125);
      pos = i+1025;
      matriz[2][2] = (pixmap.at(pos).R() * 0.25)  +
                     (pixmap.at(pos).G() * 0.5)  +
                     (pixmap.at(pos).B() * 0.125);


      float color = (8 * matriz[1][1]) -
                    matriz[0][0] -
                    matriz[0][1] -
                    matriz[0][2] -
                    matriz[1][0] -
                    matriz[1][2] -
                    matriz[2][0] -
                    matriz[2][1] -
                    matriz[2][2] ;

      pixel aux(i,color,color,color);
      pixmapAux.push_back(aux);
      i++;
      if ((i%1024) == 1022) {
         pixmapAux.push_back(aux);
         i+=1;
         pixmapAux.push_back(pixmap.at(i));
         i+=1;
         pixmapAux.push_back(aux);
         i+=1;
      }
      StatusBar->SimpleText = "obteniendo bordes de " + nombre_archivo
+ " : " + (100*i)/(2*tam) +"%";
   }

   pixmap.clear();
   for (unsigned int j=0; j<pixmapAux.size(); j++) {
      pixmap.push_back(pixmapAux.at(j));
      StatusBar->SimpleText = "obteniendo bordes de " + nombre_archivo
+ " : " + (100*(j+tam))/(2*tam) +"%";
   }

   for (int j=0; j<3; j++) {
      delete[] matriz[j];
   }
   delete[] matriz;
   StatusBar->SimpleText = nombre_archivo;
   GLScene();

}
//---------------------------------------------------------------------
-------
```

Apéndice D:

Forma de uso

Para visualizar imágenes y procesarlas hay que seguir la siguiente secuencia:

1. Obtener el archivo .PACA: Hay que convertir una imagen en formato .BMP a .PACA usando el programa BMP2PACA.

2. Cargar la imagen en la RAM: Para ello se baja el archivo arqimag3.bit a la FPGA mediante el GXSLoad. Esta arquitectura además de mostrar por pantalla el contenido de la RAM, permite cargar archivos .PACA a la RAM cuando se usa conjuntamente con el programa LoadMem. Por tanto, hay que arrancar también este programa y arrastrar hasta él la imagen transformada anteriormente. Para que se guarden los datos en la RAM hay que subir (poner a 1) el DIPSwitch1 cuando se vaya a descargar la imagen. Hay que mantener el DIPSwitch1 a 1 durante todo el proceso de descarga, esto es, desde antes de arrastrar una imagen nueva al LoadMem o darle a "Recargar", hasta que desaparece la frase "Descargando archivo ~.paca" del LoadMem.
   El arquimag3.bit tiene el reset en el Switch4.

3. Visualizar la imagen: El arqimag3.bit también visualiza los datos de la RAM. Para que se muestre la imagen en pantalla, simplemente hay que bajar (poner a 0) el DIPSwitch1.

4. Aplicar filtros: Cada filtro es un .BIT diferente pero el funcionamiento de todos es igual. Todos los filtros tienen el reset también en el Switch4, el botón de "acción" (para que comience a ejecutarse) en el Switch3 y el primer led como indicador de que ha finalizado.
   Para ejecutar un filtro, una vez que se haya descargado una imagen en la RAM se baja el archivo .BIT del filtro deseado, se pulsa el reset del filtro, a continuación se pulsa acción y se espera a que se encienda el led (si todo ha ido bien, se encenderá casi instantáneamente).
   Si se quiere volver a aplicar el filtro sobre la imagen obtenida, habrá que empezar de nuevo la secuencia, pulsando reset, luego acción y esperar a que se encienda el led.
   Para ver el resultado del filtro hay que volver a bajar el arquimag3.bit y dejar el DIPSwitch1 a 0 para que muestre el contenido de la RAM

NOTA: El arqimag3.bit realiza también un filtrado de blanco y negro al pulsar el Switch3.

9. Bibliografía

[AREN03] : *Formato .BMP*
Purificación Arenas Sánchez
http://www.fdi.ucm.es/profesor/puri/IG/bmp.pdf

[BAPP01] : "*Diseño e implementación de un computador autorreconfigurable*"
Iván Magán Barroso, Javier Basilio Pérez Ramas y Miguel Peón Quirós
Proyecto de Sistemas Informáticos, curso 2000-2001.

[DEHD02] : *Diseño e implementación de un computador autorreconfigurable*"
Del Cura, Hernández y Duque
Proyecto de Sistemas Informáticos, curso 2001-2002.

[INTE88] : *Hexadecimal Object File Format Specification*
intel, 1988
http://www.xess.com/faq/intelhex.pdf

[MEND01] : Transparencias "*LEDS, displays, LCDs y Monitores*"
José Manuel Mendías Cuadros
http://www.dacya.ucm.es/mendias/143/docs/vga.pdf

[VDBO01] : *XSV Flash Programming and Virtex Configuration*
D. Vanden Bout; XESS, July 5, 2001
http://www.xess.com/appnotes/an-070501-xsvflash.pdf

[XSVB01] : *XSV Board V 1.1 Manual*
XESS Corporation
http://www.xess.com/manuals/xsv-manual-v1_1.pdf